You are helping a client with some process automation. They have managed to get their website landscape and deployment process encapsulated in a large CloudFormation template. They have recently contracted with a third-party service to provide some automated UI testing. To initiate the test scripts, they need to make a call out to an external REST API. They would like to integrate this into their existing CloudFormation template but not quite sure of the best way to do that. Help them decide which of the following ideas is feasible and incurs the least extra cost.

1. Include an SQS queue definition in the CloudFormation template. Define a User Script on the deployed EC2 instance which will insert a message into the SQS queue only once it has fully booted. Configure the external REST API to use long polling to check the queue for new messages in order to initiate the testing process.
2. Create a Lambda function which issues a call out to the external REST API using the POST method. Define a custom resources in the CloudFormation template and associate the Lambda function and execution role with the custom resource. Include DependsOn to ensure that the function is only called after the other instances are ready.
3. Add an API Gateway deployment to the CloudFormation template. Add the DependsOn parameter to the API Gateway resource to ensure that the call to the external API only happens after all the other resources have been created. Create a POST method and define it as a proxy for the external REST API endpoint. Using SWF, call the API Gateway endpoint to trigger the testing process.
4. Add a small EC2 instance definition to the CloudFormation template. Define a User Script for that instance which will install a custom application from S3 to call out to the external REST API endpoint using the POST method to trigger the testing process. Add a CleanUp parameter to the EC2 instance definition that will shut down the instance once the activity has completed.

Answer: b

EXPLANATION:

To integrate external services into a CloudFormation template, we can use a custom resource. Lambda makes a very good choice for this scenario because it can handle some logic if needed and make a call out to an external API. Using an EC2 instances to make this call is excessive and we likely would not have the ability to configure the third-party API to poll an SQS queue.

Your company has an online shopping web application. It has adopted a microservices architecture approach and a standard SQS queue is used to receive the orders placed by the customers. A Lambda function sends orders to the queue and another Lambda function fetches messages from the queue and processes them. On some occasions the message in the queue cannot be handled properly. For example, when an order has a deleted production ID, the message cannot be consumed successfully and is returned to the queue. The problematic messages in the queue keep growing and the ability to process normal messages is affected. You need a mechanism to handle the message failure and isolate error messages for further analysis. Which method would you choose?

1. Decrease the message retention period of the queue to 1 day. When the messages are not processed properly and put back in the queue, they can be quickly deleted when the retention period expires.
2. Modify the error handling logic of the Lambda function to delete the messages whenever the processing is unsuccessful with an error or exception. The error messages do not return to the queue and the normal message handling is not blocked.
3. Create a standard queue as the dead letter queue and configure a redrive policy to put error messages to the dead letter queue. Analyze the contents of messages in the dead letter queue to diagnose the issues.
4. Create a FIFO (First-In-First-Out) queue as the dead letter queue and use a redrive policy to forward problematic messages to this new queue. Create a Lambda function to read the message contents in the FIFO queue for further analysis.

Answer: c

EXPLANATION:

It is not a good idea to adjust the retention period or simply delete the messages that fail to be processed as the question asks for a mechanism to isolate the messages for further troubleshooting. A redrive policy should be used to auto-forward error message to a dead letter queue. Then you can analyse the contents of messages to diagnose the producer’s or consumer’s issues. One thing to note is that a standard queue can only have another standard queue as the dead letter queue. Therefore a FIFO dead letter queue is incorrect as this scenario uses a standard SQS queue and requires a standard dead letter queue.

Quality Auto Parts, Inc. has installed IoT sensors across all of their manufacturing lines. The devices send data to both AWS IoT Core and Amazon Kinesis Data Streams. Kinesis Data Streams triggers a Lambda function to format the data, and then forwards it to AWS IoT Analytics to perform monitoring and time-series analyses, and to take actions based on business processes. After an equipment failure on one of the manufacturing lines causes tens of thousands of dollars in revenue losses, it's determined that alarms for a specific piece of equipment where received seventy-five seconds after the issue originated, and that automated corrective action within a few seconds of the problem could have avoided the financial losses altogether. What changes should be made to the architecture to improve the latency of device alerts?

1. Create an AWS IoT Core rule to write the message to Amazon CloudWatch Alarms to detect anomalies in the data. Invoke another AWS Lambda function from CloudWatch Alarms to perform device corrective action when needed.
2. Add Amazon Kinesis Data Analytics as a second consumer of the Kinesis Data Stream to detect anomalies in the data. Invoke another AWS Lambda function from Kinesis Data Analytics to perform device corrective action when needed.
3. Create an AWS IoT Core rule to write the message to Amazon Kinesis Data Analytics to detect anomalies in the data. Invoke another AWS Lambda function from Kinesis Data Analytics to perform device corrective action when needed.
4. Add another AWS Lambda function as a second consumer of the Kinesis Data Stream to detect anomalies in the data. Have the Lambda function write the anomalies to Amazon DynamoDB and perform device corrective action when needed.

Answer: b

EXPLANATION:

AWS IoT Analytics is useful for understanding long-term device performance, performing business reporting, and identifying predictive fleet maintenance needs, but common latencies run from seconds to minutes. If you need to analyze IoT data in real-time for device monitoring, use Kinesis Data Analytics, which provides latencies in the millisecond to seconds range. A Lambda function can be used as the destination for Kinesis Data Analytics to perform corrective actions. IoT Core rules can write messages to a Kinesis stream, but not directly to Kinesis Data Analytics. Having a Lambda function perform anomaly detection will work, but will require more logic to be written for query setup and execution than using a specialized service like Kinesis Data Analytics. With Amazon CloudWatch Alarms, an alarm will watch a single metric over a period time, but will not provide the capabilities of SQL to detect complex anomaly conditions.

Due to new corporate policies on data security, you are now required to use encryption at rest for all data. You have some EC2 Linux instances on AWS that were created without encryption for the root EBS volume. What can you do that meet the requirement and reduce administrative overhead?

1. At present, EC2 does not support encrypted root volumes. Create new encrypted EBS data volumes and attach the new volumes to the existing instances. Use RSYNC to migrate all the non-OS data over to the encrypted data volumes.
2. Stop the instances and create AMIs from the instances. Copy the AMIs to the same region and select the "Encrypt target EBS snapshots". Redeploy the instances using the AMI copies you made with encrypted root volumes.
3. Stop the instances and temporarily detach the EBS volumes. Attach the root volumes to another EC2 instance and mount them a data volume. Use a encryption tool like GPG or OpenPGP to recursively encrypt all the files on the mounted root volumes. Detach and reattach the encrypted EBS volumes to the original instances and restart. Import the encryption keys in KMS as a CMK.
4. Create an encrypted EFS instance and mount-points in the respective subnets. Log into the instance and mount an encrypted EFS mount-point. Copy all the root files over to the EFS mount point. Edit the FSTAB file to mount the EFS mount point as the root volume instead of the root EBS device and reboot.
5. Create a certificate in CMS for the encryption key. Stop the instances and temporarily detach the root volumes. Via the AWS CLI, enable encryption on the root volumes using the "ebs modify-volume" argument with the flag of "encryption=<CMS ARN>" to specify the certificate.

Answer: b

EXPLANATION:

AWS does support encrypted root volumes but conversion from unencrypted root to an encrypted root requires a bit of a process. You must first create an AMI then copy that newly created AMI to the same region, specifying that you want to encrypt the EBS volumes during the copy. You can then create a new instance with an encrypted root volume from the copied AMI. You can use either a generated key from KMS or your own CMK imported into KMS.

You have been asked to help develop a process for monitoring and alerting staff when malicious or unauthorized activity occurs. Your Chief Security Officer is asking for a solution that is both fast to implement but also very low maintenance. Which option best fits these requirements?

1. Use AWS Glue to direct all CloudTrail logs into Redshift. Use QuickSight as a presentation layer with custom reports for visualizing malicious and unauthorized behavior. Run the reports periodically and email them to the Security Officer.
2. Configure VPC Flow Logs to capture all traffic going in and out of the VPC. Use ElastiSearch to process the logs and trigger a Lambda function whenever malicious or unauthorized behavior is found.
3. Enable AWS GuardDuty to monitor for malicious and unauthorized behavior. Configure a custom blacklist for the IPs which you have seen suspect activity in the past. Setup a Lambda function triggered from a CloudWatch event when anomalies are detected.
4. Use AWS SageMaker to implement a Linear Learner algorithm that periodically reviews CloudFront logs for malicious and unauthorized behavior. When the ML model finds something suspicious, trigger an SES email to the Security Officer.
5. Configure CloudWatch to create an event whenever malicious or unauthorized behavior is observed. Trigger an SMS message via SNS to the Security Officer whenever the event happens.
6. Enable AWS Macie to monitor for malicious and unauthorized behavior. Configure a custom whitelist for the IPs that were wrongly flagged. Setup a Lambda function triggered from a CloudWatch event when anomalies are detected.

Answer: c

EXPLANATION:

AWS GuardDuty is a managed service that can watch CloudTrail, VPC Flow Logs and DNS Logs, watching for malicious activity. It has a build-in list of suspect IP addresses and you can also upload your own lists of IPs. GuardDuty can trigger CloudWatch events which can then be used for a variety of activities like notifications or automatically responding to a threat. AWS Macie is a service to discovery and classify potentially sensitive information. CloudWatch alone lacks the business rules that are provided with GuardDuty.

You are setting up a corporate newswire service for a global news company. The service consists of a REST API deployed on EC2 instances where customers can retrieve the latest news articles in real-time that happen to contain their company name. This allows companies to monitor all news sources for stories where they are mentioned. Because of the worldwide reach of the new site, you want to position servers around the globe. You want to publish one subdomain name globally (api.domain.com) and have the requesters directed to the nearest region based on latency. In each region, you want to be able to accommodate blue-green deployments without downtime as well. What steps do you take?

1. Using Route 53, we first create the top-level api.domain.com with a geolocation policy. We then create latency-based routing records for the instances in each region (us-east-2.api.domain.com). Next, we configure the countries closest to each region in the geolocation policy to direct them to the regional records.
2. We would first create geospatial records for the local resources in each region (us-east-2.api.domain.com) and assign equal weights. Next, we create latency-based routing records for the top level subdomain (api.domain.com) and direct those to the regional records as an alias. We must also disable Health Check on the latency record to ensure the localized Health Check is used.
3. First setup weighted routing records for the local instances in the region in Route 53. Assign equal weights with all sharing the same regional subdomain name (us-east-2.api.domain.com). Next, create latency alias records by creating multiple entries for api.domain.com--each pointing to the regional subdomains.
4. Use CloudFormation to create a distribution of the website. Create an alias record for the subdomain (api.domain.com) in Route 53 and assign it to the CloudFront distribution. To ensure no lag in news retrieval, set the maximum TTL on the CloudFront distribution to 0.

Answer: c

EXPLANATION:

We want to use weighted routing records for local instances so we have the ability to adjust weights and shift traffic during blue-green deployments. Latency-based routing would take care of funneling requests to the site with the lowest latency.

You are working with a client to help them design a future AWS architecture for their web environment. They are open with regard to the specific services and tools used but it needs to consist of a presentation layer and a data store layer. In a brainstorming session, these options were conceived. As the consulting architect, which of these would you consider feasible?

1. Deploy Kubernetes on an auto-scaled group of EC2 instances. Define pods to represent the multiple tiers of the landscape. Use ElastiCache for Memcached to offload queries from a Multi-AZ RDS instance. To deploy changes to the landscape, create a new EKS deployment containing all the updated service containers and deploy them to replace all the previous existing tiers. Ensure the DevOps team understands the rollback procedures.
2. Use the AngularJS framework to create a single-page application. Use the API Gateway to provide public access to DynamoDB to serve as the data layer. Store the web page on S3 and deploy it using CloudFront. When changes are required, upload the new web page to S3. Use S3 Events to trigger a Lambda function which expires the cache on CloudFront.
3. Create a monolithic architecture using Elastic Beanstalk configured in the console. Create an RDS instance outside the Beanstalk environment and configure it for multi-AZ availability. When a new landscape change is required, use a command line script to implement the change.
4. Deploy an auto scaling group of EC2 instances behind an Application Load Balancer. Provision a Mulit-AZ RDS instance to act as the data store, configuring a caching layer to offload queries from the database. Use a User Script in the AMI definition to download the latest web assets from S3 upon boot-up. When changes are required, use AWS Config to automatically fetch a new version of web content from S3 when a new version is created.
5. Setup a traditional three tier architecture with a CloudFormation template per tier and one master template to link in the others. Configure a CodeBuild stack and set this stack to perform automated Blue Green deployments whenever any code change is made.

Answer: b, c

EXPLANATION:

The only two options which contain feasible options are the Beanstalk and S3/Dynamo methods. One would not create a new K8s deployment for for a new web update. CodeBuild and AWS Config are not the correct tools for how they are being suggested.

You are designing a workflow that will handle very confidential healthcare information. You are designing a loosely coupled system comprised of different services. One service handles a decryption activity using a CMK stored in AWS KMS. To meet very strict audit requirements, you must demonstrate that you are following the Principle of Least Privilege dynamically--meaning that processes should only have the minimal amount of access and only precisely when they need it. Given this requirement and AWS limitations, what method is the most efficient to secure the Decryption service?

1. In the step right before the Decryption step, programmatically apply a grant to the CMK that allows the service access to the CMK key. In the step immediately after the decryption, explicitly revoke the grant.
2. The current AWS platform services are not well suited for implementing Principle of Least Privilege in a dynamic manner. Consider a different design that makes use of a more monolithic architecture rather than services.
3. Use a grant constraint to deny access to the key except for the service account that is running the workflow processes. Enable CloudTrail alerts if any other role attempts to access the CMK.
4. Create a IAM key policy that explicitly allows access to the CMK and assign that to a role. Assign the role to the process that is executing the Decryption service. At the end of the day, programmatically revoke that role until the start of the next day.
5. Create an IAM key policy that explicitly denies access to the Decryption operation of the CMK. Assign that policy to a role that is then assigned to the process executing the Decryption service. Use a Lambda function to programmatically remove and add the IAM policy to the role as needed by the decryption process.

Answer: a

EXPLANATION:

Grants in KMS are useful for dynamically and programmatically allowing a process the ability to use the key then revoking after the need is over. This is more efficient than manipulating IAM roles or policies.

A beach apparel company has begun an initiative to improve their sales analytics capabilities using AWS services. They'll need to be able to visualize summary sales data by product line, territory, and sales channel for each day, month, and year, and they'll need to be able to drill-down with ad-hoc queries on individual sales records. There are multiple data sources that provide transactional information in different formats. The company has chosen Amazon QuickSight as their visualization tool for the summary information. Visualizations and drill-down queries will require three years of rolling sales history, which estimates to seven petabytes of data. Which architecture will provide the best performance and cost efficiency?

1. Ingest individual sales transactions from each data source into Amazon S3 with Amazon Kineses Data Firehose. Trigger an AWS Lambda function to format the transaction data in a standard way and redeposit the results in S3. Run AWS Glue jobs to aggregate the summary data into Amazon Redshift
2. Read detailed sales transactions from each data source with Amazon Kinesis Data Firehose and load them into Amazon Redshift. Run AWS Glue jobs to format the transaction data in a standard way and perform aggregate functions to write the data into summary tables in Redshift
3. Use Amazon Kinesis Data Analytics to format the data source transactions in a standard way and load it into Amazon Aurora. Invoke Lambda functions to aggregate the data and write it into summary tables in Aurora
4. Read detailed sales transactions from each data source with Amazon Kinesis Data Streams and write them to Amazon Elastic Block Store on EC2 instances in Auto Scaling Groups. Perform data format standardization and summary aggregation on EC2, and write the summary results to Amazon Redshift tables

Answer: a

EXPLANATION:

Using S3 to store the detailed sales transaction data and using Lambda to standardize data formats is the most cost effective option. Storing the summary data in Redshift provides a high performance option for reads from QuickSight, and keeping the detailed transaction data out of Redshift allows for smaller node sizes and lower cost. Amazon Redshift Spectrum can be used for drill-down queries that join tables from both Redshift and S3. For answer number two, Redshift will be a better option than Aurora for OLAP query performance due to it's columnar organization. Answer number four provides no simple way to perform ad-hoc drill down queries.

A popular royalty free photography website has decided to run their business on AWS. They receive hundreds of images from photographers each week to be included in their catalog. Amazon S3 has been selected as the image repository. As the business has grown, the task of creating catalog entries manually has become unsustainable. They'd like to automate the process and store the catalog information in Amazon DynamoDB. Which architecture will provide the most scalable solution for automatically adding content to their image catalog going forward?

1. Programmatically call the S3 API to upload the images. Trigger an AWS Lambda function to send the image's S3 key to AWS Elemental MediaStore, which will extract the image's metadata, discover image patterns through machine learning, and deposit artifacts back into S3. Invoke a Lambda function to write the artifact data to DynamoDB.
2. Deploy Amazon Kinesis Data Firehose to ingest images into S3. Invoke a Lambda function to pass the image's S3 key to Amazon Rekognition, which will extract the image metadata and detect objects in the image. Invoke a Lambda function to store the discovered data in DynamoDB.
3. Deploy Amazon Kinesis Data Streams to ingest the images with two consumers. Setup Amazon Kinesis Firehose as the first consumer to deposit the images into S3. Configure Amazon Kinesis Video Analytics as the second consumer to extract the image's metadata and object information. Invoke a Lambda function to store the discovered information in DynamoDB.
4. Programmatically call the S3 API to upload the images. Trigger an AWS Lambda function to kick off execution of a state machine in AWS Step Functions. Create state machine sub-steps to invoke Lambda functions which extract image metadata, detect objects in the image with Amazon Rekognition, and store the discovered data in DynamoDB.

Answer: d

EXPLANATION:

Calling the S3 API to upload the images will suffice for this use case. Streaming ingest is not needed for this volume of data. AWS Step Functions will orchestrate the process of discovering both the image metadata with a Lambda function and the image object data with Rekognition. Rekognition will not return the image metadata. AWS Elemental MediaStore is used for originating and storing video assets for live or on-demand media workflows, not image recognition. Kinesis Video Analytics is not a currently supported service.

You have built an amazing new machine learning algorithm that you believe would be of benefit to many paying business customers. You want to expose it as a REST API to your customers and offer three different consumption levels: Silver, Gold and Platinum. The backend is completely serverless using Lambda functions. What is the most efficient and least cost way to make your API available for paying customers with a per-request pricing model?

1. Deploy your API to an S3 bucket using the Static Hosting feature. Enable "requester pays" for the bucket to handle billing. Create a serverless customer portal that will allow customers to register for API access and dynamically create an IAM role for them using Lambda.
2. Deploy your API using API Gateway using the "managed-service" mode. Use AWS Batch to export usage logs to S3. Use AWS Glue to aggregate and transform the raw logs into daily usage and save in DynamoDB. Build a Payment Gateway using the AWS SDK to read the DynamoDB billing table and prepare invoices for customers. Use SES to email invoices to customers.
3. Port your Lambda functions over to a Docker container and deploy using EKS. Setup metered usage for each customer you expect to subscribe and deploy unique API keys to those customers. Use CloudTrail to generate usage data for the API containers and import into RedShift for aggregation and processing. Use the Amazon Pay API to issue invoices to customers based on monthly queries of the RedShift data.
4. Setup your own API Gateway Serverless Developer Portal to create API keys for subscribers. Register as a seller with AWS Marketplace and specify the usage plans and developer portal. Submit a product load form with a dimension named "apigateway" of the "requests" type. Create a metering IAM role to allow metrics to be sent to AWS Marketplace. Associate your provided Product Code with the corresponding usage plan.
5. Use API Gateway to configure a usage plan for the production stage of the API. Register as a seller with AWS Marketplace and define three different levels of service and pricing. Assign the respective product code to the proper usage plan in the API Gateway console.

Answer: d

EXPLANATION:

Since 2016, AWS has allowed developers to monetize their APIs in AWS Marketplace using API Gateway. The developer must first create a Developer Portal to provide a method for customers to register for access and then associate the assigned Product Code, received when the developer registers the API in the Marketplace, to the desired usage plan within API Gateway. AWS then handles accounting for the usage and billing.

A composite materials company is implementing a new monitoring solution on their manufacturing floor. Wi-Fi enabled IoT devices will be registered with AWS IoT Core to read data from numerous control systems. Dashboards will be created in Amazon QuickSight to present aggregate metrics to users (average, min, max, standard deviation, variance, and percentile). Drill down capabilities will also be needed for deeper analyses of exception scenarios. Which architecture will provide the most reliable and performance efficient solution for the company's monitoring needs?

1. Install HTTP libraries on the IoT devices. Create an IoT Core rule that forwards the HTTP messages to an Amazon Kineses Data Firehose stream, which deposits the data into S3, and writes the data to an Amazon Kinesis Data Analytics stream to aggregate the data. Have an AWS Lambda function trigger to read the aggregate data and deposit it into S3.
2. Install HTTP libraries on the IoT devices. Create an IoT Core rule that forwards the HTTP messages to an AWS Lambda function. Have the Lambda function write the messages to S3, and to an Amazon Kinesis Data Analytics stream to aggregate the data. Have an AWS Lambda function trigger to read the aggregate data and deposit it into Amazon DynamoDb tables
3. Install MQTT libraries on the IoT devices. Create an IoT Core rule that forwards the MQTT messages to an Amazon Kineses Data Analytics stream, which writes aggregate data to an Amazon Kinesis Data Streams stream. Have an AWS Lambda function trigger to read the aggregate data and deposit it into Amazon DynamoDB tables
4. Install MQTT libraries on the IoT devices. Create an IoT Core rule that forwards the MQTT messages to an AWS Lambda function. Have the Lambda function write the messages to an Amazon Kinesis Data Firehose stream, which deposits them into S3

Answer: d

EXPLANATION:

The MQTT protocol is a publish/subscribe protocol that provides clients with independent existence from one another, enhancing the reliability of the solution. HTTP is a document-centric ,request-response protocol, requiring more processing and storage overhead for IoT devices. There is no need to use Kinesis Data Analytics in this case because QuickSight can perform all of the aggregate functions required for this use case. Answer number four won't allow for data drill down because the device messages are not written to any persistent storage service.

You are working with a customer to implement some better security policies. They have a group of remote employees working on a confidential project that uses some proprietary Windows software and stores data in S3. The Chief Information Security Officer is concerned about the threat of the desktop software or confidential data being smuggled out to a competitor. What architecture would you recommend to best address this concern?

1. Provision Amazon Workspaces in a secured private VPC. Do not enable Internet access for the Workspaces. Create a VPC Gateway Endpoint to S3 and implement an endpoint policy that explicitly allows access to the required bucket. Assign an S3 bucket policy that denies access unless the sourceVpce matches the VPC endpoint. Supply the users with instructions on downloading and login into the Workspaces instances.
2. Create a bucket policy using the sourceIP condition to only allow access from a specific VPC CIDR. Apply a NACL which only permits inbound port 22 and outbound ephemeral ports. Deploy Amazon Workspaces in the VPC and disable internet access. Supply the users with instructions on downloading and login into the Workspaces instances.
3. Provision Windows 2016 instances in a private subnet. Create a specific security group for the Windows machines permitting only SSH inbound. Create a NACL which allows traffic to S3 services and explicitly deny all other network traffic to and from the subnet. Assign an S3 bucket policy that only allows access for members of the Windows machine security group.
4. Use Service Catalog to deploy and manage the proprietary Windows software to the remote employees. Create an OpenVPN server instances within a VPC. Create an VPC Interface Endpoint to S3 and use a security group to only permit traffic from the OpenVPN server security group. Supply the remote employees with instructions to install and login using OpenVPN client software.

Answer: a

EXPLANATION:

Using a locked down virtual desktop concept would be the best way to manage this. AWS WorkSpaces provides this complete with client software to log into the desktops. These Workspaces can be walled off from the Internet. Using policies, you could allow access from only those in the Workspaces VPC.

A global digital automotive marketplace is using Lambda@Edge function with CloudFront to redirect incoming HTTP traffic to custom origins based on matching custom headers or client IP addresses with a list of redirection rules. The Lambda@Edge function reads these rules from a file, rules.json, which it fetches from an S3 bucket. The file changes every day because several teams in the company uses the file for different purposes, including but not limited to, (a) the security team uses the file to honeypot potential malicious traffic (b) the engineering team uses the file to do A-B testing on new features, (c) the product team experiments with new mobile platforms by redirecting traffic from a specific kind of mobile device to a specific set of server farms, etc.. As a result, the file can be as big as 200 KB. Recently, the response time of the website has degraded. On investigation, you have found that this Lambda@Edge function is taking too long to fetch the rules.json file from the S3 bucket. The existing CI-CD pipeline deploys the file to a versioning-enabled S3 bucket when any change is committed to source control. Any change in rules.json must reflect within 1 hour at all Cloudfront Edge locations. Select two options from the ones below that will not work in improving the latency of fetching this file?

1. Define a separate cache behaviour for \*.json in your Cloudfront web distribution, setting the origin as the S3 bucket. Change the Lambda@Edge function code to use the Cloudfront download URL instead of downloading the file directly from S3. This way, the file will be cached by Cloudfront avoiding expensive round trip time to S3 each time. Set the Cloudfront TTL to 45 minutes.
2. Reconfigure the S3 bucket as a static website. Use the website endpoint to download the file instead of directly accessing the bucket from the Lambda@Edge function. This will cause HTTP GET requests to be cached by S3, thus improving the latency of fetching the file
3. Change the Lambda@Edge code to save the contents of the rules.json file in a global variable so that it is cached in Lambda@Edge memory, with a TTL of 55 minutes, persisted between invocations. Lambda@Edge guarantees persistence of variables in memory between invocations.
4. Include the rules.json file in the Lambda@Edge deployment package. Change the CI-CD pipeline to deploy a new Lambda@Edge version every time the file changes. Change the Lambda@Edge function code to read the file locally instead of reading it from S3. This will improve the latency of fetching the file.

Answer: b, c

EXPLANATION:

A key to answering this question is to not miss the fact that it asks which two of the answers will not help. AWS SA-P exam can occasionally frame the question with a not. Also, knowledge of how Lambda@Edge functions work with CloudFront is important for the exam.\n\nThere will be no improvement in the fetching time if we reconfigure the S3 bucket as a static website. In fact, doing so might add a layer of redirection during routing.\n\nLambda@Edge does not guarantee the persistence of global variables in memory between invocations. While it might be possible to use global variables for a short time as cache, provided the code does not make any assumptions about the guarantee of persistence, it is a bad idea to solely depend on Lambda@Edge memory between invocations. AWS does not guarantee using the same container instance for any number of requests, though it will try to re-use a warmed up instance for the same function invocation landing on the same edge node. If it is re-using the same container instance from the one used by the last Lambda@Edge function, the global variable trick will work. However, as the option clearly says that such usage is guaranteed (which is false and will not work), it is one of the answer choices to select in this case.

You are helping a company design a fully cloud-based Customer Service application. Over 50% of their Customer Service Representatives are remote and that number increases and decreases seasonally. They need the ability to handle inbound and outbound calls as well as chatbot capabilities. Additionally, they want to provide a self-service option using interactive voice response to customers who do not need to speak to a person. Which design is feasible and makes most efficient use of AWS services?

1. Setup AWS Connect for inbound and outbound calling. Make use of Polly and Lex for interactive voice response components. Create a standard Customer Service Rep desktop and deploy using AWS Workspaces. Leverage Lex to create a chatbot component.
2. Setup Twilio with Lambda to manage inbound and outbound calling. Create a standard Customer Service Rep desktop Windows AMI and deploy via Service Catalog. Leverage Polly for creating a chatbot and Translate for an interactive voice response system.
3. Create a standardized Customer Service Rep desktop and deploy via CloudFront. Use Translate and AWS Connect to create a chatbot component. Leverage Polly to create an interactive voice response component. Use Alexa for Business for the inbound and outbound calling.
4. Use AWS Comprehend to create the chatbot and interactive voice response components. Use Asterisk PBX from AWS Marketplace to handle the inbound and outbound calling. Create a standardized Customer Service Rep desktop and deploy using Service Catalog.
5. Create a standard Customer Service Rep desktop and deploy using AWS Workspaces. Setup AWS Connect for inbound and outbound calling. Leverage Alexa for Business to create chatbot and interactive voice response components. Store call logs in Redshift and analyze using Quicksight.

Answer: a

EXPLANATION:

AWS Connect is Amazon's "call center in a box" solution that enabled interactive voice response with Lex and inbound and outbound calling. Additionally, you can use Lex to build a chatbot. AWS Workspaces is a managed DaaS that is we suited for deploying to remote workers.

The security monitor team informs you that two EC2 instances are not compliant reported by an AWS Config rule and the team receives SNS notifications. They require you to fix the issues as soon as possible for security concerns. You check that the Config rule uses a custom Lambda function to inspect if EBS volumes are encrypted using a key with imported key material. However, at the moment the EBS volumes in the EC2 instances are not encrypted at all. You know that the EC2 instances are owned by developers but you do not know the details about how the instances are created. What is the best way for you to address the issue?

1. Import a new key material to an existing Customer Managed Key (CMK) in KMS. Create an AMI from the EC2 instance. Then launch a new EC2 instance from the AMI. Encrypt the EBS volume in the new instance. Terminate the old instance after the new one is in service.
2. Modify the AWS Managed Key (AWS/EBS) in KMS to include an imported key material. Create a snapshot of the EBS volume. Then create a new volume from the snapshot with the volume encrypted. Detach the original volume and attach the new encrypted EBS to another device name of the instance.
3. Create a Customer Managed Key (CMK) in KMS with imported key material. Create a snapshot of the EBS volume. Copy the snapshot and encrypt the new one with the new CMK. Then create a volume from the snapshot. Detach the original volume and attach the new encrypted EBS to the same device name of the instance.
4. Create a new EBS key from CloudHSM with imported key material. Create a new EBS volume encrypted with the new key. Attach the volume to the EC2 instance. Use Linux dd command to copy data from non-encrypted volume to encrypted volume. Unmount the old volume after the sync is complete.

Answer: c

EXPLANATION:

The key must have imported key material according to the AWS Config rule. It should be a new key created in KMS. Existing KMS cannot import a new key material and AWS Managed Key such as aws/ebs cannot be modified either. CloudHSM is more expensive than KMS and is not required in this scenario. Besides, when the new encrypted EBS volume is attached, it should be attached to the same device name such as /dev/xvda1.

An application in your company that requires extremely high disk IO is running on m3.2xlarge EC2 instances with Provisioned IOPS SSD EBS Volumes. The EC2 instances have been EBS-optimized to provide up to 8000 IOPS. During a period of heavy usage, the EBS volume on an instance failed, and the volume was completely non-functional. The AWS Operations Team restored the volume from the latest snapshot as quickly as possible, re-attached it to the affected instance and put the instance back into production. However, the performance of the restored volume was found to be extremely poor right after it went live, during which period the latency of I/O operations was significantly high. Thousands of incoming requests timed out during this phase of poor performance.\n\nYou are the AWS Architect. The CTO wants to know why this happened and how the poor performance from a freshly restored EBS Volume can be prevented in the future. Which answer best reflects the reason and mitigation strategy?

1. The latest snapshot did not have the most current data. It only had the data from the last time a snapshot was taken. The requests timed out because of this data gap. To mitigate this, increase the frequency of taking EBS snapshots.
2. A freshly restored EBS Volume needs pre-warming to activate the inbuilt caching mechanism. To fix this, update the restoration process to run the set-up-cache command on the freshly restored EBS Volume first before the instance is put back in production. Also, include random I/O tests to ensure that desired I/O levels are reached before putting the instance back to production.
3. A freshly restored EBS Volume cannot utilize EBS Optimization Instances straight away, as the network traffic and EBS traffic traverse the same 10-gigabit network interface. Only after the entire volume is scanned by an asynchronous process, EBS Optimization kicks in. This increases the I/O latency until the volume is ready to utilize EBS Optimization. To fix this, update the restoration process to wait and run random I/O tests on a freshly restored EBS Volume. Put the instance back to production only after the desired I/O levels are reached.
4. When a data block is accessed for the first time on a freshly restored EBS Volume, EBS has to download the block from S3 first. This increases the I/O latency until all blocks are accessed at least once. To fix this, update the restoration process to run tools to read the entire volume before putting the instance back to production.

Answer: d

**EXPLANATION:** Data gap cannot be the reason for high disk I/O latency. Whether the data being requested is on the disk or not cannot be responsible for the extended period of high disk I/O latency, as all operating systems index the contents in some way. They do not scan the whole disk to conclude that something is missing. Hence, the choice that suggests data gap as the reason is eliminated.\n\nEBS Optimization works straight away after a freshly restored volume is attached to an EBS optimized instance. Hence, the choice that suggests that EBS Optimization takes some time to kick in is eliminated.\n\nThere is nothing called set-up-cache command. The option that suggests that there is an inbuilt caching mechanism that needs to be activated is completely fictional, and is eliminated.\n\nThe only correct option is the one that correctly states that every new block read from a freshly restored EBS Volume must first be downloaded from S3. This is because EBS Snapshots are saved in S3. Remember that EBS Snapshots are incremental in nature. Every time a new snapshot is taken, only the data that changed is written to that particular snapshot. Internally, it maintains the pointers to older data that was written to S3 as part of previous snapshots. These blocks of data continue to reside on S3 even after an EBS Volume is restored, and is read the first time they are accessed. Linux utilities like dd or fio can be used after restoring an EBS Volume to read the whole volume first to get rid of this latency problem when the instance is put back in production.

As the solution architect, you are assisting your customer design and develop a mobile application using API Gateway, Lambda and DynamoDB. S3 buckets are being used to serve static content. The API created using API Gateway is protected by WAF. The development team has just staged all components to the QA environment. They are using a load testing tool to generate short bursts of a high number of concurrent requests sent to the API Gateway method. During the load testing, some requests are failing with a response of 504 Endpoint Request Timed-out Exception. What is one possible reason for this error response from API Gateway endpoint?

1. The test is triggering too many Lambda functions concurrently. AWS imposes a soft limit of 1000 concurrent Lambda functions per region
2. The number of requests generated by the load testing framework has exceeded the threshold for the HTTP flood rate-based rule set in the WAF settings for the stage in question
3. The load testing tool has exceeded the soft limit for request rate allowed by API Gateway
4. The Lambda function is sometimes taking 30 seconds or more to finish executing

Answer: d

**EXPLANATION:** The SA-P exam sometimes focuses on knowledge of response codes from API Gateway and what each distinct HTTP response code could mean. The key to answering this question correctly is being able to distinguish between 4XX and 5XX HTTP error response codes. Though AWS has not been entirely consistent in their error code assignment philosophy, 4XX usually happens any time throttling kicks in because the request in that case never makes to an instance of Lambda function. 5XX happens when a Lambda function is actually instantiated, but some error (like time out) happened inside the Lambda function. One sneaky way to remember this is the fact that 5XX errors are called server errors in HTTP-land, so to generate a 5XX a server process must exist (and must have failed). Of course, in this context, the HTTP server process is a Lambda function - so in scenarios where throttling prevented a Lambda function from getting spawned, the response code cannot be 5XX. This is not consistently followed by AWS API Gateway error design, though, as we can see that AUTHORIZER\_CONFIGURATION\_ERROR and AUTHORIZER\_FAILURE are both 500, though no Lambda function is actually spawned in either case. However, the candidate must remember that throttling always results in 4XX codes. An Endpoint Request Timed-out Exception (504) suggests that the requests in question actually made its way past the API Gateway into a Lambda function instance. For the scenario where request rate exceeds API Gateway limits, the request would be blocked by API Gateway itself. The response would be 429. The exact knowledge of the code 429, however, is not needed to eliminate this choice. It is expected of the candidate to know that any kind of throttling always results in 4XX response codes, so this choice must be incorrect. The scenario where 1000 Lambda functions are already running is a similar example of throttling - the 1001st Lambda function will not even be spawned. The response, again, will be 429. However, the exact knowledge of the code 429 is not needed to eliminate this choice. It is expected of the candidate to know that any kind of throttling always results in 4XX response codes, so this choice must be incorrect. The WAF scenario is yet another example of the request not even crossing the protections placed at the gateway level. If WAF is activated on API Gateway, it will block requests when the rate exceeds the HTTP flood rate-based rule (provided all such requests come from a single client IP address). However, the response, again, will be in the 4XX area (specifically, 403 Forbidden) - however, the exact knowledge of the code 403 is not needed to eliminate this choice. It is expected of the candidate to know that any kind of throttling always results in 4XX response codes, so this choice must be incorrect. This leaves Lambda time-out as the only correct answer. The mention of 30 seconds or more is a diversion tactic, in case candidate believes that the relevant Lambda time-out is 5 minutes. A given Lambda function instance may have a time-out limit of 5 minutes, but when it is invoked from API Gateway, the timeout imposed by API Gateway is 29 seconds. If a Lambda function runs for longer than 29 seconds, API Gateway will stop waiting for it and return 504 Endpoint Request Timed-out Exception.

You are considering a migration of your on-prem containerized web application and CouchBase database to AWS. Which migration approach has the lowest risk and lowest ongoing administration requirements after migration?

1. Provision sufficient sized EC2 instances to host the web application and Couchbase. Manually install the web application and Couchbase on the EC2 instances and configure rsync and DMS to synchronize the web server and database respectively. Once the AWS environment is proven, change the DNS entries to point to the new AWS landscape. Mange the instances going forward with AWS Config.
2. Use SCT to read the existing Couchbase schema and recreate it in DynamoDB. Use DMS to initially migrate the data from Couchbase and keep it in sync. Import the web application into ECS using a Fargate cluster. Update the ECS web application to use DynamoDB. Once the AWS landscape is proven, do a final commit from the web application container state to the latest version in the registry. Wait until ECS completes the update of the new container and change DNS entries to point to the new AWS landscape.
3. Import the containers into Elastic Container Registry. Deploy the web application and database on ECS using an EC2 cluster. Once the AWS version is proven, do a final commit of the container state to the latest version in the registry and use Force New Deployment on the ECS console for the service. Change over DNS entries to point to the new AWS landscape.
4. Use Server Migration Service to migrate the on-prem servers into AWS as AMIs. Configure data volume replication to synchronize both the web server and database AMIs. Run in parallel for no longer than 90 days. When the new environment is proven, change over the DNS entry to point to the new AWS landscape.

Answer: c

EXPLANATION:

A lift-and-shift approach when containers are involved is often a very easy and low-risk way to migrate to the cloud. ECS is a good option of you already have a container landscape. Fargate provides more automated scale and management, but AWS wants users to treat Fargate as an ephemeral platform, so an application like CouchBase that requires persistant storage would not work well. Our best option for least management is ECS on an EC2 cluster.

You have just been informed that your company's data center has been struck by a meteor and it is a total loss. Your company's applications were not capable of being deployed with high availability so everything is currently offline. You do have a recent VM images and DB backup stored off-site. Your CTO has made a crisis decision to migrate to AWS as soon as possible since it would take months to rebuild the data center. Which of the following options will get your company's applications up and running again in the fastest way possible?

1. Call your data communications provider and order a Direct Connect link to your main office. Order a Snowball Edge to serve as a mobile data center. Restore the VM image to the Snowball Edge device as an EC2 instance. Restore the backup to an RDS instance on the Edge device. When the Direct Connect link is installed, use that to smoothly migrate to AWS.
2. Explain to company stakeholders that it is not possible to migrate from the backups directly to AWS. Recommend that we first find a co-location site, procure similar hardware as before the disaster and restore everything there. Then, we can carefully migrate to AWS.
3. Copy the VMs into AWS and create new AMIs from them. Create a clustered auto scaling group across multiple AZs for your application servers. Provision a multi-AZ RDS instance to eliminate the single-point-of-failure problem. Restore the data from the backups using the database admin tools.
4. Use Server Migration Service to import the VM into EC2. Use DMS to restore the backup to an RDS instance on AWS.
5. Use VM Import to upload the VM image to S3 and create the AMI of key servers. Manually start them in a single AZ. Stand-up a single AZ RDS instance and use the backup files to restore the database data.

Answer: e

EXPLANATION:

The Server Migration Service uses the Server Migration Service Connector which is an appliance VM that needs to be loaded locally in vCenter. We don't have a VMware system...only a backup of an image so this won't work. The best thing we can do is import the VM and restore the database.

You work for a Clothing Retailer and have just been informed the company is planning a huge promotional sale in the coming weeks. You are very concerned about the performance of your eCommerce site because you have reached capacity in your data center. Just normal day-to-day traffic pushes your web servers to their limit. Even your on-prem load balancer is maxed out, mostly because that's where you terminate SSL and use sticky sessions. You have evaluated various options including buying new hardware but there just isn't enough time. Your company is a current AWS customer with a nice large Direct Connect pipe between your data center and AWS. You already use Route 53 to manage your public domains. You currently use VMware to run your on-prem web servers and sadly, the decision was made long ago to move the eCommerce site over to AWS last. Your eCommerce site can scale easily by just adding VMs, but you just don't have the capacity. Given this scenario, what is the best choice that would leverage as much of your current infrastructure as possible but also allow the landscape to scale in a cost-effective manner?

1. Use VM import to import a VM of a current web server into AWS as an AMI. Create an ALB on AWS. Define a target group using public IP addresses of your on-prem web servers and additional EC2 instances created from the imported AMI. Use Route 53 to update your public facing eCommerce name to point to the ALB as an alias record.
2. Use Server Migration Service to import a VM of a current web server into AWS as an AMI. Create an NLB on AWS. Define a target group using private IP addresses of your on-prem web servers and additional AWS-based EC2 instances created from the imported AMI. Use Route 53 to update your public facing eCommerce name to point to the NLB as an alias record.
3. Use VM import to import a VM of a current web server into AWS as an AMI. Create an ALB on AWS. Define two target groups: one containing the public IP addresses of your on-prem load balancer and one including an auto scaling group of additional EC2 instances created from the imported AMI. Assign both target groups to the ALB using the same listener port. Use Route 53 to update your public facing eCommerce name to point to the ALB as an alias record.
4. Use Server Migration Service to import a VM of a current web server into AWS as an AMI. Create an ALB on AWS. Define a target group using private IP addresses of your on-prem web servers and additional AWS-based EC2 instances created from the imported AMI. Use Route 53 to update your public facing eCommerce name to point to the ALB as an alias record.

Answer: d

EXPLANATION:

A Target Group for an ALB can contain instances or IP addresses. In this case, we can define the private IP addresses of our on-prem web servers along side the private IP addresses of any EC2 instances we spin up. The caveat is that we can only use private IP addresses when defining a target group in this way.

You are consulting with a small Engineering firm that wants to move to a Bring-Your-Own-Device policy where employees are given some money to buy whatever computer they want (within certain standards). Because of device management and security concerns, along with this policy is the need to create a virtualized desktop concept. The only problem is that the specialized engineering applications used by the employees only run on Linux. Considering current platform limitations, what is the best way to deliver a desktop-as-a-service for this client?

1. Launch a Windows Workspace and install VirtualBox along with a minimal Linux image. Within that Linux image, install the required software. Create an image of the Windows Workspace and create a custom bundle from that image. Use that bundle when launching subsequent Workspaces.
2. Package the required apps as WAM packages. When launching new Windows Workspaces, instruct users to allow WAM to auto-install the suite of applications prior to using the Workspace.
3. Launch a Linux Workspace in AWS WorkSpaces and customized it with the required software. Then, create a custom bundle from that image and use that bundle when you launch subsequent Workspaces.
4. Launch an EC2 Linux instance and install XWindows and Gnome as the GUI. Configure VNC to allow remote login via GUI and load the required software. Create an AMI and use that to launch subsequent desktops.
5. Given current limitations, running Linux GUI applications remotely on AWS is not feasible. They should reconsider their BYOD policy decision.

Answer: c

EXPLANATION:

AWS Workspaces added support for Linux desktops the middle of 2018. BYOD scenarios work together well with a DaaS concept to provide security, manageability and cost-effectiveness.

A hotel chain has decided to migrate their business analytics functions to AWS to achieve higher agility when future analytics needs change, and to lower their costs. The primary data sources for their current on-premises solution are CSV downloads from Adobe Analytics and transactional records from an Oracle database. They've entered into a multi-year agreement with Tableau to be their visualization platform. For the time being, they will not be migrating their transactional systems to AWS. Which architecture will provide them with the most flexible analytics capability at the lowest cost?

1. Use Oracle Data Guard to continuously replicate Oracle transactional data to an Oracle instance on Amazon EC2. Configure AWS Glue to aggregate the transactional data from the Oracle instance for each dimension into Amazon Redshift. Use AWS Glue to write the Adobe Analytics data to Redshift. Use Amazon QuickSight to query the data for visualization.
2. Implement AWS Database Migration Service to continuously replicate Oracle transactional data to an Amazon RDS Oracle instance. Use AWS Glue to write the Adobe Analytics data to the RDS Oracle instance. Install Tableau on Amazon EC2 and write queries against the RDS Oracle database.
3. Employ AWS Database Migration Service to continuously replicate Oracle transactional data to Amazon S3. Configure AWS Glue to aggregate the transactional data from S3 for each dimension into Amazon Redshift. Use AWS Glue to write the Adobe Analytics data to Amazon S3 in Parquet format. Install Tableau on Amazon EC2 and write queries to Amazon Redshift Spectrum.
4. Configure AWS Database Migration Service to continuously replicate Oracle transactional data to Amazon Redshift. Use AWS Glue to write the Adobe Analytics data to Redshift. Use Amazon QuickSight to query the data for visualization.

Answer: c

EXPLANATION:

AWS Database Migration Service can be configured with an on-premises Oracle database as a source and S3 as a target. It can provide continuous replication between the two. AWS Glue can aggregate the data from S3 according to desired reporting dimensions and store the summaries in Redshift. Keeping the transactional detail in S3 and only keeping the aggregate information in Redshift will save on costs. The same is true for keeping transactional detail in S3 instead of RDS Oracle. AWS Glue is a great solution for transforming the Adobe Analytics CSV files to Parquet format in S3. Parquet's columnar organization will provide excellent performance for Redshift Spectrum queries that join between Redshift tables and S3. Tableau's Redshift connector supports Redshift Spectrum queries. For this use case, using Amazon QuickSight would not make sense since the company has already committed payments to Tableau via their multi-year agreement.

You work for a Genomics company which has decided to migrate its DNA Sequencing application to the AWS Cloud. The application is containerized. Currently, container image A works on genomics data residing on an on-premises file server, validating the data and updating the metadata in a local database. When it is done, engineers manually trigger 100 or more instances of container image B that process this data in parallel by reading the metadata, creating output files. When all these container instances have done their job, engineers manually trigger container image C that validates the results, cleans up and sends notifications. The CTO has decided to use S3 for storing the input and output data files. She has also mandated that the parallel processing phase should run on a fleet of Spot EC2 instances to reduce compute costs. She also wants to automate the workflow, so that engineers do not have to manually trigger the next set of actions. The requirement is to minimize administrative overhead and custom development for the migration. As the AWS Architect, which of the following approaches should you recommend?

1. Use AWS SWF workers and deciders to manage the workflow. Configure the workers to use EC2 Spot Instances
2. Use AWS Batch, setting up an array job with 100 or more copies preceded by pre-requisite and follow-up jobs where the workflow is controlled by dependencies between jobs. Also, use Spot as the Provisioning Model for compute environment
3. Use AWS ECS with Fargate Launch Type to run the container images, configuring the cluster to use Spot Instances and setting up the workflow in the service definition JSON file so that it runs Task C only after Task B is completed and it runs Task B only after Task A is completed
4. Use AWS ECS with EC2 Launch Type to run the container images, configuring the cluster to use Spot Instances and setting up the workflow in the service definition JSON file so that it runs Task C only after Task B is completed and it runs Task B only after Task A is completed

Answer: b

EXPLANATION:

AWS ECS does not natively provide workflow management. In an ECS service definition file, you cannot specify a sequence of tasks with execution dependencies such that one will be run only after the previous one completes. Hence, the two ECS choices are ruled out. Distraction warning - Fargate does not allow you to specify Spot instances as it is serverless in nature (it absolves you from specifying server details). This effectively creates a distraction - when the candidate rules out ECS Fargate due to this reason, they may be relieved to see the ECS EC2 choice and jump to a conclusion because it is relatively easy to remember that EC2 launch type actually lets you select Spot instances. However, this distraction is designed to take focus away from the fact that neither of these two choices is correct. Both of the choices require service definition files to set up execution workflows. Task instances mentioned in an ECS service definition file are executed in parallel - ECS does not control the sequence of tasks. AWS SWF does not let you specify Spot instances either. Also, SWF is usually used in cases where human intervention is needed in the workflow. This leaves AWS Batch as the correct answer. AWS Batch is indeed the most suitable AWS service for this scenario as it meets all requirements.

A client wants help setting up a way to manage access to the AWS Console and various services on AWS for their employees. They are starting out small but expect to provide AWS-hosted services to their 20,000 employees within the year. They currently have Active Directory on-premises, use VMware to host their VMs. They want something that will allow for minimal administrative overhead and something that could scale out to work for their 20,000 employees when they have more services on AWS. Due to audit requirements, they need to ensure that the solution can centrally log sign-in activity. Which option is best for them?

1. Download and install the AWS ActiveDirectory Sync appliance and install it in vCenter. Configure the Sync appliance to connect to the local AD and replicate to an instance of Simple AD on AWS. In IAM, create corresponding roles and policies for the permissions you want to allow on AWS. Assign these roles to the synchronized Simple AD users in IAM.
2. Create a OAuth Identity Provider in IAM and create roles and policies with the appropriate level of permissions. In AD, create groups which correspond to the roles you have created in IAM and populate the AD groups with the desired users. Download and install the OAuth Identity Connector for AD. Configure the connector for the OAuth Identity Provider on AWS.
3. Configure Cognito with web federation against the on-prem Active Directory. In IAM, create corresponding users corresponding to the Cognito accounts you want to allow on AWS. Assign these roles to the user pools within Cognito. Distribute the Cognito SSO client to your users.
4. Connect the multiple accounts together using AWS Organizations. Deploy AD Connector on AWS and configure their on-prem AD. Create corresponding roles and groups in IAM and map those to their local AD groups. Use STS to allow users to authenticate into AWS.
5. Connect the multiple accounts with AWS Organizations. Deploy AWS Directory Service for Microsoft Active Directory on AWS and configure a trust with your on-premises AD. Configure AWS Single Sign-On with the users and groups who are permitted to log into AWS. Give the users the URL to the AWS SSO sign-in web page.

Answer: e

EXPLANATION:

For userbases more than 5,000 and if they want to establish a trust relationship with on-prem directories, AWS recommends using AWS Directory Service for Microsoft Active Directory. This is also compatible with AWS Single Sign-On which provides a simple way to provide SSO for your users across AWS Organizations. Additionally, you can monitor and audit sign-in activity centrally using CloudTrail.

A food service business has begun an initiative to migrate all applications and data to the AWS cloud. Governance needs to be established before any migrations can occur. Business units such as sales, marketing, and product management have fluctuating infrastructure capacity and security requirements, while other business units like finance, operations, and human resources have more static demand. Security policies and compliance needs vary by project group within each business units. Each business unit is responsible for it's own cost center, and the finance group would like cost reporting to be as streamlined as possible. Which AWS account structure will best satisfy the company's governance needs?

1. Use AWS Organizations with a single Organizational Unit to consolidate costs. Create a billing account, a shared services account, and a log archive account in the Organizational Unit. Create individual accounts for each business unit. Manage security requirements for each project group with VPC networking services such as Security Groups and Network ACLs
2. Use AWS Organizations to create Organizational Units for each business unit. Create a billing account, a shared services account, and a log archive account in each Organizational Unit. Create accounts for each project group within the business unit. Establish standard tags to sort the AWS Detailed Billing report by cost center
3. Use AWS Organizations to create a core Organizational Unit that contains a billing account, a shared services account, and a log archive account. Place business units with similar security requirements in shared Organizational Units. Create accounts for each business unit in the shared Organizational Units. Manage security requirements for each project group with VPC networking services such as Security Groups and Network ACLs. Establish standard tags to sort the AWS Detailed Billing report by cost center
4. Use AWS Organizations to create a core Organizational Unit that contains a billing account, a shared services account, and a log archive account. Create an Organizational Unit for each business unit that contains accounts for each project group within the business unit. Establish standard tags to sort the AWS Detailed Billing report by cost center

Answer: d

EXPLANATION:

Leveraging AWS Organizations to manage an account structure with a core Organizational Unit and Organizational Units for each business unit provides flexibility for future organizational changes. Creating an account for each project group facilitates security policy differences within business units, and limits the exposure of a single security event. Managing differing security requirements by project group in a single account will require more governance maintenance. Creating billing, shared services, and log archive accounts in multiple Organizational Units will result in duplication of services, and can be done at the core level.

You are an AWS architect working for a B2B Merger and Acquisitions consulting firm, which has 15 business units spread across several US cities. Each business unit has its own AWS account. For administrative ease and standardization of AWS Usage patterns, corporate headquarters have decided to use AWS Organizations to manage the individual accounts by grouping them into relevant Organization Units (OU-s). You have assisted the Organization Administrator to write and attach Service Control Policies (SCP-s) to the OU-s. SCP-s have been configured as the default Deny list, and they are written to explicitly deny actions wherever required. Data Scientists in one of the Business Units are complaining that they are unable to spin up or access Sagemaker Clusters for building, training and deploying Machine Learning models. Which of the following can be a possible cause and how can this be fixed?

1. The Service Linked Role associated with AWS Sagemaker does not allow the data scientists to assume the Role. To fix this, add a Trust Policy to the Sagemaker Service Linked Role that lists the IAM user ids of the data scientists as Principal, with the value of Action is AssumeRole and Effect is set to Allow
2. The SCP for the OU to which the Business Unit Account belongs does not explicitly allow granting Sagemaker access. To fix this, add the following to the attached policy Statement of the SCP - Effect set to Allow, Action set to Everything starting with SageMaker, Resource set to All
3. The IAM Policy attached to the IAM Role that the data scientists are assuming in the Business Unit Account does not grant them Sagemaker access. To fix this, add the following to the IAM Policy Statement for that Role - Effect set to Allow, Action set to Everything starting with SageMaker, Resource set to All
4. SCP is configured as a Deny List. To fix this, SCP must be configured as an Allow List instead of a Deny List for the OU. Then, Sagemaker access should be added explicitly

Answer: c

**EXPLANATION:**

This question tests the conceptual knowledge of Service Control Policies (SCP-s) in AWS Organizations. The choice that requires the SCP to be modified is incorrect because there is no need to grant explicit allows from SCP, especially when it is configured in the default mode (Deny List mode). In this mode, everything is allowed by default. We only need to specify what we want to deny. The choice that requires the IAM Policy to be modified is correct because SCPs do not actually grant any permission. The permission that is missing in this case must be granted via IAM Roles and Policies at the Account level. The choice mentioning Service Linked Roles is incorrect as Trust Policies on Service Linked Roles cannot be modified to let an IAM user assume that role. Service Linked Roles are for AWS Services. The choice that requires re-configuration of SCP as Allow List is incorrect because configuring SCP as Allow List is usually a messy idea. In that case, all permissions will need to be explicitly granted, and it can easily defeat the purpose of streamlining management and reducing administrative overhead by using AWS Organizations. Allow Lists have very specific use cases. In addition, no change in the SCP grants or allows any permission. Permission needs to be granted using IAM Roles and Policies at the Account level.

You are consulting for a large multi-national company that is designing their AWS account structure. The company policy says that they must maintain a centralized logging repository but localized security management. For economic efficiency, they also require all sub-account charges to roll up under one invoice. Which of the following solutions most efficiently addresses these requirements?

1. Create a stand-alone consolidated logging account and configure all sub-account CloudWatch and CloudTrail activity to route to that account. Use ACLs to restrict sub-accounts from changing CloudWatch and CloudTrail configuration. Configure consolidated billing under a single account and register all sub-accounts to that billing account. Create localized IAM Admin accounts for each sub-account. Establish trust relationships between the Consolidated Billing account and all sub-accounts.
2. Create a stand-alone consolidated logging account and configure all sub-account CloudWatch and CloudTrail activity to route to that account. Use an SCP to restrict sub-accounts from changing CloudWatch and CloudTrail configuration. Configure consolidated billing under a single account and register all sub-accounts to that billing account. Create localized IAM Admin accounts for each sub-account.
3. Configure billing for each account to load into a consolidated RedShift instance. Create a centralized security account and establish trust relationships between each sub-account. Configure admin roles within IAM of each sub-account for local administrators. Create a stand-alone consolidated logging account and configure all sub-account CloudWatch and CloudTrail activity to route to that account.
4. Create a stand-alone consolidated logging account and configure all sub-account CloudWatch and CloudTrail activity to route to that account. Create localized IAM policies to restrict modification of CloudWatch and CloudTrail configuration. Configure consolidated billing under a single account and register all sub-accounts to that billing account. Create a centralized security account and establish trust relationships between each sub-account.

Answer: b

EXPLANATION:

Service Control Policies are an effective way to broadly restrict access to certain features of sub-accounts. Use of a single separate logging account is an effective way to create a secure logging repository.

You have been entrusted to act as the interim AWS Administrator following the departure of the erstwhile Administrator in your company. You notice that there are several existing roles called role-engineer, role-manager, role-qa, role-dba, role-data-scientist, etc. When a new person joins the company, the new IAM user simply assumes the right role while using AWS - this allows central management of permissions and eliminates the need to manage permissions on a per-user basis. A new QA hire joins the company a few days later. You create an IAM User for her. You attach a Policy to the new IAM User that allows Action STS AssumeRole on any Resource. However, when this employee logs in the same day and tries to switch roles to role-qa, she is denied and is unable to assume the role-qa Role. What could be one reason why this is happening and how can it be best fixed?

1. You have not modified the Trust Policy of the IAM Role role-qa to allow the new IAM User to assume the Role. To fix this, add the arn of the new IAM User to the Condition element of the Trust Policy of the Role
2. You have not modified the Trust Policy of the IAM Role role-qa to allow the new IAM User to assume the Role. To fix this, add the arn of the new IAM User to the Principal element of the Trust Policy of the Role
3. Sufficient time has not passed since you made the changes. It takes up to 12 hours to propagate IAM role changes. To fix this, ask her to try again the next day.
4. You have not modified the Trust Policy of the IAM User to trust the Role role-qa. To fix this, add a Condition to the IAM Policy attached to the new user that filters on the role and specify the arn of role-qa

Answer: b

EXPLANATION:

In order to allow an IAM User to successfully assume an IAM Role, two things must happen. First, the Policy attached to the User must allow the action STS AssumeRole. This is already true according to the question. Second, the Trust Policy of the Role itself must allow the User in question to assume the Role. This second condition can be met if we specify the arn of the User in the Principal element of the Trust Policy. In general, this question can be answered if the candidate is familiar with the concept of Principal in a Role, see link - A Principal within an Amazon IAM Role specifies the user (IAM user, federated user, or assumed-role user), AWS account, AWS service, or other principal entity that is allowed or denied to assume or impersonate that Role. Trust Policy is different than the Policy permissions - think of Policy Permissions as [what can be accessed] and Trust Policy as [who can access]. Trust Policy cannot belong to an IAM User, hence the choice that claims the problem to be an unmodified User Trust Policy is incorrect. IAM changes are instantly effective, so the choice that points at the need of a time delay is also incorrect. Among the other two choices, the knowledge needed to pick the right one is an awareness of the Principal element.

You are helping a client design their AWS network for the first time. They have a fleet of servers that run a very precise and proprietary data analysis program. It is highly dependent on keeping the system time across the servers in sync. As a result, the company has invested in a high-precision stratum-0 atomic clock and network appliance which all servers sync to using NTP. They would like any new AWS-based EC2 instances to also be in sync as close as possible to the on-prem atomic clock as well. What is the most cost-effective, lowest maintenance way to design for this requirement?

1. Create a dedicated host instance on AWS and place it within a transit VPC. Configure the server to run NTP as a stratum-2 server. Ensure NTP (UDP port 123) is allowed inbound and outbound in the Security Groups local to the stratum-2 server.
2. Create a bridged network tunnel from the on-prem time server to the VPCs on AWS. Configure the VPC route tables to route NTP (UDP 123) over the tunnel.
3. Configure your Golden AMI to use Amazon Time Sync Server at 169.254.169.123 and require this AMI to be used. Use AWS Config to periodically audit the NTP configuration of all AWS assets.
4. Configure a DHCP Option Set with the on-prem NTP server address and assign it to each VPC. Ensure NTP (UDP port 123) is allowed between AWS and your on-prem network.
5. Deploy a third-party time server from the AWS Marketplace. Configure it to sync from the on-prem time server. Ensure NTP (UDP port 123) is allow inbound in the NACLs for the VPC containing the third-party server.

Answer: d

EXPLANATION:

DHCP Option Sets provide a way to customize certain parameters that are issued to clients upon a DHCP request. Setting the NTP server is one of those parameters.

Your business depends on AWS S3 to host three kinds of files - images, documents and compressed installation packages. These files are accessed and downloaded by end-users from all US regions and west EU, though the compressed installation packages are downloaded rarely as users tend to access the service from their browsers instead of installing anything on their machines. Each installation package bundles several images and documents, and also includes binaries that are downloaded from a 3rd party service while creating the package files. The images and documents range from a few KBs to a few hundred KBs in size and they are mostly static in nature. However, the compressed installation package files are generated every few hours because of changes done by the 3rd party service to their binaries, and some of them are as large as a few hundred GB-s. The installation package files can be regenerated from the images and documents fairly quickly if required. It is important to be able to retrieve older versions of the images and documents. Which of the following storage solutions is the most cost-effective approach to design the storage for these files?

1. Store all three kinds of files in a single S3 bucket. Turn on versioning for the image and document objects only, but not for the compressed installation package files. Set Storage Class to Standard S3 while uploading images and documents. Set Storage Class to Infrequent Access while uploading compressed installation package files
2. Store all three kinds of files in a single S3 bucket. Turn on versioning for the bucket. Set Storage Class to Standard S3 while uploading images and documents. Set Storage Class to One-Zone Infrequent Access while uploading compressed installation package files
3. Store the images and documents in one bucket (A) and the compressed installation package files in another bucket (B). Turn on versioning for Bucket A only. Set Storage Class to Standard S3 while uploading objects to Bucket A. Set Storage Class to One-Zone Infrequent Access while uploading objects to Bucket B
4. Store the images and documents in one bucket (A) and the compressed installation package files in another bucket (B). Turn on versioning for both the buckets. Set Storage Class to Standard S3 while uploading objects to Bucket A. Set Storage Class to One-Zone Infrequent Access while uploading objects to Bucket B

Answer: c

**EXPLANATION:** The areas tested by this question are:\n1. Versioning cannot be enabled at the object level. It is a bucket-level feature. This rules out the choice where we have a single bucket and selectively turn on versioning on for some objects only.\n2. If you enable Versioning for a bucket containing large objects that are frequently created/uploaded, it will result in higher storage cost as all the previous versions will result in storage volume growing quickly because of frequent writes. In the given scenario, the compressed installation package files are large and also frequently generated (every few hours). There is no requirement to version them, as they can be quickly generated on-demand. Hence, putting them in a bucket that has Versioning enabled is not a good cost-effective solution. This rules out two choices - one where we have a single versioned bucket, the other where we enable versioning for both buckets.\n3. Note that all options except one correctly identify the storage class requirements - the compressed installation package files should be stored as One-Zone IA because durability is not a prime requirement for these files (simply because they can be regenerated on-demand easily). They are rarely downloaded, hence IA is the correct class. Combined with low durability, One Zone IA is the most cost-effective solution. Only one option uses the incorrect storage tier for these files - note that IA is more expensive than One-Zone IA, and the question is about cost-effectiveness. Hence, the only correct answer is the one that addresses both Versioning and Storage Class requirements correctly.

You are the solution architect for a research paper monetization company that makes large PDF Research papers available for download from an S3 bucket. The S3 bucket is configured as a static website. A Route53 CNAME record points the custom website domain to the website endpoint of the S3-hosted static website. As demand for downloads has increased throughout the world, the architecture board has decided to use a Cloudfront web distribution that fetches content from the website endpoint of the static website hosted on S3. The Route 53 CNAME record will be modified to point at the Cloudfront distribution URL.\n\nFor security, it is required that all request from client browsers use HTTPS. Additionally, the system must block anyone from accessing the S3-hosted static website directly other than the Cloudfront distribution. Which approach meets the above requirements?

1. While setting up the Cloudfront Web Distribution, use the website endpoint of the S3-hosted static website as the Origin Domain Name. Also, set up Origin Custom Header. Then specify a header like Referer, with its value set to some secret value. Set the bucket policy of the S3 bucket to allow s3 GetObject on the condition that the HTTP request includes the custom Referer header. In the Cloudfront web distribution, set the value of the property Viewer Protocol Policy to HTTPS Only, or Redirect HTTP to HTTPS.
2. While setting up the Cloudfront Web Distribution, select the S3 bucket as the origin. Select Restrict Bucket Access to Yes, and create a new Origin Access Identity (OAI) that will prevent anyone else other than the Cloudfront web distribution to access the S3 bucket. In the Cloudfront web distribution, set the value of the property Viewer Protocol Policy to HTTPS Only, or Redirect HTTP to HTTPS.
3. While setting up the Cloudfront Web Distribution, use the website endpoint of the S3-hosted static website as the Origin Domain Name. Also, set up Origin Custom Header. Then specify a header like Referer, with its value set to some secret value. Set the bucket policy of the S3 bucket to allow s3 GetObject on the condition that the HTTP request includes the custom Referer header. In the Cloudfront web distribution, set the value of the property Viewer Protocol Policy to HTTPS Only, or Redirect HTTP to HTTPS. Additionally, set the value of Origin Protocol Policy to HTTPS Only.
4. While setting up the Cloudfront Web Distribution, select the S3 bucket as the origin. Select Restrict Bucket Access to Yes, and create a new Origin Access Identity (OAI) that will prevent anyone else other than the Cloudfront web distribution to access the S3 bucket. In the Cloudfront web distribution, set the value of the property Viewer Protocol Policy to HTTPS Only, or Redirect HTTP to HTTPS. Additionally, set the value of Origin Protocol Policy to HTTPS Only.

Answer: a

EXPLANATION:

The key to answering this question correctly is to note the fact that the origin is a website and not just a plain S3 bucket - note the usage of the phrase website endpoint in the question. While setting up such an origin, one cannot just pick the S3 bucket as the origin, or use OAI. Hence, the two choices that rely on picking the S3 bucket as the origin and using OAI to restrict access are incorrect.\n\nIn the given scenario, the Cloudfront web distribution is being configured to use the website endpoint of the static website as the origin. A big difference between these two scenarios is - if you use an S3 bucket as the origin, Cloudfront uses the REST API interface of S3 to communicate with the origin. If you use the website endpoint as the origin, Cloudfront uses the website URL as the origin. These endpoints have different behaviours - see the link titled Key Differences Between the Amazon Website and the REST API Endpoint. S3 REST API is more versatile, allowing the client to pass richer information like AWS Identity, thereby allowing the exchange of information that makes OAI possible. That is the reason why OAI cannot be used when Cloudfront is using the website endpoint where only GET and HEAD requests are allowed on objects.\n\nTherefore, in this scenario, OAI cannot be used. Instead, we have to use a custom header that only Cloudfront can inject into the Origin-bound HTTP request. The bucket policy of the S3 bucket hosting the static website can then check for the existence of said header. The assumption here is that if any browser ever directly uses the website URL of the S3-hosted static website (which is of the format examplestaticwebsitebucket.s3-website-us-east-1.amazonaws.com), their request will not contain this header, and hence will be rejected by the bucket policy.\n\nAlso, S3-hosted static websites do not support HTTPS. Therefore, Origin Protocol Policy, in this case, cannot be set to HTTPS Only. We can only set Viewer Protocol Policy. Only the browser to Cloudfront half will be HTTPS. The Cloudfront to Origin half cannot be HTTPS in this case

You work for a retail services company that has 8 S3 buckets in us-east-1 region. Some of the buckets have a lot of objects in them. There are Lambda functions and EC2-hosted custom application code where the names of these buckets are hardcoded. Your manager is worried about disaster recovery. As part of her business continuity plan, she has requested you to set up Cross-Region Replication of these S3 buckets to us-west-1, ensuring that the replicated objects are using a less expensive Storage Class because they would not be accessed unless disaster strikes. You are worried that in the event of failover due to the entire us-east-1 region being unavailable, the application code, once deployed in us-west-1, must continue to work while trying to access the S3 buckets in the new region. She has also requested you to start taking periodic snapshots of EBS Volumes and make these snapshots available in the us-west-1 region so that EC2 instances can be launched in us-west-1 using these snapshots if needed. How would you ensure that (a) the launching of EC2 instances works in us-west-1 and (b) your application code works with the us-west-1 S3 buckets?

1. To ensure that EC2 instances can be launched in us-west-1 when needed, schedule periodic creation of EBS snapshots of both root and non-root volumes using Data Lifecycle Manager. Then, set up a Lambda function to copy these snapshots to the us-west-1 region using the copy-snapshot API. Use the root volume snapshots to create an AMI in us-west-1 region when needed. Launch EC2 instances from this AMI.\n\nTo ensure application compatibility with S3 buckets in us-west-1, create corresponding S3 buckets with different names in us-west-1. Change the application code to not hardcode the names of S3 buckets. Instead, read the S3 bucket names from AWS Systems Manager Parameter Store. Set up a Parameter Store in us-west-1 with the same keys but containing the us-west-1 bucket names. Set up Cross-Region Replication and specify that the object be moved to Infrequent Access Storage Class in the destination bucket. Run a script to copy pre-existing objects over as they are not copied automatically while setting up Cross-Region Replication
2. To ensure that EC2 instances can be launched in us-west-1 when needed, schedule periodic creation of EBS snapshots of both root and non-root volumes using Data Lifecycle Manager such that the snapshots are created directly in us-west-1 region. Use the root volume snapshots to create an AMI in us-west-1 region when needed. Launch EC2 instances from this AMI.\n\nTo ensure application compatibility with S3 buckets in us-west-1, create the S3 buckets in us-west-1 with the same names as the corresponding ones in us-east-1, so that application code does not break. Set up Cross-Region Replication and specify that the object be moved to Infrequent Access Storage Class in the destination bucket. Run a script to copy pre-existing objects over as they are not copied automatically while setting up Cross-Region Replication
3. To ensure that EC2 instances can be launched in us-west-1 when needed, schedule periodic creation of EBS snapshots of both root and non-root volumes using Data Lifecycle Manager. Then, set up a Lambda function to copy these snapshots to the us-west-1 region using the copy-snapshot API. Use the non-root volume snapshots to create an AMI in us-west-1 region when needed. Launch EC2 instances from this AMI.\n\nTo ensure application compatibility with S3 buckets in us-west-1, create the corresponding S3 buckets with different names in us-west-1. Change the application code to not hardcode the names of S3 buckets. Instead, read the S3 bucket names from AWS Systems Manager Parameter Store. Set up a Parameter Store in us-west-1 with the same keys but containing the us-west-1 bucket names. Set up Cross-Region Replication and specify that the object be moved to Infrequent Access Storage Class in the destination bucket. Pre-existing objects are copied over automatically while setting up Cross-Region Replication
4. To ensure that EC2 instances can be launched in us-west-1 when needed, schedule periodic creation of EBS snapshots of both root and non-root volumes using Data Lifecycle Manager such that the snapshots are created directly in us-west-1 region. Use the non-root volume snapshots to create an AMI in us-west-1 region when needed. Launch EC2 instances from this AMI.\n\nTo ensure application compatibility with S3 buckets in us-west-1, create the S3 buckets in us-west-1 with the same names as the corresponding ones in us-east-1, so that application code does not break. Set up Cross-Region Replication and specify that the object be moved to Infrequent Access Storage Class in the destination bucket. Pre-existing objects are copied over automatically while setting up Cross-Region Replication

Answer: a

EXPLANATION:

This question presents two problems - (1) how to ensure that EBS snapshots are created periodically and are also made available in a different region for launching required EC2 instances in case of failure of the primary region (2) how to deal with application code where S3 bucket names are hardcoded and whether this hardcoding will impact disaster recovery while trying to run in a different region. Both of these problems are real-life issues AWS customers face when designing and planning their disaster recovery solutions.\n\n(1)Remember that Data Lifecycle Manager can only schedule snapshot creation in the same Region. If we want to copy that snapshot into a different region, we must write our own scripts or Lambda functions for doing that. Hence, the choices that state that DLM can be used to directly create the snapshot into different regions are eliminated. Additionally, only root volume snapshots can be used to create an AMI. Non-root EBS Volume snapshots cannot be used to generate an AMI. Hence, the choices that specify using non-root volume snapshots are eliminated.\n\n(2)Remember that S3 bucket names are globally unique. Hence, one cannot create a second S3 bucket in the DR Region with the same name as the bucket in the primary region. Hence, the options that hint the creation of S3 buckets by the same name are eliminated. This results in a problem if S3 names are hardcoded in the application - that application will simply not run in a new region, it will fail. Hence, it is best to avoid hardcoding, and fetch the S3 bucket name from a key-value storage service like AWS Systems Manager Parameter Store at runtime. Creating this Parameter Store in each region and storing the correct bucket names in them can help in designing this non-hardcoded solution. Additionally, enabling Cross-Region Replication does not copy pre-existing content. Hence, the choices that suggest that pre-existing content will be automatically copied are eliminated.

You are the Enterprise Architect in a Risk Quantification firm. The firm has a website which end-users can use to apply for loans and also track the status of their loan application if they log in. When a loan application comes in, several downstream systems need to independently process the application. Right now, the website server-side code invokes these systems one after the other, synchronously, in a tight loop. If one of these downstream systems times out or throws an exception, the entire loan application processing errors out. Even if none of these downstream systems fail, the time it takes to process a loan application is very high due to the serial nature of these systems being invoked. Your CTO wants only the loan-processing application moved to the AWS cloud and re-architected at the same time. The downstream systems are all hosted on-premises and will continue to remain on-premises. They expose REST endpoints that accept POST HTTPS requests, use self-signed certificates and respond synchronously only when they are done processing an application. After re-architecture, all downstream systems must independently start processing an incoming loan application simultaneously. Your CTO wants to know how the loan-processing website application can be architected in the AWS Cloud, and what supporting changes will be needed in the downstream systems on-premises. He wants to minimize code changes to the downstream on-premises systems. Choose the best option

1. For the website that accepts loan applications, run it on EC2 instances in an Auto Scaling Group spanning multiple Availability Zones with an Elastic Load Balancer. It should publish each incoming loan application to an SNS Topic. Configure the SNS topic to have multiple HTTPS subscribers - add each of the downstream system REST API endpoints as a subscriber. Make the following changes in the downstream systems - (a) Make them asynchronous - they should respond with HTTP 201 Accepted immediately without waiting so that SNS does not retry, and then later post the results to a new API Gateway API that will invoke a second Lambda function to update an RDS database which the loan application website can later read to provide status as needed (b) Parse SNS-specific HTTP headers and JSON body format to extract the payload correctly (c) Make them idempotent for the same loan application as SNS may retry in case of lost messages or timeouts (d) Procure server certificates from a trusted Certificate Authority (CA) instead of using self-signed certificate as SNS will not be able to POST to a server with self-signed certificate
2. For the website that accepts loan applications, run it on EC2 instances in an Auto Scaling Group spanning multiple Availability Zones with an Elastic Load Balancer. It should publish each incoming loan application to an SQS Standard Queue. Configure a Lambda listener for the queue. The Lambda function will invoke the REST APIs for all downstream systems in a loop. Make the following changes in the downstream systems - (a) Make them asynchronous - they should respond with HTTP 201 Accepted immediately without waiting, and then later post the results to a new API Gateway API that will invoke a second Lambda function to update an RDS database which the loan application website can later read to provide status as needed (b) Make them idempotent in case Lambda times out or errors and a given loan application re-appears in the queue only to be picked up by another Lambda instance and re-sent to the downstream systems and (c) Procure server certificates from a trusted Certificate Authority (CA) instead of using self-signed certificate as your Lambda function will not be able to POST to a server with self-signed certificate
3. For the website that accepts loan applications, run it on EC2 instances in an Auto Scaling Group spanning multiple Availability Zones with an Elastic Load Balancer. It should publish each incoming loan application to an SNS Topic. Configure the SNS topic to have multiple HTTPS subscribers - add each of the downstream system REST API endpoints as a subscriber. Override the default delivery policy on the subscriber endpoint to remove retries so that downstream systems do not have to worry about synchronous responses taking time or idempotency of retries. Make the following changes in the downstream systems - (a) Parse SNS-specific HTTP headers and JSON body format to extract the payload correctly (b) Procure server certificates from a trusted Certificate Authority (CA) instead of using the self-signed certificate as SNS will not be able to POST to a server with a self-signed certificate
4. For the website that accepts loan applications, run it on EC2 instances in an Auto Scaling Group spanning multiple Availability Zones with an Elastic Load Balancer. It should publish each incoming loan application to an SQS Standard Queue. Configure a Lambda listener for the queue. The Lambda function will invoke the REST APIs for all downstream systems in a loop. Make the following changes in the downstream systems - (a) Make them asynchronous - they should respond with HTTP 201 Accepted immediately without waiting, and then later post the results to a new API Gateway API that will invoke a second Lambda function to update an RDS database which the loan application website can later read to provide status as needed and (b) Make them idempotent in case Lambda times out or errors and a given loan application re-appears in the queue only to be picked up by another Lambda instance and re-sent to the downstream systems

Answer: d

**EXPLANATION:** This is an example of a verbose question with verbose answer choices. You can expect a few such questions in the exam, testing your time management skills. Try to vertically scan the answers to see which parts differ between them. Sometimes, though the answers seem big, a large part of each is identical. You can ignore those parts, as there is nothing to choose between the. Among the four choices, two use SQS and two use SNS to feed the incoming loan applications to the downstream systems. You cannot automatically eliminate either SQS or SNS, as a working solution can be designed with either. Let us see how we can achieve this using SNS first. The basic requirement here is fan-out - a single loan application must be processed by several downstream systems, so there are multiple consumers. Hence, SNS is a natural fit. SNS supports multiple subscribers for a topic. SNS also supports HTTP/HTTPS subscribers. SNS makes POST REST API call to as many HTTP/HTTPS subscribers exist on the topic, so it fits the bill. However, there is a small problem - the requirement states that the downstream systems must be changed as little as possible. If we follow this design, we must change the HTTP Listening part of the downstream systems significantly. Because SNS is directly calling them now, SNS will use its own headers and body format. In fact, SNS POST-s two kinds of messages - one is Subscription Confirmation and one is Notification. A special HTTP header (x-amz-sns-message-type) has the right type in its value. The server side now must parse this header out and look for only the Notification type of message. The body itself will then be JSON formatted with the payload. While the server is probably used to process just the core payload (loan application data) as the HTTP body, the same will now be hidden inside a JSON field called Message inside the request body. Additionally, the downstream systems will have to deal with SNS retries, thus the loan application part must be made idempotent (if the same loan application lands twice, it will ignore the duplicates). Thus, though it is technically possible to design the solution using SNS, it will result in a lot of changes in the downstream systems. Hence, though the SNS option will work, it is not the correct answer because of this reason. Now, let us see how we can design this using SQS. While SQS does not support fan-out (multiple consumers for the same message), the proposed solution uses a Lambda function to achieve fan-out. The Lambda function will pick up the message, and then call the downstream systems one by one. The key to making this work is, of course, to modify the downstream systems from synchronous monolithic beasts to asynchronous servers so that they can instantly respond to the Lambda function and then continue to process the application. We will then have to provide a callback for when it is done. The solution uses an API Gateway for that purpose. Overall, the solution is elegant, and changes to the downstream systems are less than what SNS requires. Hence, SQS is the correct answer. Note that one version of the SNS design proposes to retain the synchronous nature of the downstream systems. That will not work as SNS will not wait more than 15 seconds for a response. The response will then be lost and the main website app will never know the results from the downstream systems. Also, note that though SNS requires the HTTPS subscriber to present a trusted CA-signed certificate, there is no such requirement for Lambda because Lambda is basically your code, you can decide to trust anyone.

You are consulting for a company that performs specialized customer data analytics. Their customers can upload raw customer data to a website and receive back demographic statistics. Their application consists of a REST API created using PHP and Apache. The application is self-contained and works in real-time to return results as a JSON response to the REST API call. Because there is customer data involved, company policy states that data must be encrypted in transit and at rest. Sometimes, there are data quality issues and the PHP application will throw an error. The company wants to be notified immediately when this occurs so they can proactively reach out to the customer. Additionally, many of the company's customers use very old mainframe systems that can only access internet resources using IP address rather than a FQDN. Which architecture will meet these requirements fully?

1. Provision a Network Load Balancer in front of your EC2 target group and terminate SSL at the load balancer using Certificate Manager. Install CloudWatch Logging agent on the EC2 instances and stream logs to CloudWatch. Configure notification via SNS when application errors are noticed in the system logs. Configure the server AMI to use encrypted EBS volumes with a key from AWS KMS.
2. Provision an Application Load Balancer with an EIP in front of your EC2 target group and terminate SSL at the ALB. Install CloudWatch Logging agent on the EC2 instances and stream logs to CloudWatch. Configure notification via SNS when application errors are noticed in the system logs. Configure the server AMI to use encrypted EBS volumes with a key from AWS KMS.
3. Provision an Application Load Balancer in front of your EC2 target group and offload SSL to CloudHSM. Install CloudWatch Logging agent on the EC2 instances and stream logs to CloudWatch and configure notification via SNS when application errors are noticed in the system logs. Configure the server AMI to use encrypted EBS volumes with a key from CloudHSM.
4. Provision a Network Load Balancer with an EIP in front of your EC2 target group. Install the CloudWatch Logging agent on the EC2 instances and stream logs to CloudWatch. Configure notification via SNS when application errors are noticed in the system logs. Configure the server AMI to use encrypted EBS volumes with a key from AWS KMS. Terminate SSL on the EC2 instances.
5. Deploy the web application on Lambda with API Gateway as the front-end. Offload SSL termination using AWS KMS. Setup CloudWatch to alert via SNS if there are application exceptions. Encryption at rest is not required as there is no data stored in this architecture.
6. Deploy the web application on Lambda with API Gateway as the front-end. Enabled SSL termination on the API Gateway using Certificate Manager. Setup CloudWatch to alert via SNS if there are application exceptions. Encryption at rest is not required as there is no data stored in this architecture.

Answer: d

EXPLANATION:

The requirement of a static IP leads us to a Network Load Balancer with an EIP.