# 6-simulacoes profissionais

| | | | |
|---|---|---|---|
| **Notebook:** | solutions profissional_ef8e1c0f-5d3c-42ab-ab04-9217e53e12cd | | |
| **Created:** | 19/10/2025 18:36 | **Updated:** | 21/10/2025 01:06 |
| **Author:** | erikachen19@gmail.com | | |
| **URL:** | https://portal.tutorialsdojo.com/courses/aws-certified-solutions-architect-professional... | | |

22 A company plans to migrate its on-premises legacy application to AWS and develop a highly scalable application. Currently, all user requests are sent to the on-premises load balancer which forwards the requests to two Linux servers hosting the legacy application. The database is hosted on two servers in master-master configuration. Since this is an old application, the communication to the database servers is done through static IP addresses and not via DNS names. The license of the application is tied to the MAC address of the network adapter of the Linux server. If the application is to be installed on a new server, it will take about 15 hours for the software vendor to send the new license via email.

Which combination of actions must be done to meet the company requirements? (Select TWO.)

(view) 1 0 1 00:01:40

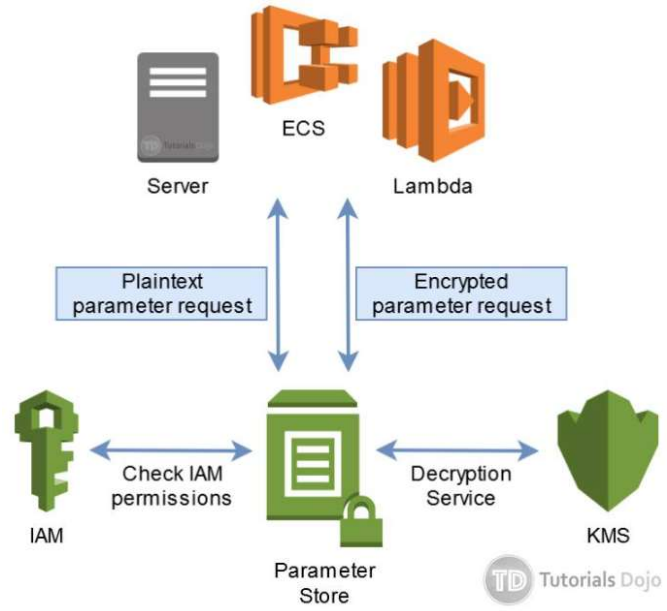
- [x] Provision a pool of Elastic Network Interfaces (ENIs). Request a license file for each ENI from the software vendor. Store the license files on an Amazon S3 bucket and use bootstrap scripts to retrieve an unused license file and attach corresponding ENI when provisioning EC2 instances.
- [ ] Provision a pool of Elastic Network Interfaces (ENIs). Request a license file for each ENI from the software vendor. Store the license files inside an Amazon EC2 instance and create a base AMI from this EC2 instance. Use bootstrap scripts to configure license keys and attach the corresponding ENI when provisioning EC2 instances.
- [ ] Create an Amazon EC2 bootstrap script that will resolve the database DNS names into IP addresses. Update the local configuration files with the resolved values.
- [x] Create an AWS Lambda function to update the database IP addresses on the Systems Manager Parameter Store. Create an Amazon EC2 bootstrap script that will retrieve the database IP address from SSM Parameter Store. Update the local configuration files with the parameters.
- [ ] Install the application on an EC2 instance and configure the needed license file. Update the local configuration with the database IP addresses. Use this instance as the base AMI for all instances in the Auto Scaling group.

**AWS Systems Manager Parameter Store** provides secure, hierarchical storage for configuration data management and secrets management. You can store data such as passwords, database strings, Amazon Machine Image (AMI) IDs, and license codes as parameter values. You can store values as plain text or encrypted data. You can reference Systems Manager parameters in your scripts, commands, SSM documents, and configuration and automation workflows by using the unique name that you specified when you created the parameter.

Parameter Store offers these benefits:

- You can use a secure, scalable, hosted secrets management service with no servers to manage.
- Improve your security posture by separating your data from your code.
- Store configuration data and encrypted strings in hierarchies and track versions.
- Control and audit access at granular levels.

Parameter Store provides support for three types of parameters: `String`, `StringList`, and `SecureString`.




An **elastic network interface (ENI)** is a logical networking component in a VPC that represents a virtual network card. It can include the following attributes:

- A primary private IPv4 address from the IPv4 address range of your VPC
- One or more secondary private IPv4 addresses from the IPv4 address range of your VPC
- One Elastic IP address (IPv4) per private IPv4 address
- One public IPv4 address
- One or more IPv6 addresses
- One or more security groups
- A MAC address
- A source/destination check flag
- A description

You can create and configure network interfaces in your account and attach them to instances in your VPC. Your account might also have requester-managed network interfaces, which are created and managed by AWS services to enable you to use other resources and services.

You can create a network interface, attach it to an instance, detach it from an instance, and attach it to another instance. The attributes of a network interface follow it as it's attached or detached from an instance and reattached to another instance. When you move a network interface from one instance to another, network traffic is redirected to the new instance.

Each instance has a default network interface, called the primary network interface. You cannot detach a primary network interface from an instance. You can create and attach additional network interfaces.

The option that says: **Provision a pool of Elastic Network Interfaces (ENIs). Request a license file for each ENI from the software vendor. Store the license files on an Amazon S3 bucket and use bootstrap scripts to retrieve an unused license file and attach corresponding ENI when provisioning EC2 instances** is correct. Having the license files on an Amazon S3 bucket reduces the management overhead for the EC2 instances, as you can easily add/remove more license keys if needed.

The option that says: **Create an AWS Lambda function to update the database IP addresses on the Systems Manager Parameter Store. Create an Amazon EC2 bootstrap script that will retrieve the database IP address from SSM Parameter Store. Update the local configuration files with the parameters** is correct. Having the database IP addresses on Parameter Store ensures that all the EC2 instances will have a central location to retrieve the IP addresses. This also reduces the need to constantly update any script from inside the EC2 instance even if you add/remove more databases in the future.

The option that says: **Provision a pool of Elastic Network Interfaces (ENIs). Request a license file for each ENI from the software vendor. Store the license files inside an Amazon EC2 instance and create a base AMI from this EC2 instance. Use bootstrap scripts to configure license keys and attach the corresponding ENI when provisioning EC2 instances** is incorrect. Although this is possible, this is not an ideal solution. If you need more EC2 instances and more ENIs, you will have to manually update your base AMI to include all the new license files. This can be a lot of work if you scale your cluster on a regular basis.

The option that says: **Create an Amazon EC2 bootstrap script that will resolve the database DNS names into IP addresses. Update the local configuration files with the resolved values** is incorrect. Although this is possible, this is not recommended. You will have to update the bootstrap script manually for the new DNS name every time you create a new database such as when you are scaling out your database instances.

The option that says: **Install the application on an EC2 instance and configure the needed license file. Update the local configuration with the database IP addresses. Use this instance as the base AMI for all instances in the Auto Scaling group** is incorrect. This will not work because the application license is tied to the MAC address on which it was installed. When you provision a new EC2 instance, it will have a new IP address and a newly assigned MAC address for its network adapter.

24 A data analytics company has recently adopted a hybrid cloud infrastructure with AWS. They are in the business of collecting and processing vast amounts of data. Each data set generates up to several thousands of files which can range from 10 MB to 1 GB in size. The archived data is rarely restored and in case there is a request to retrieve it, the company has a maximum of 24 hours to send the files. The data sets can be searched using its file ID, set name, authors, tags, and other criteria.

Which of the following options provides the most cost-effective architecture to meet the above requirements?

(view) 1 0 1 00:00:00

1. For each completed data set, compress and concatenate all of the files into a single Glacier archive. 2. Store the associated archive ID for the compressed files along with other search metadata in a DynamoDB table. 3. For retrieving the data, query the DynamoDB table for files that match the search criteria and then restore the files from the retrieved archive ID.

1. Store the files of the completed data sets into a single S3 bucket. 2. Store the S3 object key for the compressed files along with other search metadata in a DynamoDB table. 3. For retrieving the data, query the DynamoDB table for files that match the search criteria and then restore the files from the S3 bucket.

1. Store individually compressed files to an S3 bucket. Also, store the search metadata and the S3 object key of the files in a separate S3 bucket. 2. Create a lifecycle rule to move the data from an S3 Standard class to Glacier after a certain month. 3. For retrieving the data, query the S3 bucket for files matching the search criteria and then retrieve the file from the other S3 bucket.

1. Store individual files in Glacier using the filename as the archive name. 2. For retrieving the data, query the Glacier vault for files matching the search criteria.

**search criteria.**搜索条件。

**For each completed data set, compress and concatenate all of the files into a single Glacier archive.** 对于每个完成的数据集，将所有文件压缩并连接到单个 **Glacier** 档案中。

You can further lower the cost of storing data by compressing it to a zip or tar file. In addition, searching for archives in Glacier takes a long time, which is why it is advisable to store the search criteria and archive ID in a database for faster search. You can alternatively use Glacier Select to perform filtering operations using simple Structured Query Language (SQL).


The following option is incorrect because storing the data in an S3 Standard class is costly. It is more cost-effective to use Amazon Glacier instead.

**1. Store the files of the completed data sets into a single S3 bucket.**
**2. Store the S3 object key for the compressed files along with other search metadata in a DynamoDB table.**
**3. For retrieving the data, query the DynamoDB table for files that match the search criteria and then restore the files from the S3 bucket.**

The following option is incorrect because initially storing the archive to an S3 Standard class for a month still entails additional cost. Since the company allows a maximum of 24 hours to retrieve the files, you can directly store the archives in Glacier instead:

**1. Store individual compressed files to an S3 bucket. Also store the search metadata and the S3 object key of the files in a separate S3 bucket.**
**2. Create a lifecycle rule to move the data from an S3 Standard class to Glacier after a certain a month.**
**3. For retrieving the data, query the S3 bucket for files matching the search criteria and then retrieve the file from the other S3 bucket.**

The following option is incorrect because Glacier doesn't have a built-in search function to help you retrieve the data. You have to store the archive ID in a database, such as DynamoDB, to help you effectively search the required data:

**1. Store individual files in Glacier using the filename as the archive name.**
**2. For retrieving the data, query the Glacier vault for files matching the search criteria.**

25 A company hosts an e-commerce application on an Auto Scaling group (ASG) of EC2 instances behind an Application Load Balancer (ALB). The ASG is set to maintain 10 EC2 instances at all times. It uses Elastic Load Balancing (ELB) health checks to detect and replace any unhealthy instances. Each EC2 instance has a CloudWatch agent installed that sends application logs to Amazon CloudWatch Logs at 5-minute intervals.

The operations team is unable to recover logs that were stored in terminated instances. This prevents them from conducting proper troubleshooting. For this reason, they require a solution that can automatically collect and back up relevant log data from instances that are about to be terminated.

Which solution would meet the requirements?

(view) 1 0 1 00:01:43

Create a shell script for backing up log data to an S3 bucket. Install the script to the EC2 instance's user data. Run the script using a Lambda function via Remote Call Procedure (RPC). Create an Amazon EventBridge rule that invokes a Lambda function when an instance is in the `Terminating` state.

Change the interval at which the CloudWatch Logs agent sends data to 1 minute. Increase the deregistration delay of the Application Load Balancer. SSH into the instance to run the script for backing up the log data to an S3 bucket.

Add a lifecycle hook to the Auto Scaling Group for the `autoscaling:EC2_INSTANCE_TERMINATING` event and set the default result to `CONTINUE`. Implement a script using an AWS Systems Manager Automation document to backup log data to an Amazon S3 bucket. Create an Amazon EventBridge rule that invokes a Lambda function when an instance is in the `Terminating:Wait` state. Configure the function to call the `SendCommand` API to run the automation document.

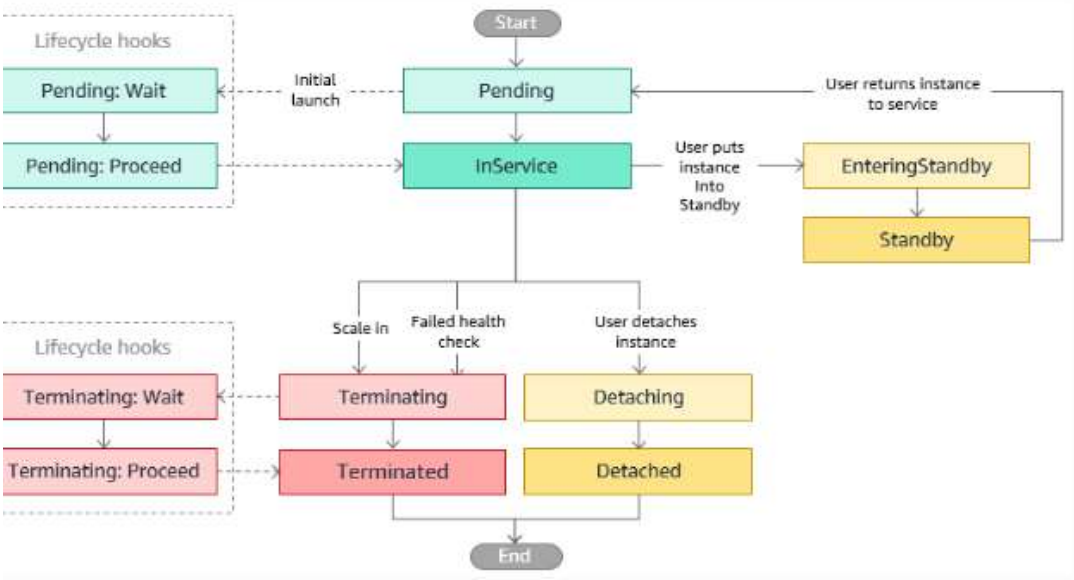
Use an AWS Systems Manager Automation document to create the script for backing up log data to an Amazon S3 bucket. Create an OpsItem for EC2 termination events in AWS Systems Manager OpsCenter. Configure a remediation runbook that calls the `CompleteLifecycleAction` Auto Scaling API with `ABANDON` to pause the termination of instance and `SendCommand` System Manager API to run the automation document.

The EC2 instances in an Auto Scaling group have a path, or lifecycle, that differs from that of other EC2 instances. The lifecycle starts when the Auto Scaling group launches an instance and puts it into service. The lifecycle ends when you terminate the instance, or the Auto Scaling group takes the instance out of service and terminates it.

You can add a lifecycle hook to your Auto Scaling group so that you can perform custom actions when instances launch or terminate.

When Amazon EC2 Auto Scaling responds to a scale out event, it launches one or more instances. These instances start in the `Pending` state. If you added an `autoscaling:EC2_INSTANCE_LAUNCHING` lifecycle hook to your Auto Scaling group, the instances move from the `Pending` state to the `Pending:Wait` state. After you complete the lifecycle action, the instances enter the `Pending:Proceed` state. When the instances are fully configured, they are attached to the Auto Scaling group and they enter the `InService` state.

When Amazon EC2 Auto Scaling responds to a scale in event, it terminates one or more instances. These instances are detached from the Auto Scaling group and enter the `Terminating` state. If you added an `autoscaling:EC2_INSTANCE_TERMINATING` lifecycle hook to your Auto Scaling group, the instances move from the `Terminating` state to the `Terminating:Wait` state. After you complete the lifecycle action, the instances enter the `Terminating:Proceed` state. When the instances are fully terminated, they enter the `Terminated` state.




In the scenario, we can use AWS Auto Scaling lifecycle hooks, specifically the `autoscaling:EC2_INSTANCE_TERMINATING` event, to intercept the instance before termination. During this intercepted state, AWS Systems Manager Automation can run a script to backup logs to an Amazon S3 bucket. To automate this process, an Amazon EventBridge rule can be set to trigger a Lambda function when the instance enters the `Terminating:Wait` state. This Lambda function can then call the `SendCommand` API to run the automation document, ensuring that the logs are backed up before termination.

The option that says: **Create a shell script for backing up log data to an S3 bucket. Install the script to the EC2 instance's user data. Run the script using a Lambda function via Remote Call Procedure (RPC). Create an Amazon EventBridge rule that invokes a Lambda function when an instance is in the `Terminating` state** is incorrect. This approach does not guarantee the script to be executed before the instance is terminated. Termination can occur very quickly, and it's very likely that the script won't have time to run and back up the logs.

The option that says: **Change the interval at which the CloudWatch Logs agent sends data to 1 minute. Increase the deregistration delay of the Application Load Balancer. SSH into the instance to run the script for backing up the log data to an S3 bucket** is incorrect. Having to SSH into instances manually isn't a scalable or reliable solution. If there are many instances, or if the instances are frequently scaled in or out, this method becomes unmanageable. Moreover, it doesn't provide a solution for automatic collection and backup of logs.

The option that says: **Use an AWS Systems Manager Automation document to create the script for backing up log data to an Amazon S3 bucket. Create an OpsItem for EC2 termination events in AWS Systems Manager OpsCenter. Configure a remediation runbook that calls the `CompleteLifecycleAction` Auto Scaling API with `ABANDON` to pause the termination of instance and `SendCommand` System Manager API to run the automation document** is incorrect. In this solution, the runbook remediation is triggered upon instance termination events. This means that the termination may have already proceeded past a point where it could be halted so the backup script can be executed. Moreover, the use of Systems Manager OpsCenter adds unnecessary complexity to the solution. OpsCenter is more oriented towards incident management rather than an operational activity like capturing logs.

26 A tech company that performs large-scale analytics uses an Amazon Redshift cluster with reserved nodes to support its daily data workloads. A recently hired director has initiated a new requirement: generating a deep audit analysis report. This new reporting initiative has introduced complex read queries that are highly CPU-intensive, resulting in unexpected bursts of usage on the Redshift cluster.

The existing setup must continue to handle both read and write queries consistently, even during performance spikes. A solutions architect must propose the most cost-effective solution to address the rising CPU metrics while maintaining system responsiveness and minimizing operational cost.

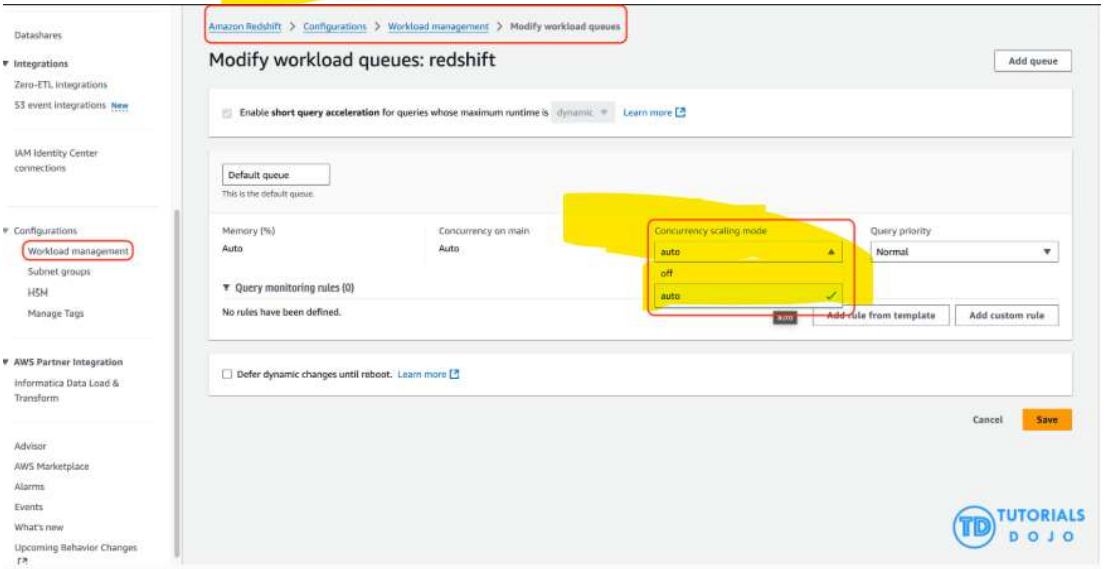
What is the most economical approach to manage the increased workload introduced by this new reporting initiative?

(view)

- Deploy an Amazon EMR cluster to offload high-load transformation jobs from the main analytics pipeline.
- Enable concurrency scaling in the Redshift setup to handle spikes in concurrent query loads without impacting performance.
- Launch additional Amazon EC2 instances to scale horizontally Redshift for query parallelization.
- Export data from Redshift and import it into Amazon RDS PostgreSQL for deep audit analysis during periods of high load.

With **Amazon Redshift's Concurrency Scaling** feature, you can support thousands of concurrent users and queries with consistently fast query performance. When you turn on concurrency scaling, Redshift automatically adds additional cluster capacity to process an increase in both read and write queries. Users see the most current data, whether the queries run on the main or concurrency-scaling clusters.

You're charged for concurrency-scaling clusters only when they're actively running queries.




Enabling concurrency scaling in Amazon Redshift is the most cost-efficient and operationally elegant solution to address CPU-intensive workload spikes, such as the deep audit analysis report. This feature is designed for situations where query demand temporarily exceeds cluster capacity. It works by automatically adding transient clusters to handle excess read workloads, ensuring that the main cluster can continue to service both read and write operations without latency or disruption. The beauty of this approach lies in its simplicity. This ensures the system remains responsive without permanently overprovisioning resources.

Hence, the correct answer is: **Enable concurrency scaling in the Redshift setup to handle spikes in concurrent query loads without impacting performance**.

The option that says: **Deploy an Amazon EMR cluster to offload high-load transformation jobs from the main analytics pipeline** is incorrect. Amazon EMR is robust for distributed data processing, but would only introduce unnecessary complexity in this case. The issue is not about transformation or ETL pipelines. It is about read query concurrency and CPU bottlenecks within the Redshift cluster itself. Offloading workloads to EMR simply doesn't solve the immediate pressure on Redshift's compute layer, nor is it primarily focused on short-term spikes. Additionally, EMR adds extra infrastructure to manage and incurs costs, making it neither the simplest nor the most cost-effective option.

The option that says: **Launch additional Amazon EC2 instances to scale horizontally Redshift for query parallelization** is incorrect. Scaling Amazon Redshift by launching EC2 instances fundamentally misunderstands how Redshift is designed to operate. Redshift is a managed data warehouse that cannot be scaled horizontally through EC2 the way traditional application servers might be. Redshift scaling is handled through built-in features like Elastic Resize or Concurrency Scaling, not EC2 fleets. Trying to scale Redshift with EC2 is not just incorrect; it's not even technically possible. This option simply doesn't apply to the way Redshift architecture works.

The option that says: **Export data from Redshift and import it into Amazon RDS PostgreSQL for deep audit analysis during periods of high load** is incorrect. Exporting Redshift data to Amazon RDS PostgreSQL during high-load periods may sound like an alternative strategy, but it's a step in the wrong direction. RDS PostgreSQL is designed primarily for transactional workloads, not for handling CPU-heavy, complex analytical queries. Attempting to mirror Redshift's analytical capabilities in RDS would result in performance degradation, additional data movement pipelines, and more operational overhead. This approach is not only technically mismatched but also simply adds more cost and complexity without addressing the core issues of query concurrency and CPU pressure within Redshift itself.

1 An IT company provides on-demand video training materials to its employees. High-resolution MP4 format videos are created monthly to increase the topics covered in the video library. Most employees that will view the videos are working remotely from different countries. The video application installed on the employee laptops requires the videos to be in HTTP Live Streaming (HLS) format. The management asked the solutions architect to design a cost-effective solution that will transcode the training videos into the proper format.

Which of the following architectures meet the above requirements while maintaining high availability and good quality video transmission?

(view) 1 1 0 00:00:00

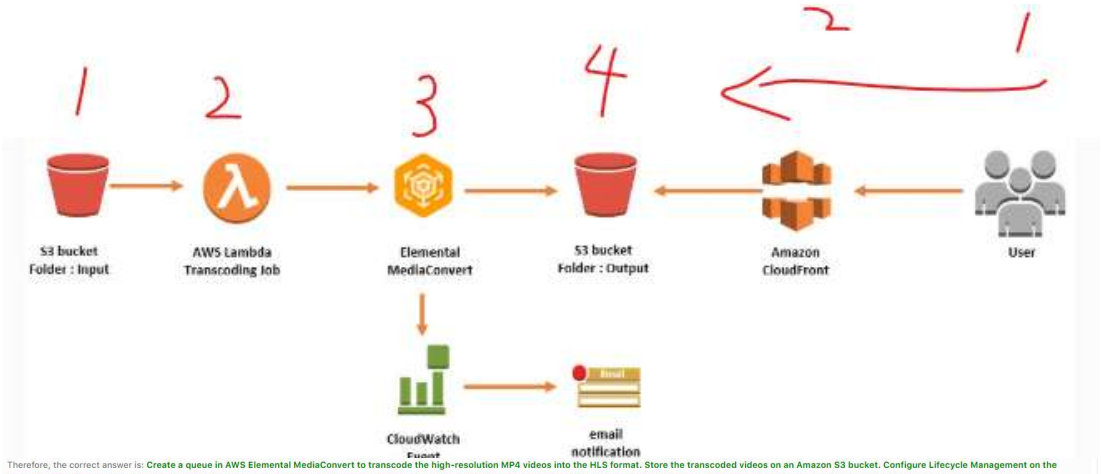
- Create a Jenkins pipeline to transcode the high-resolution MP4 videos. Use an SQS queue to distribute the transcoding job to a set of Amazon EC2 instances in an Auto Scaling group that scales based on the length of the queue. Store the transcoded videos on EBS volumes and configure automated snapshots to back up the files every few days. Create an Amazon CloudFront distribution and set the EC2 instances as the origin to serve the transcoded videos.
- Create a pipeline on Amazon Elastic Transcoder to transcode the high-resolution MP4 videos into the HLS format. Host the transcoded videos on large EBS volumes attached to Amazon EC2 instances. Configure automated EBS snapshots to back up the video files every few days. Create an Amazon CloudFront distribution and set the EC2 instances as the origin to serve the transcoded videos.
- Create a queue in AWS Elemental MediaConvert to transcode the high-resolution MP4 videos into the HLS format. Store the transcoded videos on an Amazon S3 bucket. Configure Lifecycle Management on the bucket to move the original videos to Amazon S3 Glacier Instant Retrieval. Create an Amazon CloudFront distribution and set the S3 bucket as the origin to serve the transcoded videos.
- Create a pipeline in AWS CodePipeline with stages that transcode the high-resolution MP4 videos into the HLS format and send the transcoded videos to an Amazon S3 bucket. Configure Lifecycle Management on the bucket to move the original videos to Amazon S3 Glacier Instant Retrieval. Then, set up an endpoint in AWS Wavelength and configure it to serve the transcoded videos from the S3 bucket.

AWS Elemental MediaConvert is a file-based video transcoding service with broadcast-grade features. It provides a comprehensive suite of advanced transcoding features with on-demand rates. It allows you to easily create video-on-demand (VOD) content for broadcast and multiscreen delivery at scale. The service combines advanced video and audio capabilities with a simple web services interface and pay-as-you-go pricing. With AWS Elemental MediaConvert, you can focus on delivering compelling media experiences without having to worry about the complexity of building and operating your own video processing infrastructure. MediaConvert is optimized to improve scalability, which allows you to process more files in parallel.

Amazon Elastic Transcoder uses pipelines to manage transcoding jobs. When you create a job, you specify the pipeline that you want to submit the job to. Pipelines are closely tied to an S3 bucket that you specify. Queues in AWS Elemental MediaConvert are similar to pipelines. The number of concurrent jobs processed in a given queue is determined based on the number of queues in your account. However, one of the key differences is that queues are not tied to a specific S3 bucket. This means that you can submit jobs that reference input files in different S3 buckets to the same queue. In Elastic Transcoder, you select the playlist version upon job creation. In MediaConvert, the HLS version will change as you enable features that require a specific version.

AWS Elemental MediaConvert 是一项基于文件的视频转码服务，具备广播级功能。它提供一整套先进的转码功能，并支持按需收费。它让您能够轻松创建**视频点播 (VOD) 内容**，用于大**规模广播和多屏传输**。该服务将先进的视频和音频功能与简单的 Web 服务界面以及按需付费的定价模式相结合。借助 AWS Elemental MediaConvert，您可以专注于提供引人入胜的媒体体验，而无需担心构建和运营自己的视频处理基础设施的复杂性。MediaConvert 经过优化，可扩展性更高，让您能够并行处理更多文件。

Amazon Elastic Transcoder 使用**管道来管理转码作业**。创建作业时，您可以指定要将作业提交到的管道。管道与您指定的 S3 存储桶紧密相关。AWS Elemental **MediaConvert 中的队列**与管道类似。给定队列中**处理的并发作业数取决于您账户中的队列数**。然而，一个关键的区别是**队列并不绑定到特定的 S3 存储桶**。这意味着您可以将引用不同 S3 存储桶中输入文件的作业提交到同一队列。在 Elastic Transcoder 中，您可以在创建作业时选择播放列表版本。在 MediaConvert 中，HLS 版本会随着您启用需要特定版本的功能而变化。




Therefore, the correct answer is: **Create a queue in AWS Elemental MediaConvert to transcode the high-resolution MP4 videos into the HLS format. Store the transcoded videos on an Amazon S3 bucket. Configure Lifecycle Management on the bucket to move the original videos to Amazon S3 Glacier Instant Retrieval. Create an Amazon CloudFront distribution and set the S3 bucket as the origin to serve the transcoded videos.** AWS Elemental MediaConvert is a new file-based video transcoding service aimed to replace Amazon Elastic Transcoder. Additionally, storing the transcoded videos on S3 buckets is highly durable and cost-effective, as you don't have to pay for EC2 instance costs.

The option that says: **Create a Jenkins pipeline to transcode the high-resolution MP4 videos. Use an SQS queue to distribute the transcoding job to a set of Amazon EC2 instances in an Auto Scaling group that scales based on the length of the queue. Store the transcoded videos on EBS volumes and configure automated snapshots to back up the files every few days. Create an Amazon CloudFront distribution and set the EC2 instances as the origin to serve the transcoded videos** is incorrect. AWS has managed services to automate video transcoding for you. Additionally, using large EBS volumes to store the transcoded videos is not very cost-effective compared to using an S3 bucket.

The option that says: **Create a pipeline on Amazon Elastic Transcoder to transcode the high-resolution MP4 videos into the HLS format. Host the transcoded videos on large EBS volumes attached to Amazon EC2 instances. Configure automated EBS snapshots to back up the video files every few days. Create an Amazon CloudFront distribution and set the EC2 instances as the origin to serve the transcoded videos** is incorrect. This may be possible, however, using large EBS volumes to store the transcoded videos is not very cost-effective compared to using an S3 bucket. Additionally, the performance of the EC2 instance may vary depending on the traffic load.

The option that says: **Create a pipeline in AWS CodePipeline with stages that transcode the high-resolution MP4 videos into the HLS format and send the transcoded videos to an Amazon S3 bucket. Configure Lifecycle Management on the bucket to move the original videos to Amazon S3 Glacier Instant Retrieval. Then, set up an endpoint in AWS Wavelength and configure it to serve the transcoded videos from the S3 bucket** is incorrect. AWS CodePipeline is primarily designed for CI/CD pipelines to deploy new application versions. You can't directly create a stage that will transcode videos to a specific format. Moreover, AWS Wavelength is designed to provide ultra-low latency for applications on 5G networks. It's ideal for applications that require minimal latency over mobile networks. However, in your scenario, the employees are located in various countries and may not necessarily be using 5G networks.

2 A startup develops a new web application that manages job listings for contractors and seasonal workers. The application source code is stored on a GitHub repository. For continuous integration, the developers trigger a pipeline on AWS CodePipeline. This pipeline includes a build stage that generates an artifact that is sent to an Amazon S3 bucket. The developers want to improve this process and have requested the solutions architect to create a strategy that meets the following requirements:

- Create a pipeline to accommodate Feature-Driven Development (FDD) process for the app
- Feature development must not affect the existing production application
- Must include a stage for continuous unit testing
- Create separate development and production artifacts
- Maintain the capability of integrating development and production code.

Which of the following options is the recommended solution to meet the company's requirements?

(view) 1 1 0 00:00:00

Create a separate pipeline on CodePipeline that is triggered by the "feature" branch from GitHub. Add a stage for AWS CodeBuild to run unit testing and stage the generated artifacts in an S3 bucket in a separate testing AWS account.

Create a separate pipeline on CodePipeline that is triggered by the "feature" branch from GitHub. Add a stage that will trigger AWS Lambda functions to perform unit testing on the generated artifacts. Use AWS CodeDeploy to stage the artifacts on an S3 bucket in a separate testing AWS account.

Create a new pipeline in Jenkins that is triggered by the "feature" tags from GitHub. Add a stage that will run unit testing on the generated artifacts. Send the artifacts to a staging S3 bucket in a separate AWS account using the AWS Well-Architected Tool.

Create a new repository in GitHub for features development. Use Amazon CodeGuru to review and improve your code. Then, use the commits on this repo as triggers for AWS CodeBuild to build and test the artifacts. Use AWS CodeDeploy to stage the artifacts on a separate S3 bucket in the production AWS account.

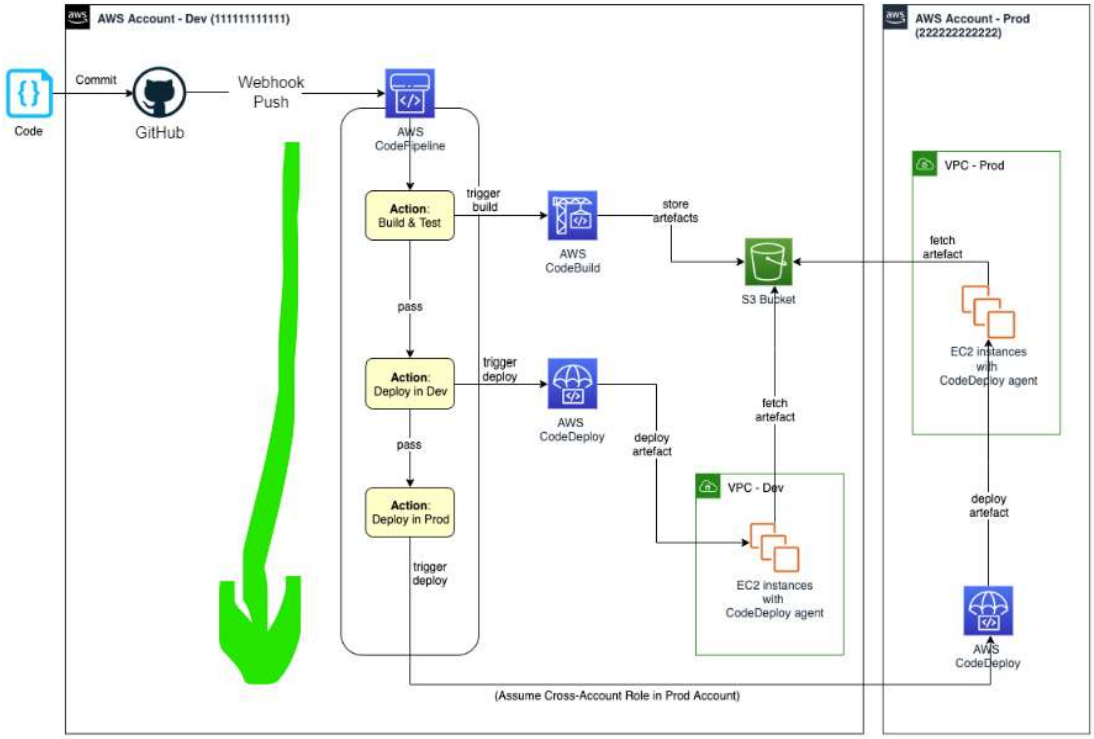
**AWS CodePipeline** is a continuous delivery service that you can use to model, visualize, and automate the steps required to release your software. You can quickly model and configure the different stages of a software release process. CodePipeline automates the steps required to release your software changes continuously.

**AWS CodeBuild** is a fully managed build service in the cloud. CodeBuild compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. CodeBuild eliminates the need to provision, manage, and scale your own build servers.

You can automate your release process by using AWS CodePipeline to test your code and run your builds with AWS CodeBuild. You can use the AWS suite of CI/CD services to compile, build, and install a version-controlled Java application onto a set of Amazon Elastic Compute Cloud (Amazon EC2) Linux instances via a fully automated and secure pipeline. With these tools, you can promote a code commit or change to pass through various automated stage gates all the way from development to production environments across AWS accounts.

The following diagram illustrates the workflow:

- A change or commit to the code in the GitHub repository triggers CodePipeline through a webhook event.
- The pipeline downloads the code from the GitHub repository, initiates the Build and Test action using CodeBuild, and securely saves the built artifact on the S3 bucket.
- If the preceding step is successful, the pipeline triggers the Deploy in Dev action using CodeDeploy and deploys the app in the dev account.
- If successful, the pipeline triggers the Deploy in Prod action using CodeDeploy and deploys the app in the prod account.




The option that says: **Create a separate pipeline on CodePipeline that is triggered by the "feature" branch from GitHub. Add a stage that will trigger AWS Lambda functions to perform unit testing on the generated artifacts. Use AWS CodeDeploy to stage the artifacts on an S3 bucket in a separate testing AWS account** is incorrect. This is possible; however, it is not necessary to write Lambda code for unit testing. You just need to include the tests on the CodeBuild stage, which will perform the unit testing.

The option that says: **Create a new pipeline in Jenkins that is triggered by the "feature" tags from GitHub. Add a stage that will run unit testing on the generated artifacts. Send the artifacts to a staging S3 bucket in a separate AWS account using the AWS Well-Architected Tool** is incorrect. Using a separate Jenkins instance to run your pipeline is not recommended. AWS CodePipeline is an AWS-managed service that can automate code builds, unit testing, and deployments on your behalf. Moreover, the AWS Well-Architected Tool simply allows you to monitor the status of multiple workloads across your organization and helps you understand potential risks. This is not meant to send artifacts to a staging S3 bucket located in a separate AWS account.

The option that says: **Create a new repository in GitHub for features development. Use Amazon CodeGuru to review and improve your code. Then, use the commits on this repo as triggers for AWS CodeBuild to build and test the artifacts. Use AWS CodeDeploy to stage the artifacts on a separate S3 bucket in the production AWS account** is incorrect. You don't need to create a separate repository for your feature development. This will just duplicate your GitHub storage and can add complexity to your deployments. It is recommended to create a feature branch instead and use it to test and deploy new features. Moreover, while CodeGuru can help improve code quality, it's not explicitly required to meet the stated requirements. The scenario doesn't mention the need for code review or performance optimization, which are the primary functions of CodeGuru. Therefore, while CodeGuru could be beneficial in a general sense for improving code quality, it's not strictly necessary based on the specific requirements given in the question.

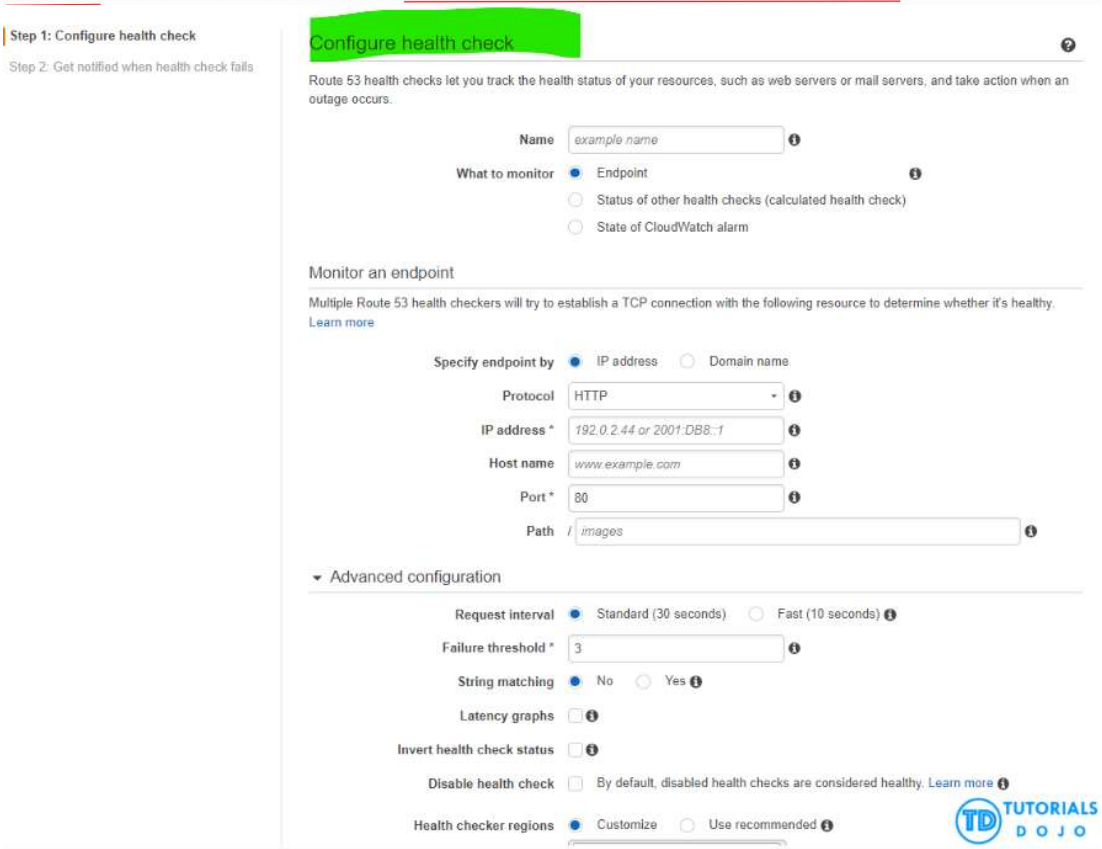
3 An e-commerce company runs its website on Amazon EC2 instances. Each of the two AWS Regions has its own set of instances managed by an Application Load Balancer (ALB). The company wants to ensure that customers are served by the region closest to them for the fastest response times. Customers should be served from the other Region if the website becomes unavailable in the closest Region.

Which of the following options should the solutions architect recommend to meet these requirements?

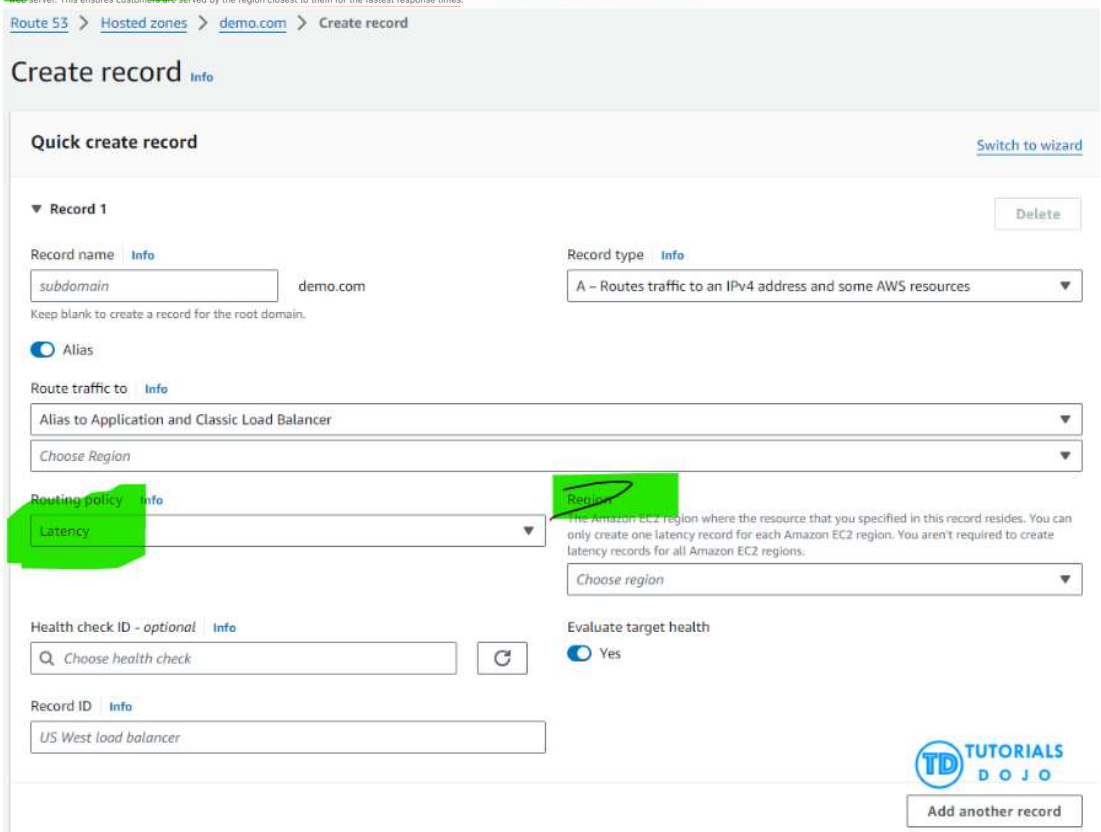
(view)

- Set up a CloudFront distribution with an origin group that includes the EC2 instances in both regions.
- Use AWS Global Accelerator to improve the availability and performance of the applications.
- Use Amazon Route 53 health checks to monitor the availability of the EC2 instances in both regions and Route 53 failover routing to redirect traffic in case of an outage.
- Set up Amazon Route 53 health checks to monitor the availability of the EC2 instances in both regions. Use Route 53 latency alias records to redirect traffic.

Amazon Route 53 health checks monitor the health of your resources, such as web servers and email servers. You can specify values that define how you want the health check to work, such as the IP address or domain name of the endpoint that you want Route 53 to monitor, the protocol that you want Amazon Route 53 to use to perform the check, how often you want Route 53 to send a request to the endpoint, and how many consecutive times the endpoint must fail to respond to requests before Route 53 considers it unhealthy. If a health check determines that the underlying resource is unhealthy, Route 53 routes traffic from the associated record. This ensures high availability and failover support for DNS records.




Latency-based routing in Amazon Route 53 allows you to improve performance for your users by serving their requests from the AWS Region that provides the lowest latency. When Route 53 receives a DNS query for your domain or subdomain, it determines which AWS Regions you've created latency records for, determines which Region gives the user the lowest latency, and then selects a latency record for that Region. Route 53 responds with the value from the selected record, such as the IP address for a web server. This ensures customers are served by the region closest to them for the fastest response times.




Hence, the correct answer is: **Set up Amazon Route 53 health checks to monitor the availability of the EC2 instances in both regions. Use Route 53 latency alias records to redirect traffic.** This ensures that customers are served by the region closest to them, and if the website becomes unavailable in the closest Region, customers are served from the other Region.

The option that says: **Set up a CloudFront distribution with an origin group that includes the EC2 instances in both regions** is incorrect because CloudFront primarily routes traffic based on latency and does not automatically route traffic based on the health of the application or the region.

The option that says: **Use AWS Global Accelerator to improve the availability and performance of the applications** is incorrect because while AWS Global Accelerator does improve the availability and performance of your applications, it does not inherently provide failover capabilities between regions based on the health of your application.

The option that says: **Use Amazon Route 53 health checks to monitor the availability of the EC2 instances in both regions and Route 53 failover routing to redirect traffic in case of an outage** is incorrect because while it provides a failover mechanism, it does not ensure that customers are served by the region closest to them for the fastest response times.

6 A company has several IoT enabled devices and sells them to customers around the globe. Every 5 minutes, each IoT device sends back a data file that includes the device status and other information to an Amazon S3 bucket. Every midnight, a Python cron job runs from an Amazon EC2 instance to read and process each data file on the S3 bucket and loads the values on a designated Amazon RDS database. The cron job takes about 10 minutes to process a day's worth of data. After each data file is processed, it is eventually deleted from the S3 bucket. The company wants to expedite the process and access the processed data on the Amazon RDS as soon as possible.

Which of the following actions would you implement to achieve this requirement with the LEAST amount of effort?

(view) 1 1 0 00:04:25

- Increase the Amazon EC2 instance size and spawn more instances to speed up the processing of the data files. Set the Python script cron job schedule to a 1-minute interval to further improve the access time.
- Convert the Python script cron job to an AWS Lambda function. Configure AWS CloudTrail to log data events of the Amazon S3 bucket. Set up an Amazon EventBridge rule to trigger the Lambda function whenever an upload event on the S3 bucket occurs.
- Convert the Python script cron job to an AWS Lambda function. Create an Amazon EventBridge rule scheduled at 1-minute intervals and trigger the Lambda function. Create parallel CloudWatch rules that trigger the same Lambda function to further reduce the processing time.
- Convert the Python script cron job to an AWS Lambda function. Configure the Amazon S3 bucket event notifications to trigger the Lambda function whenever an object is uploaded to the bucket.

expedite 加快the process and access the processed data on the Amazon RDS as soon as possible.加快流程并尽快访问 Amazon RDS 上处理的数据。

7 A company develops Docker containers to host web applications on its on-premises data center. The company wants to migrate its workload to the cloud and use AWS Fargate. The solutions architect has created the necessary task definition and service for the Fargate cluster. For security requirements, the cluster is placed on a private subnet in the VPC that has no direct connection outside of the VPC. The following error is received when trying to launch the Fargate task:

```
CannotPullContainerError: API error (500): Get https://111122223333.dkr.ecr.us-east-1.amazonaws.com/v2/: net/http: request canceled while waiting for connection
```

Which of the following options should be able to fix this issue?

(view) 1 1

- Update the AWS Fargate task definition and set the auto-assign public IP option to ENABLED. Create a gateway VPC endpoint for Amazon ECR. Update the route table to allow AWS Fargate to pull images on Amazon ECR via the endpoint.
- Update the AWS Fargate task definition and set the auto-assign public IP option to DISABLED. Launch a NAT gateway on the public subnet of the VPC and update the route table of the private subnet to route requests to the Internet.
- Update the AWS Fargate task definition and set the auto-assign public IP option to DISABLED. Launch a NAT gateway on the private subnet of the VPC and update the route table of the private subnet to route requests to the Internet.
- This is a limitation of the " `awsvpc` " network mode. Update the AWS Fargate definition to use the " `bridge` " network mode instead to allow connections to the Internet.

The `CannotPullContainer error (500)` is caused by the `Connection timed out` when connecting to Amazon ECR. This indicates that when creating a task, the container image specified could not be retrieved.

When a Fargate task is launched, its elastic network interface requires a route to the Internet to pull container images. If you receive an error similar to the following when launching a task, it is because a route to the Internet does not exist:

```
CannotPullContainerError: API error (500): Get https://111122223333.dkr.ecr.us-east-1.amazonaws.com/v2/: net/http: request canceled while waiting for connection"
```

To resolve this issue, you can:

- For tasks in public subnets, specify **ENABLED** for **Auto-assign public IP** when launching the task.
- For tasks in private subnets, specify **DISABLED** for **Auto-assign public IP** when launching the task, and configure a NAT gateway in your VPC to route requests to the Internet.

Therefore, the correct answer is: **Update the AWS Fargate task definition and set the auto-assign public IP option to DISABLED. Launch a NAT gateway on the public subnet of the VPC and update the route table of the private subnet to route requests to the Internet.** The NAT gateway in the public subnet should have a public IP address and a route to the Intenet Gateway. The tasks in the private subnet will send internet traffic to the NAT gateway to be able to pull the images on Amazon Elastic Container Registry.

The option that says: **Update the AWS Fargate task definition and set the auto-assign public IP option to ENABLED. Create a gateway VPC endpoint for Amazon ECR. Update the route table to allow AWS Fargate to pull images on Amazon ECR via the endpoint** is incorrect. Since the Fargate tasks are on private subnet, you don't need to enable the auto-assign public IP option. Additionally, you should interface VPC endpoint, not gateway VPC endpoint.

The option that says: **Update the AWS Fargate task definition and set the auto-assign public IP option to DISABLED. Launch a NAT gateway on the private subnet of the VPC and update the route table of the private subnet to route requests to the Internet** is incorrect. The NAT gateway should be placed in a public subnet because it needs a Public IP address and a direct route to the Internet Gateway (IGW). If it is placed on a private subnet, it will have the same routing limitation as those resources in the private subnet.

The option that says: **This is a limitation of the " `awsvpc` " network mode. Update the AWS Fargate definition to use the " `bridge` " network mode instead to allow connections to the Internet** is incorrect. AWS Fargate only supports the " `awsvpc` " network mode. Each task is allocated its own elastic network interface (ENI) that is used for communication inside the VPC.

A company develops new android and iOS mobile apps. The company is considering storing user customization data in AWS. This would provide a more uniform cross-platform experience to their users using multiple mobile devices to access their apps. The preference data for each user is estimated to be 4 KB in size. Additionally, 3 million customers are expected to use the application on a regular basis, using their social login accounts for easier user authentication.

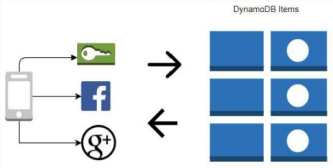
How should the Solutions Architect design a highly available, cost-effective, scalable, and secure solution to meet the above requirements?

(view) 1 0 1 00:00:00

- Have the user preference data stored in S3, and set up a DynamoDB table with an item for each user and an item attribute referencing the user's S3 object. The mobile app will retrieve the S3 URL from DynamoDB and then access the S3 object directly utilizing STS, Web identity federation, and S3 Access Points.
- Provision a table in DynamoDB containing an item for each user having the necessary attributes to hold the user preferences. The mobile app will query the user preferences directly from the table. Use STS, Web Identity Federation, and DynamoDB's Fine-Grained Access Control for authentication and authorization.
- Launch an RDS MySQL instance in 2 availability zones to contain the user preference data. Deploy a public-facing application on a server in front of the database which will manage authentication and access controls.
- Create an RDS MySQL instance with multiple read replicas in 2 availability zones to store the user preference data. The mobile application will then query the user preferences from the read replicas. Finally, utilize MySQL's user management and access privilege system to handle the security and access credentials of the users.

DynamoDB's **Fine-Grained Access Control** for authentication and authorization.,
DynamoDB 的细粒度访问控制用于身份验证和授权。

Take note that the question mentioned the use of social media logins. With web identity federation, you don't need to create custom sign-in code or manage your own user identities. Instead, users of your app can sign in using a well-known identity provider (IdP) —such as Login with Amazon, Facebook, Google, or any other OpenID Connect (OIDC)-compatible IdP, receive an authentication token, and then exchange that token for temporary security credentials in AWS that map to an IAM role with permissions to use the resources in your AWS account. Using an IdP helps you keep your AWS account secure, because you don't have to embed and distribute long-term security credentials with your application.




The option that says: **Provision a table in DynamoDB containing an item for each user having the necessary attributes to hold the user preferences. The mobile app will query the user preferences directly from the table. Use STS, Web Identity Federation, and DynamoDB's Fine Grained Access Control for authentication and authorization** is correct because it uses DynamoDB for scalability and cost-efficiency. It uses federated access using Web Identity Provider, and uses fine-grained access privileges for authenticating the access.

The option that says: **Have the user preference data stored in S3, and set up a DynamoDB table with an item for each user and an item attribute referencing the user's S3 object. The mobile app will retrieve the S3 URL from DynamoDB and then access the S3 object directly utilizing STS, Web identity Federation, and S3 Access Points** is incorrect because it doesn't use DynamoDB's built-in Fine-Grained Access Control for authentication and authorization feature. Additionally, using Amazon S3 bucket with DynamoDB is recommended for storing large data with the metadata stored on DynamoDB. The user preference data is only 4KB in size which can be stored effectively in a DynamoDB table. The use of S3 Access points is not necessary because these are just unique hostnames that enforce distinct permissions and network controls for any request made through the access point.

The option that says: **Launch an RDS MySQL instance in 2 availability zones to contain the user preference data. Deploy a public-facing application on a server in front of the database which will manage authentication and access controls** is incorrect because RDS MySQL is not as scalable and cost-effective as DynamoDB.

The option that says: **Create an RDS MySQL instance with multiple read replicas in 2 availability zones to store the user preference data. The mobile application will then query the user preferences from the read replicas. Finally, utilize MySQL's user management and access privilege system to handle the security and access credentials of the users** is incorrect because RDS MySQL is not as scalable and cost-effective as DynamoDB. Additionally, the user management and access privilege system for RDS cannot be used for controlling access.

DynamoDB's built-in Fine-Grained Access Control for authentication and authorization feature

**Fine-Grained Access Control (FGAC)** in **Amazon DynamoDB** is a feature that lets you **control access to individual items or attributes** in a DynamoDB table — not just at the table level.
It provides **authentication and authorization** at a *more detailed level* than traditional IAM policies.

### 📋 Example Policy

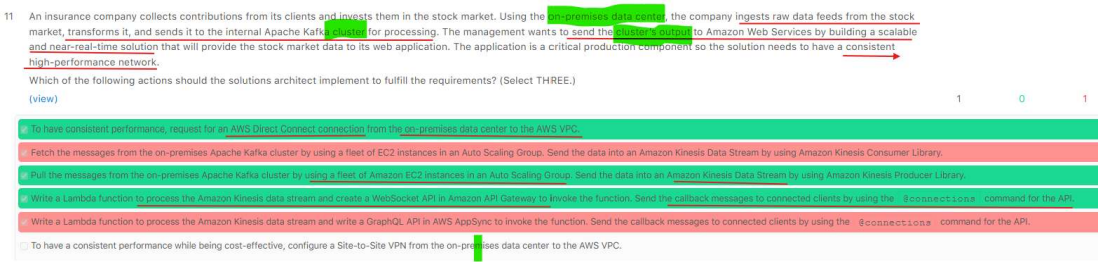
This example allows a user to **read only their own record** in the `Users` table:

json Copy code

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["dynamodb:GetItem", "dynamodb:Query"],
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/Users",
      "Condition": {
        "ForAllValues:StringEquals": {
          "dynamodb:LeadingKeys": ["${aws:username}"]
        }
      }
    }
  ]
}
```

🟢 Here:

- Each IAM user can **only access the item** where the **partition key** ( `UserId` ) equals their **username.**




the company ingests raw data feeds 获取from the stock market该公司从股票市场获取原始数据

ingestion row data ->transformation-> send -> kafka cluster > aws Web Serivce -> stock market data-> web aplication

kinesis data stream -> lambda process data stream ->api gateway -> client

**Amazon Kinesis Data Streams (KDS)** is a **real-time data streaming service** in AWS that lets you **collect, process, and analyze data continuously and at scale —** as the data is generated.

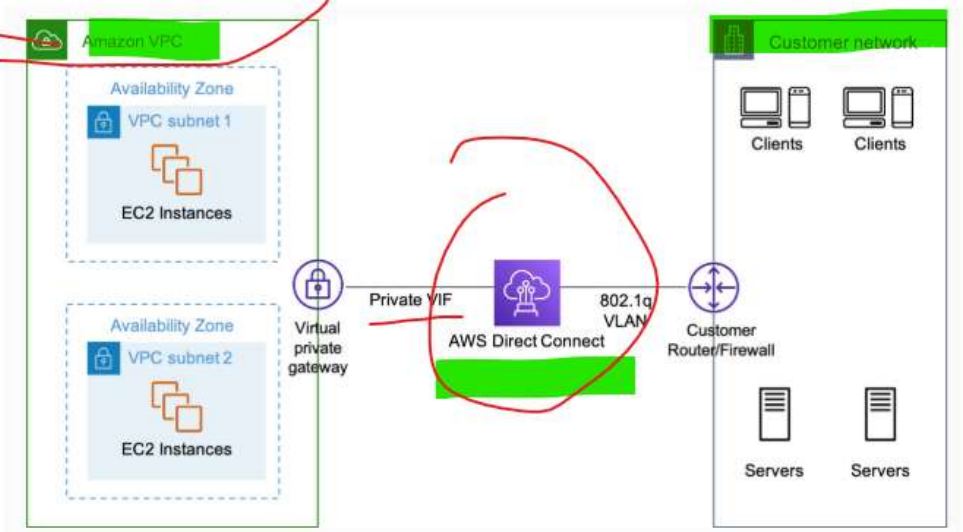
It's designed for use cases where you need to handle **high-throughput, real-time data ingestion** such as logs, metrics, clickstreams, IoT data, or financial transactions.

[Data Producers]
  ↓
[Kinesis Data Stream (shards)]
  ↓
[Data Consumers]
  ↓
[Processing / Storage: Lambda, Kinesis Data Analytics, S3, Redshift, etc.]

## How it works:

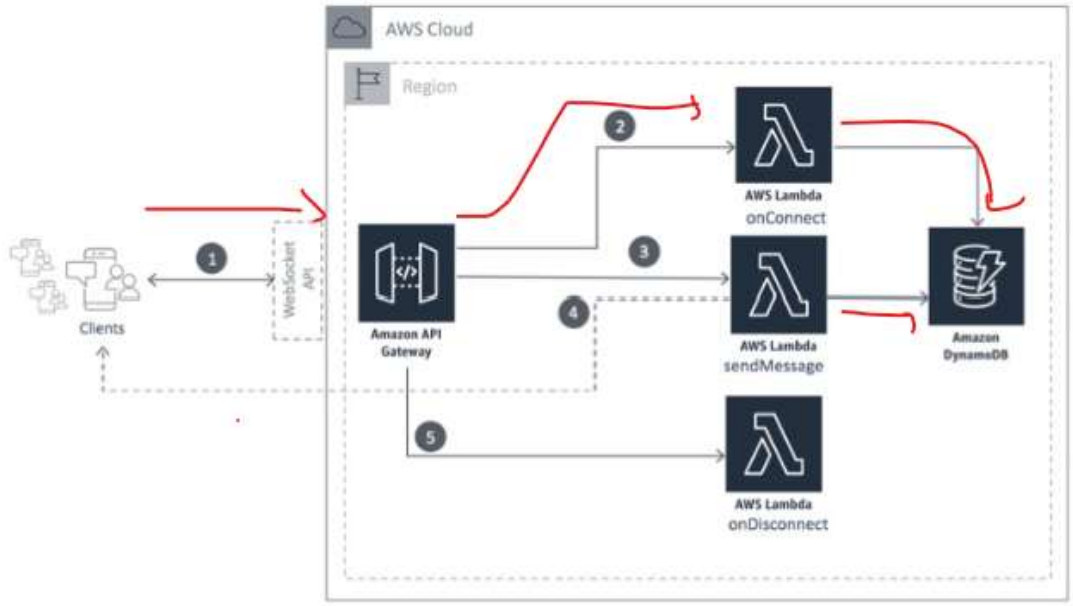
1. **Producers** (like an app or IoT device) send data records into a **Kinesis stream**.
2. The stream stores these records across **shards** for a short retention period (default 24 hours, up to 7 days).
3. **Consumers** (like AWS Lambda or your app) read and process the data in near real-time.
4. Processed data can be sent to **S3, Redshift, Elasticsearch, DynamoDB**, or **dashboards**.

- **Kinesis Data Firehose** → for automatic delivery to S3, Redshift, or OpenSearch.
- **Kinesis Data Analytics** → for SQL-based real-time analytics.




A producer puts data records into Amazon Kinesis data streams. For example, a web server sending log data to a Kinesis data stream is a producer. A consumer processes the data records from a stream. To put data into the stream, you must specify the name of the stream, a partition key, and the data blob to be added to the stream. The partition key is used to determine which shard in the stream the data record is added to. All the data in the shard is sent to the same worker that is processing the shard.

A **WebSocket API** in API Gateway is a collection of WebSocket routes that are integrated with backend HTTP endpoints, Lambda functions, or other AWS services. You can use API Gateway features to help you with all aspects of the API lifecycle, from creation through monitoring your production APIs.




API Gateway WebSocket APIs are bidirectional. A client can send messages to a specific service, and services can independently send messages to clients. This bidirectional behavior enables richer client/service interactions because services can push data to clients without requiring clients to make an explicit request. WebSocket APIs are often used in real-time applications such as chat applications, collaboration platforms, multiplayer games, and financial trading platforms.

You can use the `@connections` API from your backend service to send a callback message to a connected client, get connection information or disconnect from the client.

The option that says: **Fetch the messages from the on-premises Apache Kafka cluster by using a fleet of EC2 instances in an Auto Scaling Group. Send the data into an Amazon Kinesis Data Stream by using Amazon Kinesis Consumer Library** is incorrect because you should use Amazon Kinesis Producer Library, not Consumer Library.

The option that says: **Write a Lambda function to process the Amazon Kinesis data stream and write a GraphQL API in AWS AppSync to invoke the function. Send the callback messages to connected clients by using the `@connections` command for the API** is incorrect because using `@connections` to have the backend service connect back to the clients is not a feature of the GraphQL API when using AWS AppSync.

The option that says: **To have a consistent performance while being cost-effective, configure a Site-to-Site VPN from the on-premises data center to the AWS VPC** is incorrect because a Site-to-Site VPN does not provide a reliable and high-performance network connection between the on-premises data center and Amazon VPC.