Amazon

AWS-SOLUTION-ARCHITECT-ASSOCIATE Exam

AWS Certified Solutions Architect - Associate

Questions & Answers
Demo

Version: 16.0

Question: 1

A 3-tier e-commerce web application is current deployed on-premises and will be migrated to AWS for greater scalability and elasticity The web server currently shares read-only data using a network distributed file system The app server tier uses a clustering mechanism for discovery and shared session state that depends on IP multicast The database tier uses shared-storage clustering to provide database fall over capability, and uses several read slaves for scaling Data on all servers and the distributed file system directory is backed up weekly to off-site tapes
Which AWS storage and database architecture meets the requirements of the application?

A. Web servers: store read-only data in S3, and copy from S3 to root volume at boot time. App servers: share state using a combination of DynamoDB and IP unicast. Database: use RDS with multi-AZ deployment and one or more read replicas. Backup: web servers, app servers, and database backed up weekly to Glacier using snapshots.
B. Web servers: store read-only data in an EC2 NFS server, mount to each web server at boot time. App servers: share state using a combination of DynamoDB and IP multicast. Database: use RDS with multi-AZ deployment and one or more Read Replicas. Backup: web and app servers backed up weekly via AMIs, database backed up via DB snapshots.
C. Web servers: store read-only data in S3, and copy from S3 to root volume at boot time. App servers: share state using a combination of DynamoDB and IP unicast. Database: use RDS with multi-AZ deployment and one or more Read Replicas. Backup: web and app servers backed up weekly via AMIs, database backed up via DB snapshots.
D. Web servers: store read-only data in S3, and copy from S3 to root volume at boot time. App servers: share state using a combination of DynamoDB and IP unicast. Database: use RDS with multi-AZ deployment. Backup: web and app servers backed up weekly via AMIs, database backed up via DB snapshots.

Answer: C

Explanation:
Amazon RDS Multi-AZ deployments provide enhanced availability and durability for Database (DB) Instances, making them a natural fit for production database workloads. When you provision a Multi-AZ DB Instance, Amazon RDS automatically creates a primary DB Instance and synchronously replicates the data to a standby instance in a different Availability Zone (AZ). Each AZ runs on its own physically distinct, independent infrastructure, and is engineered to be highly reliable. In case of an infrastructure failure (for example, instance hardware failure, storage failure, or network disruption), Amazon RDS performs an automatic failover to the standby, so that you can resume database operations as soon as the failover is complete. Since the endpoint for your DB Instance remains the same after a failover, your application can resume database operation without the need for manual administrative intervention.
Benefits
Enhanced Durability

Multi-AZ deployments for the MySQL, Oracle, and PostgreSQL engines utilize synchronous physical replication to keep data on the standby up-to-date with the primary. Multi-AZ deployments for the SQL Server engine use synchronous logical replication to achieve the same result, employing SQL Server-native Mirroring technology. Both approaches safeguard your data in the event of a DB Instance failure or loss of an Availability Zone.

If a storage volume on your primary fails in a Multi-AZ deployment, Amazon RDS automatically initiates a failover to the up-to-date standby. Compare this to a Single-AZ deployment: in case of a Single-AZ database failure, a user-initiated point-in-time-restore operation will be required. This operation can take several hours to complete, and any data updates that occurred after the latest restorable time (typically within the last five minutes) will not be available.

Amazon Aurora employs a highly durable, SSD-backed virtualized storage layer purpose-built for database workloads. Amazon Aurora automatically replicates your volume six ways, across three Availability Zones. Amazon Aurora storage is fault-tolerant, transparently handling the loss of up to two copies of data without affecting database write availability and up to three copies without affecting read availability. Amazon Aurora storage is also self-healing. Data blocks and disks are continuously scanned for errors and replaced automatically.

Increased Availability

You also benefit from enhanced database availability when running Multi-AZ deployments. If an Availability Zone failure or DB Instance failure occurs, your availability impact is limited to the time automatic failover takes to complete: typically under one minute for Amazon Aurora and one to two minutes for other database engines (see the RDS FAQ for details).

The availability benefits of Multi-AZ deployments also extend to planned maintenance and backups. In the case of system upgrades like OS patching or DB Instance scaling, these operations are applied first on the standby, prior to the automatic failover. As a result, your availability impact is, again, only the time required for automatic failover to complete.

Unlike Single-AZ deployments, I/O activity is not suspended on your primary during backup for Multi-AZ deployments for the MySQL, Oracle, and PostgreSQL engines, because the backup is taken from the standby. However, note that you may still experience elevated latencies for a few minutes during backups for Multi-AZ deployments.

On instance failure in Amazon Aurora deployments, Amazon RDS uses RDS Multi-AZ technology to automate failover to one of up to 15 Amazon Aurora Replicas you have created in any of three Availability Zones. If no Amazon Aurora Replicas have been provisioned, in the case of a failure, Amazon RDS will attempt to create a new Amazon Aurora DB instance for you automatically.

No Administrative Intervention

DB Instance failover is fully automatic and requires no administrative intervention. Amazon RDS monitors the health of your primary and standbys, and initiates a failover automatically in response to a variety of failure conditions.

Failover conditions

Amazon RDS detects and automatically recovers from the most common failure scenarios for Multi-AZ deployments so that you can resume database operations as quickly as possible without administrative intervention. Amazon RDS automatically performs a failover in the event of any of the following:

Loss of availability in primary Availability Zone

Loss of network connectivity to primary

Compute unit failure on primary

Storage failure on primary

Note: When operations such as DB Instance scaling or system upgrades like OS patching are initiated for Multi-AZ deployments, for enhanced availability, they are applied first on the standby prior to an

automatic failover. As a result, your availability impact is limited only to the time required for automatic failover to complete. Note that Amazon RDS Multi-AZ deployments do not failover automatically in response to database operations such as long running queries, deadlocks or database corruption errors.

Question: 2

Your customer wishes to deploy an enterprise application to AWS which will consist of several web servers, several application servers and a small (50GB) Oracle database information is stored, both in the database and the file systems of the various servers. The backup system must support database recovery whole server and whole disk restores, and individual file restores with a recovery time of no more than two hours. They have chosen to use RDS Oracle as the database
Which backup architecture will meet these requirements?

A. Backup RDS using automated daily DB backups Backup the EC2 instances using AMIs and supplement with file-level backup to S3 using traditional enterprise backup software to provide file level restore
B. Backup RDS using a Multi-AZ Deployment Backup the EC2 instances using Amis, and supplement by copying file system data to S3 to provide file level restore.
C. Backup RDS using automated daily DB backups Backup the EC2 instances using EBS snapshots and supplement with file-level backups to Amazon Glacier using traditional enterprise backup software to provide file level restore
D. Backup RDS database to S3 using Oracle RMAN Backup the EC2 instances using Amis, and supplement with EBS snapshots for individual volume restore.

Answer: A

Explanation:
Point-In-Time Recovery
In addition to the daily automated backup, Amazon RDS archives database change logs. This enables you to recover your database to any point in time during the backup retention period, up to the last five minutes of database usage.
Amazon RDS stores multiple copies of your data, but for Single-AZ DB instances these copies are stored in a single availability zone. If for any reason a Single-AZ DB instance becomes unusable, you can use point-in-time recovery to launch a new DB instance with the latest restorable data. For more information on working with point-in-time recovery, go to Restoring a DB Instance to a Specified Time.
Note
Multi-AZ deployments store copies of your data in different Availability Zones for greater levels of data durability. For more information on Multi-AZ deployments, see High Availability (Multi-AZ).

Question: 3

Your company has HQ in Tokyo and branch offices all over the world and is using a logistics software with a multi-regional deployment on AWS in Japan, Europe and USA, The logistic software has a 3-tier architecture and currently uses MySQL 5.6 for data persistence. Each region has deployed its own database

In the HQ region you run an hourly batch process reading data from every region to compute cross-regional reports that are sent by email to all offices this batch process must be completed as fast as possible to quickly optimize logistics how do you build the database architecture in order to meet the requirements'?

A. For each regional deployment, use RDS MySQL with a master in the region and a read replica in the HQ region
B. For each regional deployment, use MySQL on EC2 with a master in the region and send hourly EBS snapshots to the HQ region
C. For each regional deployment, use RDS MySQL with a master in the region and send hourly RDS snapshots to the HQ region
D. For each regional deployment, use MySQL on EC2 with a master in the region and use S3 to copy data files hourly to the HQ region
E. Use Direct Connect to connect all regional MySQL deployments to the HQ region and reduce network latency for the batch process

Answer: A

Question: 4

A customer has a 10 GB AWS Direct Connect connection to an AWS region where they have a web application hosted on Amazon Elastic Computer Cloud (EC2). The application has dependencies on an on-premises mainframe database that uses a BASE (Basic Available. Sort stale Eventual consistency) rather than an ACID (Atomicity. Consistency isolation. Durability) consistency model. The application is exhibiting undesirable behavior because the database is not able to handle the volume of writes. How can you reduce the load on your on-premises database resources in the most cost-effective way?

A. Use an Amazon Elastic Map Reduce (EMR) S3DistCp as a synchronization mechanism between the on-premises database and a Hadoop cluster on AWS.
B. Modify the application to write to an Amazon SQS queue and develop a worker process to flush the queue to the on-premises database.
C. Modify the application to use DynamoDB to feed an EMR cluster which uses a map function to write to the on-premises database.
D. Provision an RDS read-replica database on AWS to handle the writes and synchronize the two databases using Data Pipeline.

Answer: A

Explanation:
Reference:
https://aws.amazon.com/blogs/aws/category/amazon-elastic-map-reduce/

Question: 5

Company B is launching a new game app for mobile devices. Users will log into the game using their existing social media account to streamline data capture. Company B would like to directly save

player data and scoring information from the mobile app to a DynamoDS table named Score Data When a user saves their game the progress data will be stored to the Game state S3 bucket. What is the best approach for storing data to DynamoDB and S3?

A. Use an EC2 Instance that is launched with an EC2 role providing access to the Score Data DynamoDB table and the GameState S3 bucket that communicates with the mobile app via web services.
B. Use temporary security credentials that assume a role providing access to the Score Data DynamoDB table and the Game State S3 bucket using web identity federation.
C. Use Login with Amazon allowing users to sign in with an Amazon account providing the mobile app with access to the Score Data DynamoDB table and the Game State S3 bucket.
D. Use an IAM user with access credentials assigned a role providing access to the Score Data DynamoDB table and the Game State S3 bucket for distribution with the mobile app.

Answer: B

Explanation:
Web Identity Federation
Imagine that you are creating a mobile app that accesses AWS resources, such as a game that runs on a mobile device and stores player and score information using Amazon S3 and DynamoDB.
When you write such an app, you'll make requests to AWS services that must be signed with an AWS access key. However, we strongly recommend that you do not embed or distribute long-term AWS credentials with apps that a user downloads to a device, even in an encrypted store. Instead, build your app so that it requests temporary AWS security credentials dynamically when needed using web identity federation. The supplied temporary credentials map to an AWS role that has only the permissions needed to perform the tasks required by the mobile app.
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.
For most scenarios, we recommend that you use Amazon Cognito because it acts as an identity broker and does much of the federation work for you. For details, see the following section, Using Amazon Cognito for Mobile Apps.
If you don't use Amazon Cognito, then you must write code that interacts with a web IdP (Login with Amazon, Facebook, Google, or any other OIDC-compatible IdP) and then calls the AssumeRoleWithWebIdentity API to trade the authentication token you get from those IdPs for AWS temporary security credentials. If you have already used this approach for existing apps, you can continue to use it.
Using Amazon Cognito for Mobile Apps
The preferred way to use web identity federation is to use Amazon Cognito. For example, Adele the developer is building a game for a mobile device where user data such as scores and profiles is stored in Amazon S3 and Amazon DynamoDB. Adele could also store this data locally on the device and use Amazon Cognito to keep it synchronized across devices. She knows that for security and maintenance reasons, long-term AWS security credentials should not be distributed with the game. She also knows that the game might have a large number of users. For all of these reasons, she does not want

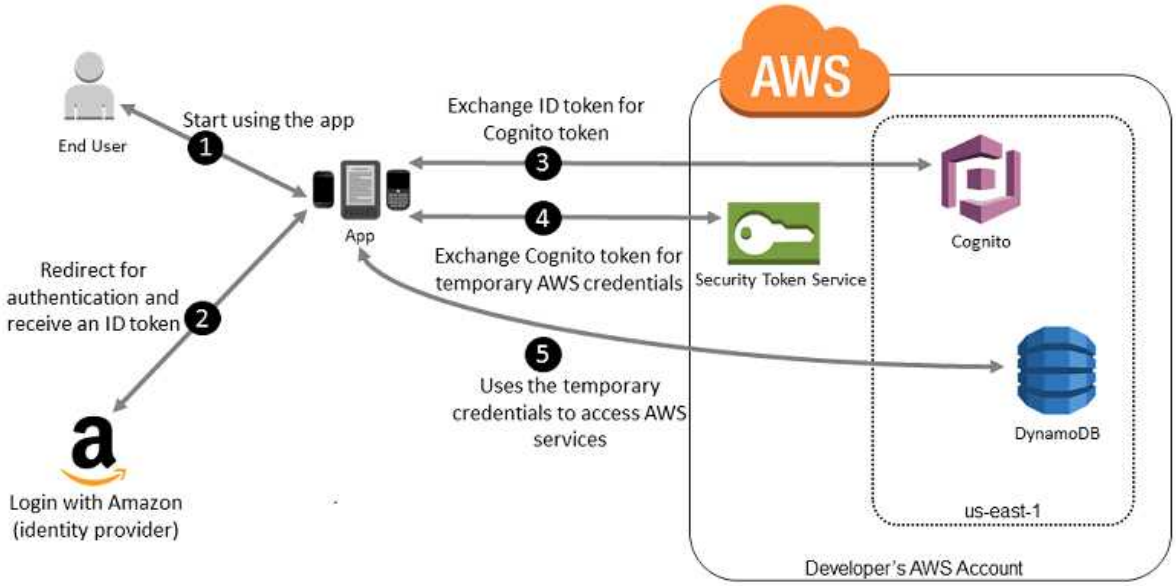
to create new user identities in IAM for each player. Instead, she builds the game so that users can sign in using an identity that they've already established with a well-known identity provider, such as Login with Amazon, Facebook, Google, or any OpenID Connect (OIDC)-compatible identity provider. Her game can take advantage of the authentication mechanism from one of these providers to validate the user's identity.

To enable the mobile app to access her AWS resources, Adele first registers for a developer ID with her chosen IdPs. She also configures the application with each of these providers. In her AWS account that contains the Amazon S3 bucket and DynamoDB table for the game, Adele uses Amazon Cognito to create IAM roles that precisely define permissions that the game needs. If she is using an OIDC IdP, she also creates an IAM OIDC identity provider entity to establish trust between her AWS account and the IdP.

In the app's code, Adele calls the sign-in interface for the IdP that she configured previously. The IdP handles all the details of letting the user sign in, and the app gets an OAuth access token or OIDC ID token from the provider. Adele's app can trade this authentication information for a set of temporary security credentials that consist of an AWS access key ID, a secret access key, and a session token. The app can then use these credentials to access web services offered by AWS. The app is limited to the permissions that are defined in the role that it assumes.

The following figure shows a simplified flow for how this might work, using Login with Amazon as the IdP. For Step 2, the app can also use Facebook, Google, or any OIDC-compatible identity provider, but that's not shown here.

Sample workflow using Amazon Cognito to federate users for a mobile application



A customer starts your app on a mobile device. The app asks the user to sign in.

The app uses Login with Amazon resources to accept the user's credentials.

The app uses Cognito APIs to exchange the Login with Amazon ID token for a Cognito token.

The app requests temporary security credentials from AWS STS, passing the Cognito token.

The temporary security credentials can be used by the app to access any AWS resources required by the app to operate. The role associated with the temporary security credentials and its assigned policies determines what can be accessed.

Use the following process to configure your app to use Amazon Cognito to authenticate users and give your app access to AWS resources. For specific steps to accomplish this scenario, consult the documentation for Amazon Cognito.

(Optional) Sign up as a developer with Login with Amazon, Facebook, Google, or any other OpenID Connect (OIDC)–compatible identity provider and configure one or more apps with the provider. This step is optional because Amazon Cognito also supports unauthenticated (guest) access for your users.
Go to Amazon Cognito in the AWS Management Console. Use the Amazon Cognito wizard to create an identity pool, which is a container that Amazon Cognito uses to keep end user identities organized for your apps. You can share identity pools between apps. When you set up an identity pool, Amazon Cognito creates one or two IAM roles (one for authenticated identities, and one for unauthenticated "guest" identities) that define permissions for Amazon Cognito users.
Download and integrate the AWS SDK for iOS or the AWS SDK for Android with your app, and import the files required to use Amazon Cognito.
Create an instance of the Amazon Cognito credentials provider, passing the identity pool ID, your AWS account number, and the Amazon Resource Name (ARN) of the roles that you associated with the identity pool. The Amazon Cognito wizard in the AWS Management Console provides sample code to help you get started.
When your app accesses an AWS resource, pass the credentials provider instance to the client object, which passes temporary security credentials to the client. The permissions for the credentials are based on the role or roles that you defined earlier.

Question: 6

Your company plans to host a large donation website on Amazon Web Services (AWS). You anticipate a large and undetermined amount of traffic that will create many database writes. To be certain that you do not drop any writes to a database hosted on AWS. Which service should you use?

A. Amazon RDS with provisioned IOPS up to the anticipated peak write throughput.
B. Amazon Simple Queue Service (SOS) for capturing the writes and draining the queue to write to the database.
C. Amazon ElastiCache to store the writes until the writes are committed to the database.
D. Amazon DynamoDB with provisioned write throughput up to the anticipated peak write throughput.

Answer: B

Explanation:
Amazon Simple Queue Service (Amazon SQS) offers a reliable, highly scalable hosted queue for storing messages as they travel between computers. By using Amazon SQS, developers can simply move data between distributed application components performing different tasks, without losing messages or requiring each component to be always available. Amazon SQS makes it easy to build a distributed, decoupled application, working in close conjunction with the Amazon Elastic Compute Cloud (Amazon EC2) and the other AWS infrastructure web services.
What can I do with Amazon SQS?
Amazon SQS is a web service that gives you access to a message queue that can be used to store messages while waiting for a computer to process them. This allows you to quickly build message queuing applications that can be run on any computer on the internet. Since Amazon SQS is highly scalable and you only pay for what you use, you can start small and grow your application as you wish, with no compromise on performance or reliability. This lets you focus on building sophisticated message-based applications, without worrying about how the messages are stored and managed.

You can use Amazon SQS with software applications in various ways. For example, you can:
Integrate Amazon SQS with other AWS infrastructure web services to make applications more reliable and flexible.
Use Amazon SQS to create a queue of work where each message is a task that needs to be completed by a process. One or many computers can read tasks from the queue and perform them.
Build a microservices architecture, using queues to connect your microservices.
Keep notifications of significant events in a business process in an Amazon SQS queue. Each event can have a corresponding message in a queue, and applications that need to be aware of the event can read and process the messages.

Question: 7

You have launched an EC2 instance with four (4) 500 GB EBS Provisioned IOPS volumes attached The EC2 Instance Is EBS-Optimized and supports 500 Mbps throughput between EC2 and EBS The two EBS volumes are configured as a single RAID o device, and each Provisioned IOPS volume is provisioned with 4.000 IOPS (4 000 16KB reads or writes) for a total of 16.000 random IOPS on the instance The EC2 Instance initially delivers the expected 16 000 IOPS random read and write performance Sometime later in order to increase the total random I/O performance of the instance, you add an additional two 500 GB EBS Provisioned IOPS volumes to the RAID Each volume Is provisioned to 4.000 IOPs like the original four for a total of 24.000 IOPS on the EC2 instance Monitoring shows that the EC2 instance CPU utilization increased from 50% to 70%. but the total random IOPS measured at the instance level does not increase at all.
What is the problem and a valid solution?

A. Larger storage volumes support higher Provisioned IOPS rates: increase the provisioned volume storage of each of the 6 EBS volumes to 1TB
B. The EBS-Optimized throughput limits the total IOPS that can be utilized use an EBS-Optimized instance that provides larger throughput.
C. Small block sizes cause performance degradation, limiting the I'O throughput, configure the instance device driver and file system to use 64KB blocks to increase throughput.
D. RAID 0 only scales linearly to about 4 devices, use RAID 0 with 4 EBS Provisioned IOPS volumes but increase each Provisioned IOPS EBS volume to 6.000 IOPS.
E. The standard EBS instance root volume limits the total IOPS rate, change the instant root volume to also be a 500GB 4.000 Provisioned IOPS volume.

Answer: E

Question: 8

You have recently joined a startup company building sensors to measure street noise and air quality in urban areas. The company has been running a pilot deployment of around 100 sensors for 3 months each sensor uploads 1KB of sensor data every minute to a backend hosted on AWS.
During the pilot, you measured a peak or 10 IOPS on the database, and you stored an average of 3GB of sensor data per month in the database.
The current deployment consists of a load-balanced auto scaled Ingestion layer using EC2 instances and a PostgreSQL RDS database with 500GB standard storage.
The pilot is considered a success and your CEO has managed to get the attention or some potential

investors. The business plan requires a deployment of at least 100K sensors which needs to be supported by the backend. You also need to store sensor data for at least two years to be able to compare year over year Improvements.
To secure funding, you have to make sure that the platform meets these requirements and leaves room for further scaling. Which setup win meet the requirements?

A. Add an SQS queue to the ingestion layer to buffer writes to the RDS instance
B. Ingest data into a DynamoDB table and move old data to a Redshift cluster
C. Replace the RDS instance with a 6 node Redshift cluster with 96TB of storage
D. Keep the current architecture but upgrade RDS storage to 3TB and 10K provisioned IOPS

Answer: C

Question: 9

Your company is in the process of developing a next generation pet collar that collects biometric information to assist families with promoting healthy lifestyles for their pets Each collar will push 30kb of biometric data In JSON format every 2 seconds to a collection platform that will process and analyze the data providing health trending information back to the pet owners and veterinarians via a web portal Management has tasked you to architect the collection platform ensuring the following requirements are met.
Provide the ability for real-time analytics of the inbound biometric data
Ensure processing of the biometric data is highly durable. Elastic and parallel
The results of the analytic processing should be persisted for data mining
Which architecture outlined below win meet the initial requirements for the collection platform?

A. Utilize S3 to collect the inbound sensor data analyze the data from S3 with a daily scheduled Data Pipeline and save the results to a Redshift Cluster.
B. Utilize Amazon Kinesis to collect the inbound sensor data, analyze the data with Kinesis clients and save the results to a Redshift cluster using EMR.
C. Utilize SQS to collect the inbound sensor data analyze the data from SQS with Amazon Kinesis and save the results to a Microsoft SQL Server RDS instance.
D. Utilize EMR to collect the inbound sensor data, analyze the data from EUR with Amazon Kinesis and save me results to DynamoDB.

Answer: B

Question: 10

You need a persistent and durable storage to trace call activity of an IVR (Interactive Voice Response) system. Call duration is mostly in the 2-3 minutes timeframe. Each traced call can be either active or terminated. An external application needs to know each minute the list of currently active calls, which are usually a few calls/second. Put once per month there is a periodic peak up to 1000 calls/second for a few hours. The system is open 24/7 and any downtime should be avoided. Historical data is periodically archived to files. Cost saving is a priority for this project.
What database implementation would better fit this scenario, keeping costs as low as possible?

A. Use RDS Multi-AZ with two tables, one for -Active calls" and one for -Terminated calls". In this way the "Active calls_ table is always small and effective to access.
B. Use DynamoDB with a "Calls" table and a Global Secondary Index on a "IsActive'" attribute that is present for active calls only In this way the Global Secondary index is sparse and more effective.
C. Use DynamoDB with a 'Calls" table and a Global secondary index on a 'State" attribute that can equal to "active" or "terminated" in this way the Global Secondary index can be used for all Items in the table.
D. Use RDS Multi-AZ with a "CALLS" table and an Indexed "STATE* field that can be equal to 'ACTIVE" or -TERMINATED" In this way the SOL query Is optimized by the use of the Index.

Answer: A

Question: 11

A web design company currently runs several FTP servers that their 250 customers use to upload and download large graphic files They wish to move this system to AWS to make it more scalable, but they wish to maintain customer privacy and Keep costs to a minimum.
What AWS architecture would you recommend?

A. ASK their customers to use an S3 client instead of an FTP client. Create a single S3 bucket Create an IAM user for each customer Put the IAM Users in a Group that has an IAM policy that permits access to sub-directories within the bucket via use of the 'username' Policy variable.
B. Create a single S3 bucket with Reduced Redundancy Storage turned on and ask their customers to use an S3 client instead of an FTP client Create a bucket for each customer with a Bucket Policy that permits access only to that one customer.
C. Create an auto-scaling group of FTP servers with a scaling policy to automatically scale-in when minimum network traffic on the auto-scaling group is below a given threshold. Load a central list of ftp users from S3 as part of the user Data startup script on each Instance.
D. Create a single S3 bucket with Requester Pays turned on and ask their customers to use an S3 client instead of an FTP client Create a bucket tor each customer with a Bucket Policy that permits access only to that one customer.

Answer: A

Question: 12

You have been asked to design the storage layer for an application. The application requires disk performance of at least 100,000 IOPS in addition, the storage layer must be able to survive the loss of an individual disk. EC2 instance, or Availability Zone without any data loss. The volume you provide must have a capacity of at least 3 TB. Which of the following designs will meet these objectives'?

A. Instantiate a c3.8xlarge instance in us-east-1. Provision 4x1TB EBS volumes, attach them to the instance, and configure them as a single RAID 5 volume. Ensure that EBS snapshots are performed every 15 minutes.
B. Instantiate a c3.8xlarge instance in us-east-1. Provision 3xlTB EBS volumes, attach them to the Instance, and configure them as a single RAID 0 volume. Ensure that EBS snapshots are performed every 15 minutes.

C. Instantiate an i2.8xlarge instance in us-east-1a. Create a RAID 0 volume using the four 800GB SSD ephemeral disks provided with the instance. Provision 3x1TB EBS volumes, attach them to the instance, and configure them as a second RAID 0 volume. Configure synchronous, block-level replication from the ephemeral-backed volume to the EBS-backed volume.
D. Instantiate a c3.8xlarge instance in us-east-1. Provision an AWS Storage Gateway and configure it for 3 TB of storage and 100,000 IOPS. Attach the volume to the instance. E. Instantiate an i2.8xlarge instance in us-east-1a. Create a RAID 0 volume using the four 800GB SSD ephemeral disks provided with the instance. Configure synchronous, block- level replication to an identically configured instance in us-east-1b.

Answer: C

Question: 13

You would like to create a mirror image of your production environment in another region for disaster recovery purposes. Which of the following AWS resources do not need to be recreated in the second region? (Choose 2 answers)

A. Route 53 Record Sets
B. IM1 Roles
C. Elastic IP Addresses (EIP)
D. EC2 Key Pairs
E. Launch configurations
F. Security Groups

Answer: A, C

Explanation:
Reference:
http://ltech.com/wp-content/themes/optimize/download/AWS_Disaster_Recovery.pdf (page 6)

Question: 14

Your company runs a customer facing event registration site This site is built with a 3-tier architecture with web and application tier servers and a MySQL database The application requires 6 web tier servers and 6 application tier servers for normal operation, but can run on a minimum of 65% server capacity and a single MySQL database. When deploying this application in a region with three availability zones (AZs) which architecture provides high availability?

A. A web tier deployed across 2 AZs with 3 EC2 (Elastic Compute Cloud) instances in each AZ inside an Auto Scaling Group behind an ELB (elastic load balancer), and an application tier deployed across 2 AZs with 3 EC2 instances in each AZ inside an Auto Scaling Group behind an ELB, and one RDS (Relational Database Service) instance deployed with read replicas in the other AZ.
B. A web tier deployed across 3 AZs with 2 EC2 (Elastic Compute Cloud) instances in each AZ inside an Auto Scaling Group behind an ELB (elastic load balancer) and an application tier deployed across 3 AZs with 2 EC2 instances in each AZ inside an Auto Scaling Group behind an ELB and one RDS (Relational Database Service) Instance deployed with read replicas in the two other AZs.

C. A web tier deployed across 2 AZs with 3 EC2 (Elastic Compute Cloud) instances in each AZ inside an Auto Scaling Group behind an ELB (elastic load balancer) and an application tier deployed across 2 AZs with 3 EC2 instances m each AZ inside an Auto Scaling Group behind an ELS and a Multi-AZ RDS (Relational Database Service) deployment.
D. A web tier deployed across 3 AZs with 2 EC2 (Elastic Compute Cloud) instances in each AZ Inside an Auto Scaling Group behind an ELB (elastic load balancer). And an application tier deployed across 3 AZs with 2 EC2 instances in each AZ inside an Auto Scaling Group behind an ELB. And a Multi-AZ RDS (Relational Database services) deployment.

Answer: D

Explanation:
Amazon RDS Multi-AZ Deployments
Amazon RDS Multi-AZ deployments provide enhanced availability and durability for Database (DB) Instances, making them a natural fit for production database workloads. When you provision a Multi-AZ DB Instance, Amazon RDS automatically creates a primary DB Instance and synchronously replicates the data to a standby instance in a different Availability Zone (AZ). Each AZ runs on its own physically distinct, independent infrastructure, and is engineered to be highly reliable. In case of an infrastructure failure (for example, instance hardware failure, storage failure, or network disruption), Amazon RDS performs an automatic failover to the standby, so that you can resume database operations as soon as the failover is complete. Since the endpoint for your DB Instance remains the same after a failover, your application can resume database operation without the need for manual administrative intervention.
Enhanced Durability
Multi-AZ deployments for the MySQL, Oracle, and PostgreSQL engines utilize synchronous physical replication to keep data on the standby up-to-date with the primary. Multi-AZ deployments for the SQL Server engine use synchronous logical replication to achieve the same result, employing SQL Server-native Mirroring technology. Both approaches safeguard your data in the event of a DB Instance failure or loss of an Availability Zone.
If a storage volume on your primary fails in a Multi-AZ deployment, Amazon RDS automatically initiates a failover to the up-to-date standby. Compare this to a Single-AZ deployment: in case of a Single-AZ database failure, a user-initiated point-in-time-restore operation will be required. This operation can take several hours to complete, and any data updates that occurred after the latest restorable time (typically within the last five minutes) will not be available.
Amazon Aurora employs a highly durable, SSD-backed virtualized storage layer purpose-built for database workloads. Amazon Aurora automatically replicates your volume six ways, across three Availability Zones. Amazon Aurora storage is fault-tolerant, transparently handling the loss of up to two copies of data without affecting database write availability and up to three copies without affecting read availability. Amazon Aurora storage is also self-healing. Data blocks and disks are continuously scanned for errors and replaced automatically.
Increased Availability
You also benefit from enhanced database availability when running Multi-AZ deployments. If an Availability Zone failure or DB Instance failure occurs, your availability impact is limited to the time automatic failover takes to complete: typically under one minute for Amazon Aurora and one to two minutes for other database engines (see the RDS FAQ for details).
The availability benefits of Multi-AZ deployments also extend to planned maintenance and backups. In the case of system upgrades like OS patching or DB Instance scaling, these operations are applied first on the standby, prior to the automatic failover. As a result, your availability impact is, again, only

the time required for automatic failover to complete.
Unlike Single-AZ deployments, I/O activity is not suspended on your primary during backup for Multi-AZ deployments for the MySQL, Oracle, and PostgreSQL engines, because the backup is taken from the standby. However, note that you may still experience elevated latencies for a few minutes during backups for Multi-AZ deployments.
On instance failure in Amazon Aurora deployments, Amazon RDS uses RDS Multi-AZ technology to automate failover to one of up to 15 Amazon Aurora Replicas you have created in any of three Availability Zones. If no Amazon Aurora Replicas have been provisioned, in the case of a failure, Amazon RDS will attempt to create a new Amazon Aurora DB instance for you automatically.

Question: 15

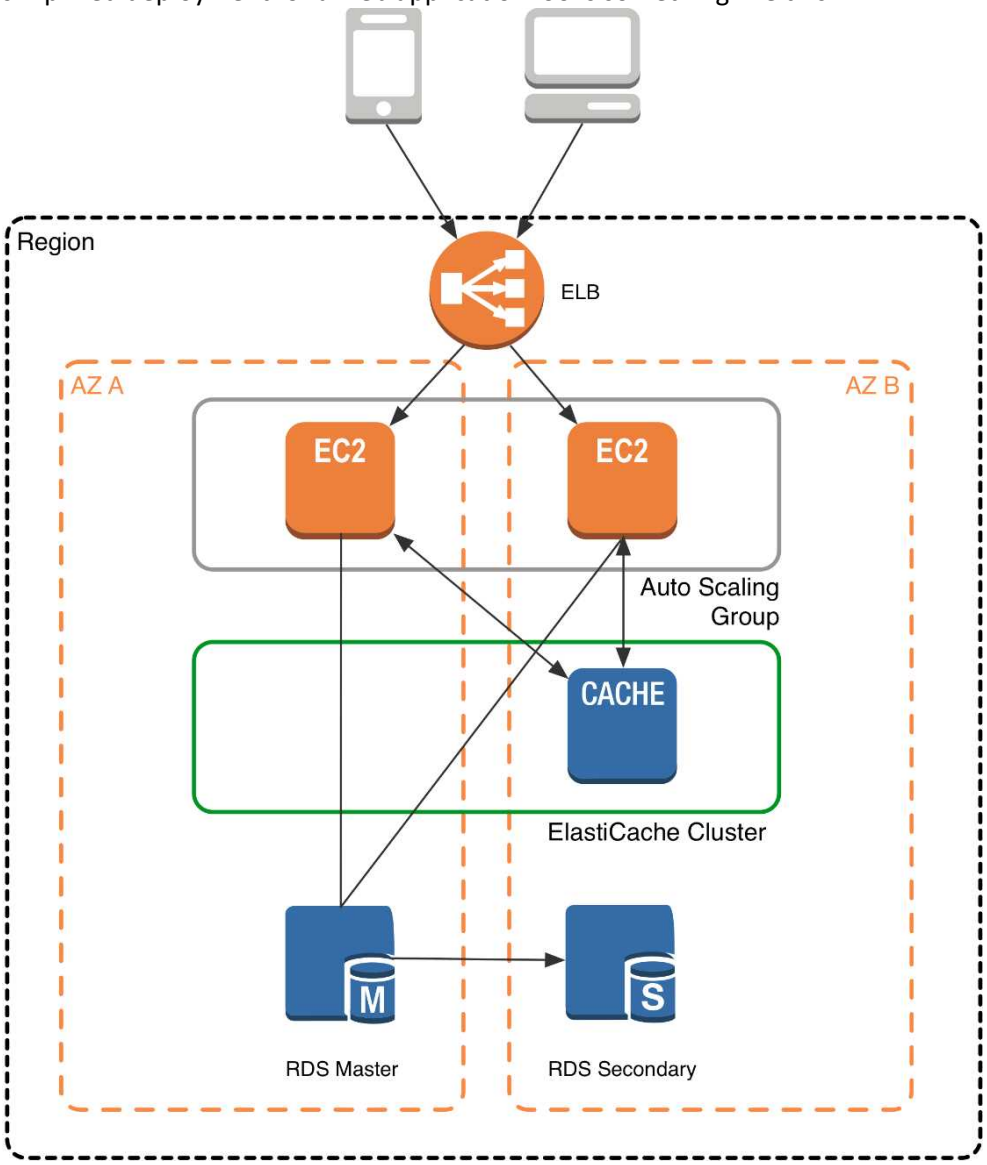
Your application is using an ELB in front of an Auto Scaling group of web/application servers deployed across two AZs and a Multi-AZ RDS Instance for data persistence.
The database CPU is often above 80% usage and 90% of I/O operations on the database are reads. To improve performance you recently added a single-node Memcached ElastiCache Cluster to cache frequent DB query results. In the next weeks the overall workload is expected to grow by 30%.
Do you need to change anything in the architecture to maintain the high availability or the application with the anticipated additional load? Why?

A. Yes, you should deploy two Memcached ElastiCache Clusters in different AZs because the RDS instance will not be able to handle the load if the cache node fails.
B. No, if the cache node fails you can always get the same data from the DB withouthaving any availability impact.
C. No, if the cache node fails the automated ElastiCache node recovery feature will prevent any availability impact.
D. Yes, you should deploy the Memcached ElastiCache Cluster with two nodes in the same AZ as the RDS DB master instance to handle the load if one cache node fails.

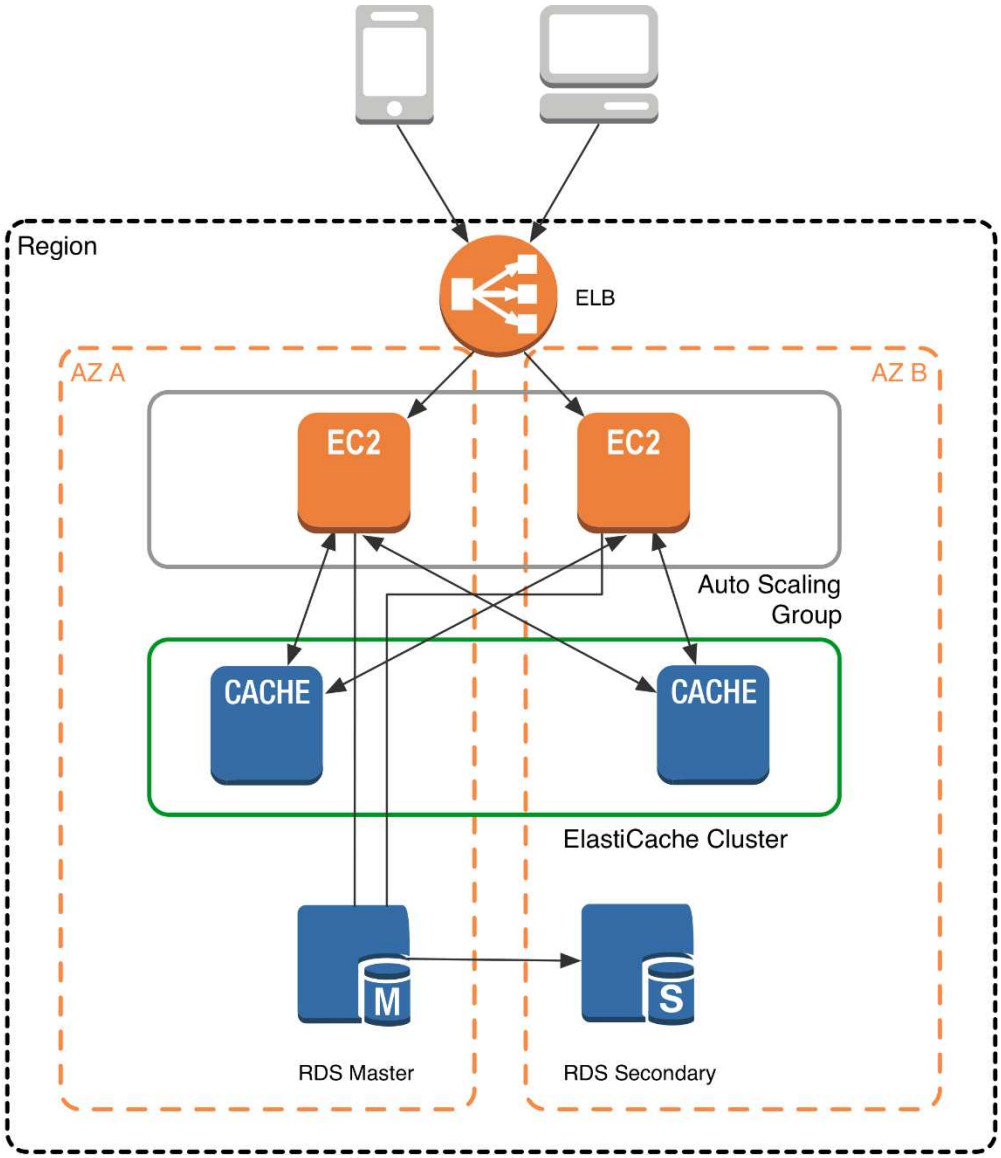
Answer: A

Explanation:
ElastiCache for Memcached
The primary goal of caching is typically to offload reads from your database or other primary data source. In most apps, you have hot spots of data that are regularly queried, but only updated periodically. Think of the front page of a blog or news site, or the top 100 leaderboard in an online game. In this type of case, your app can receive dozens, hundreds, or even thousands of requests for the same data before it's updated again. Having your caching layer handle these queries has several advantages. First, it's considerably cheaper to add an in-memory cache than to scale up to a larger database cluster. Second, an in-memory cache is also easier to scale out, because it's easier to distribute an in-memory cache horizontally than a relational database.
Last, a caching layer provides a request buffer in the event of a sudden spike in usage. If your app or game ends up on the front page of Reddit or the App Store, it's not unheard of to see a spike that is 10 to 100 times your normal application load. Even if you autoscale your application instances, a 10x request spike will likely make your database very unhappy.
Let's focus on ElastiCache for Memcached first, because it is the best fit for a cachingfocused solution. We'll revisit Redis later in the paper, and weigh its advantages and disadvantages.
Architecture with ElastiCache for Memcached

When you deploy an ElastiCache Memcached cluster, it sits in your application as a separate tier alongside your database. As mentioned previously, Amazon ElastiCache does not directly communicate with your database tier, or indeed have any particular knowledge of your database. A simplified deployment for a web application looks something like this:

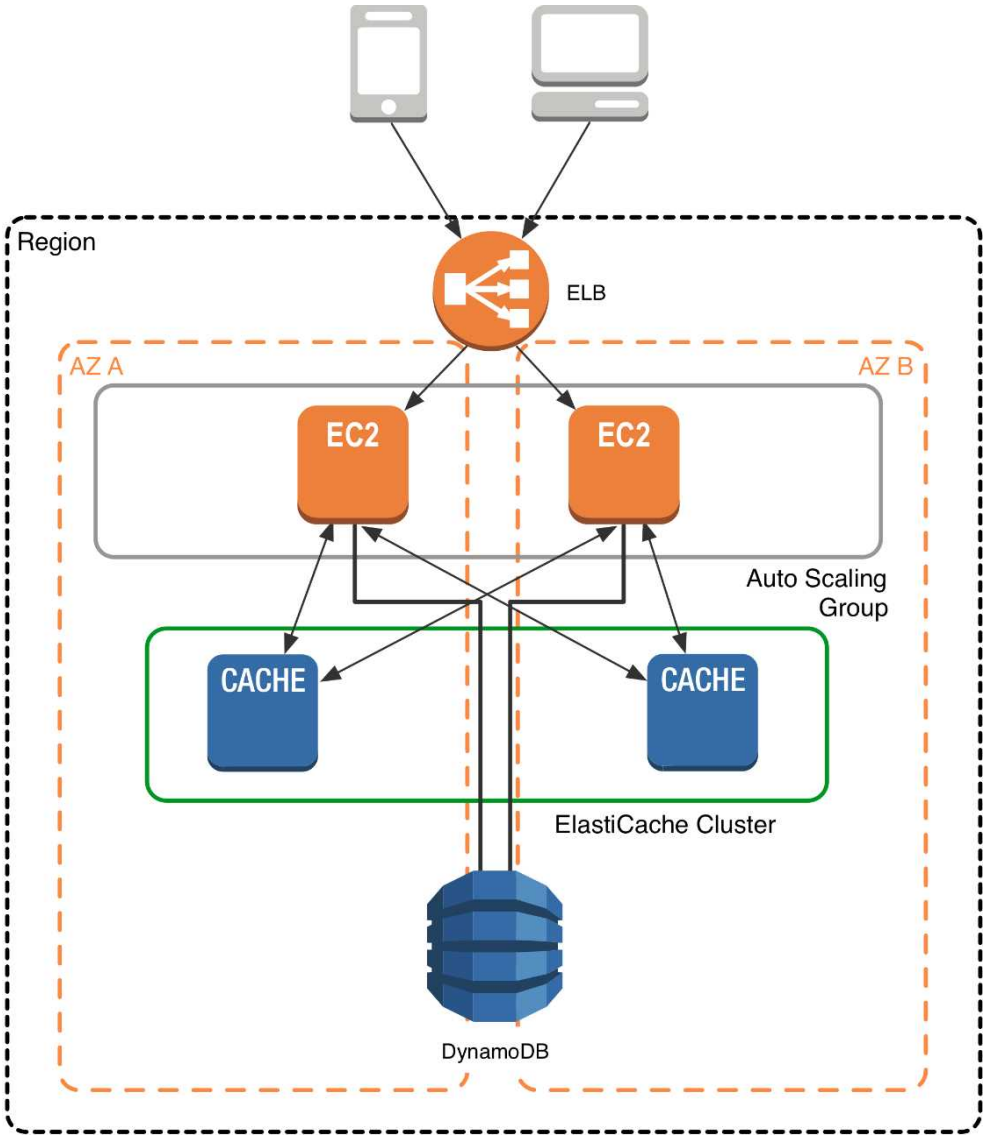


In this architecture diagram, the Amazon EC2 application instances are in an Auto Scaling group, located behind a load balancer using Elastic Load Balancing, which distributes requests among the instances. As requests come into a given EC2 instance, that EC2 instance is responsible for communicating with ElastiCache and the database tier. For development purposes, you can begin with a single ElastiCache node to test your application, and then scale to additional cluster nodes by modifying the ElastiCache cluster. As you add additional cache nodes, the EC2 application instances are able to distribute cache keys across multiple ElastiCache nodes. The most common practice is to use client-side sharding to distribute keys across cache nodes, which we will discuss later in this paper.



When you launch an ElastiCache cluster, you can choose the Availability Zone(s) that the cluster lives in. For best performance, you should configure your cluster to use the same Availability Zones as your application servers. To launch an ElastiCache cluster in a specific Availability Zone, make sure to specify the Preferred Zone(s) option during cache cluster creation. The Availability Zones that you specify will be where ElastiCache will launch your cache nodes. We recommend that you select Spread Nodes Across Zones, which tells ElastiCache to distribute cache nodes across these zones as evenly as possible. This distribution will mitigate the impact of an Availability Zone disruption on your ElastiCache nodes. The trade-off is that some of the requests from your application to ElastiCache will go to a node in a different Availability Zone, meaning latency will be slightly higher. For more details, refer to Creating a Cache Cluster in the Amazon ElastiCache User Guide.

As mentioned at the outset, ElastiCache can be coupled with a wide variety of databases. Here is an example architecture that uses Amazon DynamoDB instead of Amazon RDS and MySQL:



This combination of DynamoDB and ElastiCache is very popular with mobile and game companies, because DynamoDB allows for higher write throughput at lower cost than traditional relational databases. In addition, DynamoDB uses a key-value access pattern similar to ElastiCache, which also simplifies the programming model. Instead of using relational SQL for the primary database but then key-value patterns for the cache, both the primary database and cache can be programmed similarly. In this architecture pattern, DynamoDB remains the source of truth for data, but application reads are offloaded to ElastiCache for a speed boost.

Question: 16

You are responsible for a legacy web application whose server environment is approaching end of life You would like to migrate this application to AWS as quickly as possible, since the application environment currently has the following limitations:
The VM's single 10GB VMDK is almost full
Me virtual network interface still uses the 10Mbps driver, which leaves your 100Mbps WAN connection completely underutilized

It is currently running on a highly customized. Windows VM within a VMware environment:
You do not have me installation media
This is a mission critical application with an RTO (Recovery Time Objective) of 8 hours. RPO (Recovery Point Objective) of 1 hour. How could you best migrate this application to AWS while meeting your business continuity requirements?

A. Use the EC2 VM Import Connector for vCenter to import the VM into EC2.
B. Use Import/Export to import the VM as an ESS snapshot and attach to EC2.
C. Use S3 to create a backup of the VM and restore the data into EC2.
D. Use me ec2-bundle-instance API to Import an Image of the VM into EC2

Answer: A

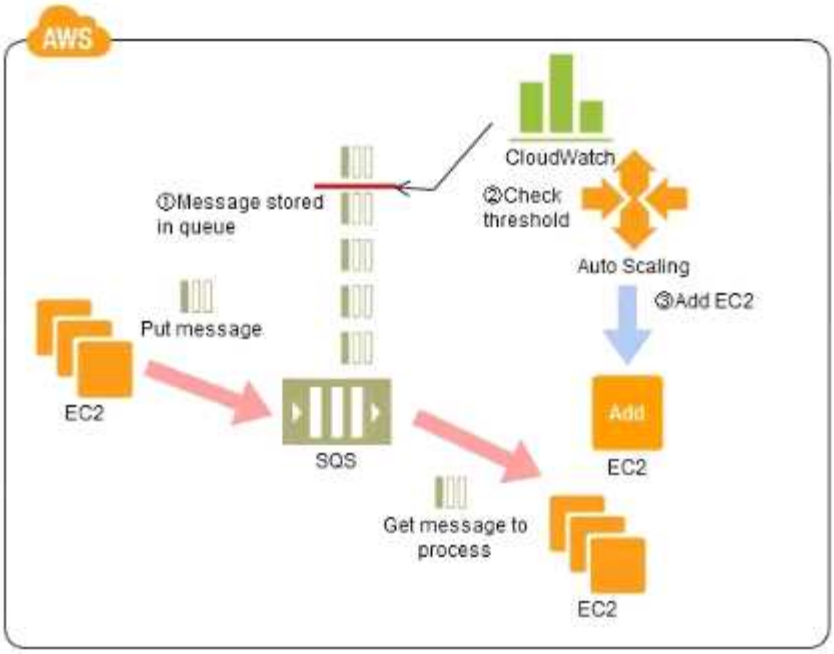
Question: 17

An International company has deployed a multi-tier web application that relies on DynamoDB in a single region For regulatory reasons they need disaster recovery capability In a separate region with a Recovery Time Objective of 2 hours and a Recovery Point Objective of 24 hours They should synchronize their data on a regular basis and be able to provision me web application rapidly using CloudFormation.
The objective is to minimize changes to the existing web application, control the throughput of DynamoDB used for the synchronization of data and synchronize only the modified elements.
Which design would you choose to meet these requirements?

A. Use AWS data Pipeline to schedule a DynamoDB cross region copy once a day. create a Lastupdated' attribute in your DynamoDB table that would represent the timestamp of the last update and use it as a filter.
B. Use EMR and write a custom script to retrieve data from DynamoDB in the current region using a SCAN operation and push it to DynamoDB in the second region.
C. Use AWS data Pipeline to schedule an export of the DynamoDB table to S3 in the current region once a day then schedule another task immediately after it that will import data from S3 to DynamoDB in the other region.
D. Send also each Ante into an SQS queue in me second region; use an auto-scaling group behind the SQS queue to replay the write in the second region.

Answer: A

Question: 18



Refer to the architecture diagram above of a batch processing solution using Simple Queue Service (SQS) to set up a message queue between EC2 instances which are used as batch processors Cloud Watch monitors the number of Job requests (queued messages) and an Auto Scaling group adds or deletes batch servers automatically based on parameters set in Cloud Watch alarms. You can use this architecture to implement which of the following features in a cost effective and efficient manner?

A. Reduce the overall lime for executing jobs through parallel processing by allowing a busy EC2 instance that receives a message to pass it to the next instance in a daisy-chain setup.
B. Implement fault tolerance against EC2 instance failure since messages would remain in SQS and worn can continue with recovery of EC2 instances implement fault tolerance against SQS failure by backing up messages to S3.
C. Implement message passing between EC2 instances within a batch by exchanging messages through SQS.
D. Coordinate number of EC2 instances with number of job requests automatically thus Improving cost effectiveness.
E. Handle high priority jobs before lower priority jobs by assigning a priority metadata field to SQS messages.

Answer: D

Explanation:
Reference:

There are cases where a large number of batch jobs may need processing, and where the the jobs may need to be re-prioritized.
For example, one such case is one where there are differences between different levels of services for unpaid users versus subscriber users (such as the time until publication) in services enabling, for example, presentation files to be uploaded for publication from a web browser. When the user uploads a presentation file, the conversion processes, for example, for publication are performed as

batch processes on the system side, and the file is published after the conversion. Is it then necessary to be able to assign the level of priority to the batch processes for each type of subscriber.

Explanation of the Cloud Solution/Pattern

A queue is used in controlling batch jobs. The queue need only be provided with priority numbers. Job requests are controlled by the queue, and the job requests in the queue are processed by a batch server. In Cloud computing, a highly reliable queue is provided as a service, which you can use to structure a highly reliable batch system with ease. You may prepare multiple queues depending on priority levels, with job requests put into the queues depending on their priority levels, to apply prioritization to batch processes. The performance (number) of batch servers corresponding to a queue must be in accordance with the priority level thereof.

Implementation

In AWS, the queue service is the Simple Queue Service (SQS). Multiple SQS queues may be prepared to prepare queues for individual priority levels (with a priority queue and a secondary queue). Moreover, you may also use the message Delayed Send function to delay process execution.

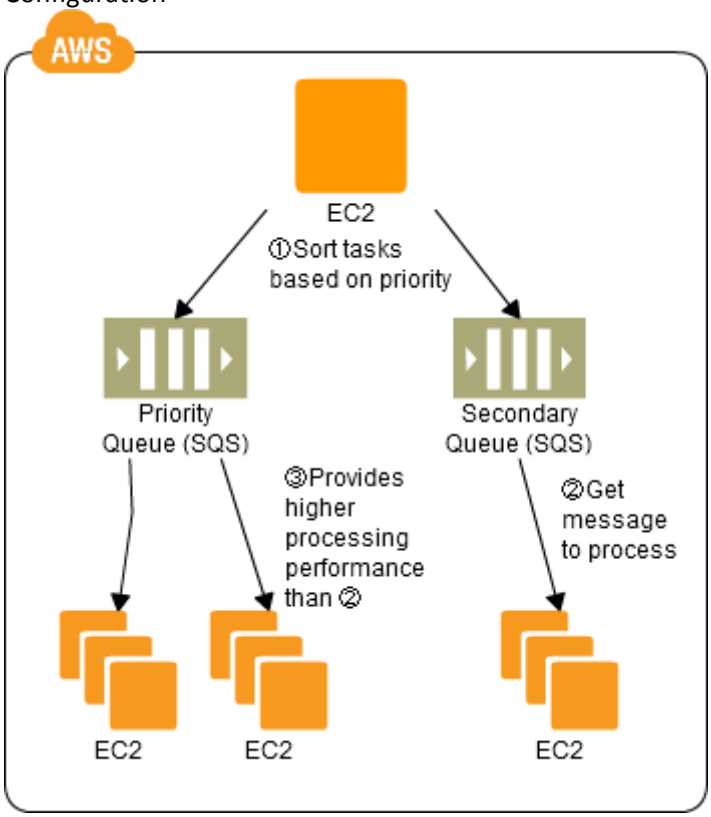
Use SQS to prepare multiple queues for the individual priority levels.

Place those processes to be executed immediately (job requests) in the high priority queue.

Prepare numbers of batch servers, for processing the job requests of the queues, depending on the priority levels.

Queues have a message "Delayed Send" function. You can use this to delay the time for starting a process.

Configuration



Benefits

You can increase or decrease the number of servers for processing jobs to change automatically the processing speeds of the priority queues and secondary queues.

You can handle performance and service requirements through merely increasing or decreasing the number of EC2 instances used in job processing.

Even if an EC2 were to fail, the messages (jobs) would remain in the queue service, enabling processing to be continued immediately upon recovery of the EC2 instance, producing a system that is robust to failure.
Cautions
Depending on the balance between the number of EC2 instances for performing the processes and the number of messages that are queued, there may be cases where processing in the secondary queue may be completed first, so you need to monitor the processing speeds in the primary queue and the secondary queue.

Question: 19

Your company currently has a 2-tier web application running in an on-premises data center. You have experienced several infrastructure failures in the past two months resulting in significant financial losses. Your CIO is strongly agreeing to move the application to AWS. While working on achieving buy-in from the other company executives, he asks you to develop a disaster recovery plan to help improve Business continuity in the short term. He specifies a target Recovery Time Objective (RTO) of 4 hours and a Recovery Point Objective (RPO) of 1 hour or less. He also asks you to implement the solution within 2 weeks. Your database is 200GB in size and you have a 20Mbps Internet connection. How would you do this while minimizing costs?

A. Create an EBS backed private AMI which includes a fresh install of your application. Develop a CloudFormation template which includes your AMI and the required EC2, AutoScaling, and ELB resources to support deploying the application across Multiple- Availability-Zones. Asynchronously replicate transactions from your on-premises database to a database instance in AWS across a secure VPN connection.
B. Deploy your application on EC2 instances within an Auto Scaling group across multiple availability zones. Asynchronously replicate transactions from your on-premises database to a database instance in AWS across a secure VPN connection.
C. Create an EBS backed private AMI which includes a fresh install of your application. Setup a script in your data center to backup the local database every 1 hour and to encrypt and copy the resulting file to an S3 bucket using multi-part upload.
D. Install your application on a compute-optimized EC2 instance capable of supporting the application's average load. Synchronously replicate transactions from your on-premises database to a database instance in AWS across a secure Direct Connect connection.

Answer: A

Explanation:
Overview of Creating Amazon EBS-Backed AMIs
First, launch an instance from an AMI that's similar to the AMI that you'd like to create. You can connect to your instance and customize it. When the instance is configured correctly, ensure data integrity by stopping the instance before you create an AMI, then create the image. When you create an Amazon EBS-backed AMI, we automatically register it for you.
Amazon EC2 powers down the instance before creating the AMI to ensure that everything on the instance is stopped and in a consistent state during the creation process. If you're confident that your instance is in a consistent state appropriate for AMI creation, you can tell Amazon EC2 not to power down and reboot the instance. Some file systems, such as XFS, can freeze and unfreeze activity, making it safe to create the image without rebooting the instance.

During the AMI-creation process, Amazon EC2 creates snapshots of your instance's root volume and any other EBS volumes attached to your instance. If any volumes attached to the instance are encrypted, the new AMI only launches successfully on instances that support Amazon EBS encryption. For more information, see Amazon EBS Encryption.
Depending on the size of the volumes, it can take several minutes for the AMI-creation process to complete (sometimes up to 24 hours).You may find it more efficient to create snapshots of your volumes prior to creating your AMI. This way, only small, incremental snapshots need to be created when the AMI is created, and the process completes more quickly (the total time for snapshot creation remains the same). For more information, see Creating an Amazon EBS Snapshot.
After the process completes, you have a new AMI and snapshot created from the root volume of the instance. When you launch an instance using the new AMI, we create a new EBS volume for its root volume using the snapshot. Both the AMI and the snapshot incur charges to your account until you delete them. For more information, see Deregistering Your AMI.
If you add instance-store volumes or EBS volumes to your instance in addition to the root device volume, the block device mapping for the new AMI contains information for these volumes, and the block device mappings for instances that you launch from the new AMI automatically contain information for these volumes. The instance-store volumes specified in the block device mapping for the new instance are new and don't contain any data from the instance store volumes of the instance you used to create the AMI. The data on EBS volumes persists. For more information, see Block Device Mapping.

Question: 20

An ERP application is deployed across multiple AZs in a single region. In the event of failure, the Recovery Time Objective (RTO) must be less than 3 hours, and the Recovery Point Objective (RPO) must be 15 minutes the customer realizes that data corruption occurred roughly 1.5 hours ago.
What DR strategy could be used to achieve this RTO and RPO in the event of this kind of failure?

A. Take hourly DB backups to S3, with transaction logs stored in S3 every 5 minutes.
B. Use synchronous database master-slave replication between two availability zones.
C. Take hourly DB backups to EC2 Instance store volumes with transaction logs stored In S3 every 5 minutes.
D. Take 15 minute DB backups stored In Glacier with transaction logs stored in S3 every 5 minutes.

Answer: A

Question: 21

Your startup wants to implement an order fulfillment process for selling a personalized gadget that needs an average of 3-4 days to produce with some orders taking up to 6 months you expect 10 orders per day on your first day. 1000 orders per day after 6 months and 10,000 orders after 12 months.
Orders coming in are checked for consistency men dispatched to your manufacturing plant for production quality control packaging shipment and payment processing If the product does not meet the quality standards at any stage of the process employees may force the process to repeat a step Customers are notified via email about order status and any critical issues with their orders such as payment failure.

Your case architecture includes AWS Elastic Beanstalk for your website with an RDS MySQL instance for customer data and orders.
How can you implement the order fulfillment process while making sure that the emails are delivered reliably?

A. Add a business process management application to your Elastic Beanstalk app servers and re-use the ROS database for tracking order status use one of the Elastic Beanstalk instances to send emails to customers.
B. Use SWF with an Auto Scaling group of activity workers and a decider instance in another Auto Scaling group with min/max=1 Use the decider instance to send emails to customers.
C. Use SWF with an Auto Scaling group of activity workers and a decider instance in another Auto Scaling group with min/max=1 use SES to send emails to customers.
D. Use an SQS queue to manage all process tasks Use an Auto Scaling group of EC2 Instances that poll the tasks and execute them. Use SES to send emails to customers.

Answer: C

Question: 22

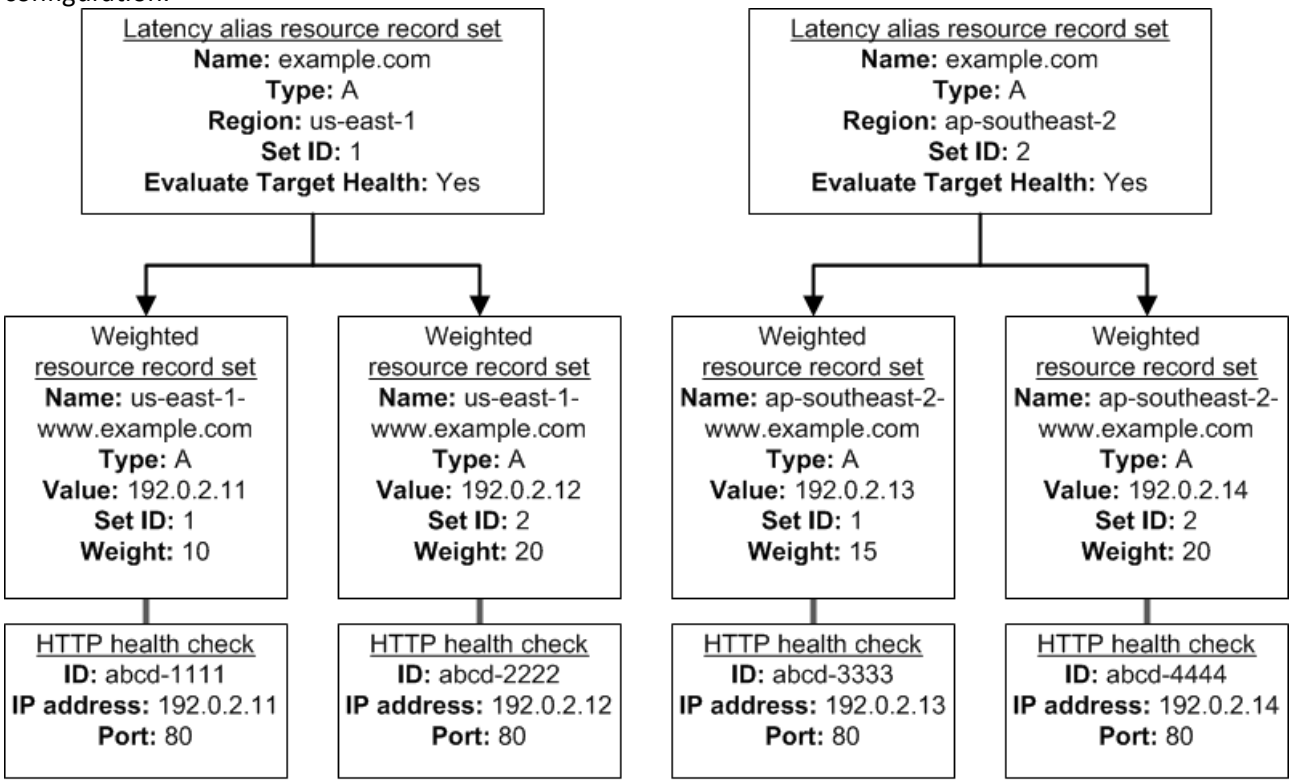
You have deployed a web application targeting a global audience across multiple AWS Regions under the domain name.example.com. You decide to use Route53 Latency-Based Routing to serve web requests to users from the region closest to the user. To provide business continuity in the event of server downtime you configure weighted record sets associated with two web servers in separate Availability Zones per region. Dunning a DR test you notice that when you disable all web servers in one of the regions Route53 does not automatically direct all users to the other region. What could be happening? (Choose 2 answers)

A. Latency resource record sets cannot be used in combination with weighted resource record sets.
B. You did not setup an HTTP health check tor one or more of the weighted resource record sets associated with me disabled web servers.
C. The value of the weight associated with the latency alias resource record set in the region with the disabled servers is higher than the weight for the other region.
D. One of the two working web servers in the other region did not pass its HTTP health check.
E. You did not set "Evaluate Target Health" to "Yes" on the latency alias resource record set associated with example com in the region where you disabled the servers.

Answer: B, E

Explanation:
How Health Checks Work in Complex Amazon Route 53 Configurations
Checking the health of resources in complex configurations works much the same way as in simple configurations. However, in complex configurations, you use a combination of alias resource record sets (including weighted alias, latency alias, and failover alias) and nonalias resource record sets to build a decision tree that gives you greater control over how Amazon Route 53 responds to requests. For more information, see How Health Checks Work in Simple Amazon Route 53 Configurations.
For example, you might use latency alias resource record sets to select a region close to a user and use weighted resource record sets for two or more resources within each region to protect against

the failure of a single endpoint or an Availability Zone. The following diagram shows this configuration.



Here's how Amazon EC2 and Amazon Route 53 are configured:

You have Amazon EC2 instances in two regions, us-east-1 and ap-southeast-2. You want Amazon Route 53 to respond to queries by using the resource record sets in the region that provides the lowest latency for your customers, so you create a latency alias resource record set for each region. (You create the latency alias resource record sets after you create resource record sets for the individual Amazon EC2 instances.)

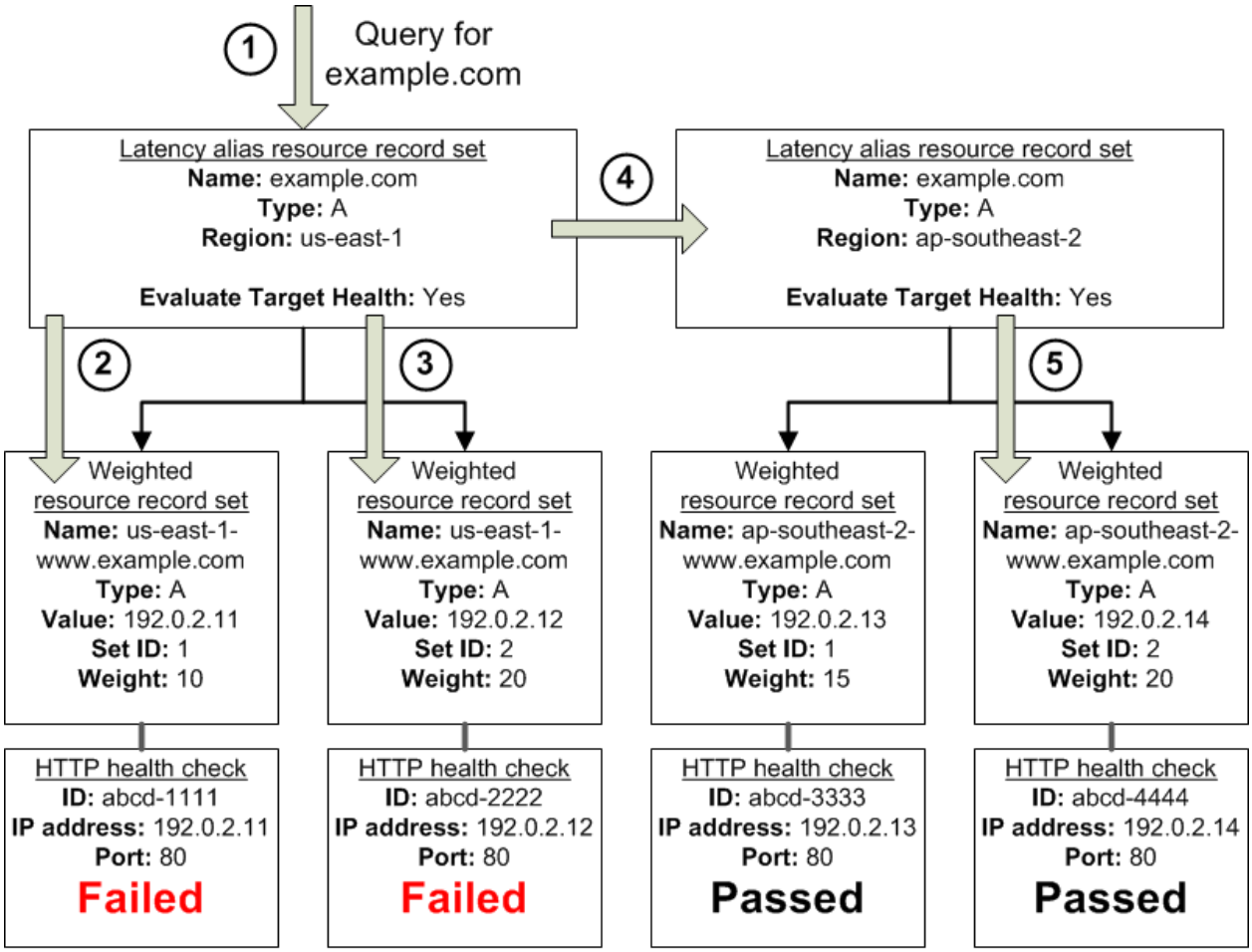
Within each region, you have two Amazon EC2 instances. You create a weighted resource record set for each instance. The name and the type are the same for both of the weighted resource record sets in each region.

When you have multiple resources in a region, you can create weighted or failover resource record sets for your resources. You can also create even more complex configurations by creating weighted alias or failover alias resource record sets that, in turn, refer to multiple resources.

Each weighted resource record set has an associated health check. The IP address for each health check matches the IP address for the corresponding resource record set. This isn't required, but it's the most common configuration.

For both latency alias resource record sets, you set the value of Evaluate Target Health to Yes.

You use the Evaluate Target Health setting for each latency alias resource record set to make Amazon Route 53 evaluate the health of the alias targets—the weighted resource record sets—and respond accordingly.



The preceding diagram illustrates the following sequence of events:

Amazon Route 53 receives a query for example.com. Based on the latency for the user making the request, Amazon Route 53 selects the latency alias resource record set for the us-east-1 region.

Amazon Route 53 selects a weighted resource record set based on weight. Evaluate Target Health is Yes for the latency alias resource record set, so Amazon Route 53 checks the health of the selected weighted resource record set.

The health check failed, so Amazon Route 53 chooses another weighted resource record set based on weight and checks its health. That resource record set also is unhealthy.

Amazon Route 53 backs out of that branch of the tree, looks for the latency alias resource record set with the next-best latency, and chooses the resource record set for ap-southeast-2.

Amazon Route 53 again selects a resource record set based on weight, and then checks the health of the selected resource record set. The health check passed, so Amazon Route 53 returns the applicable value in response to the query.

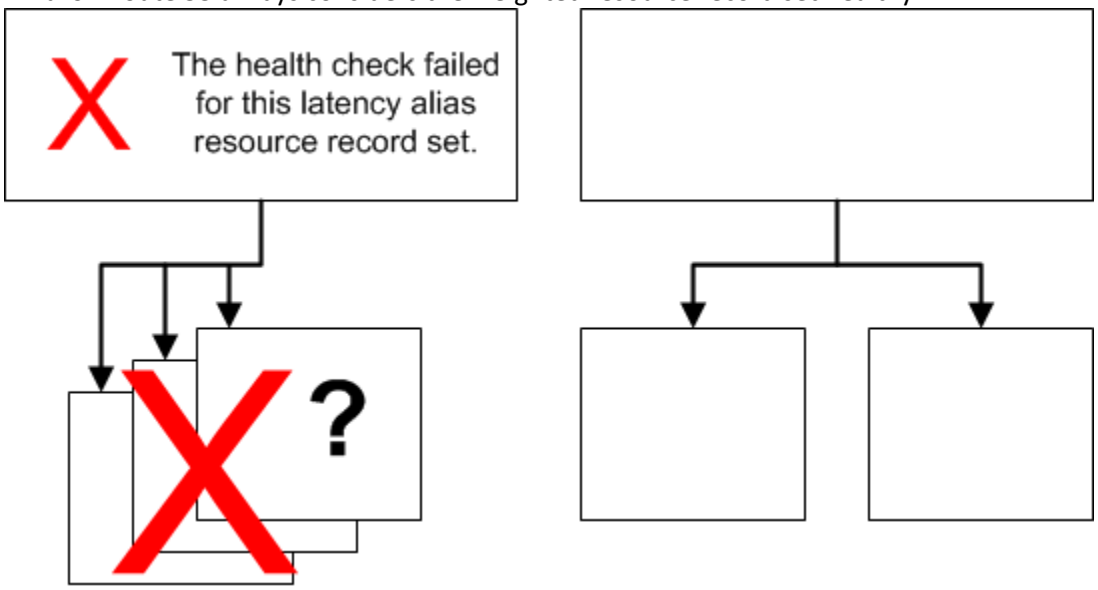
What Happens When You Associate a Health Check with an Alias Resource Record Set?

You can associate a health check with an alias resource record set instead of or in addition to setting the value of Evaluate Target Health to Yes. However, it's generally more useful if Amazon Route 53 responds to queries based on the health of the underlying resources—the HTTP servers, database servers, and other resources that your alias resource record sets refer to. For example, suppose the following configuration:

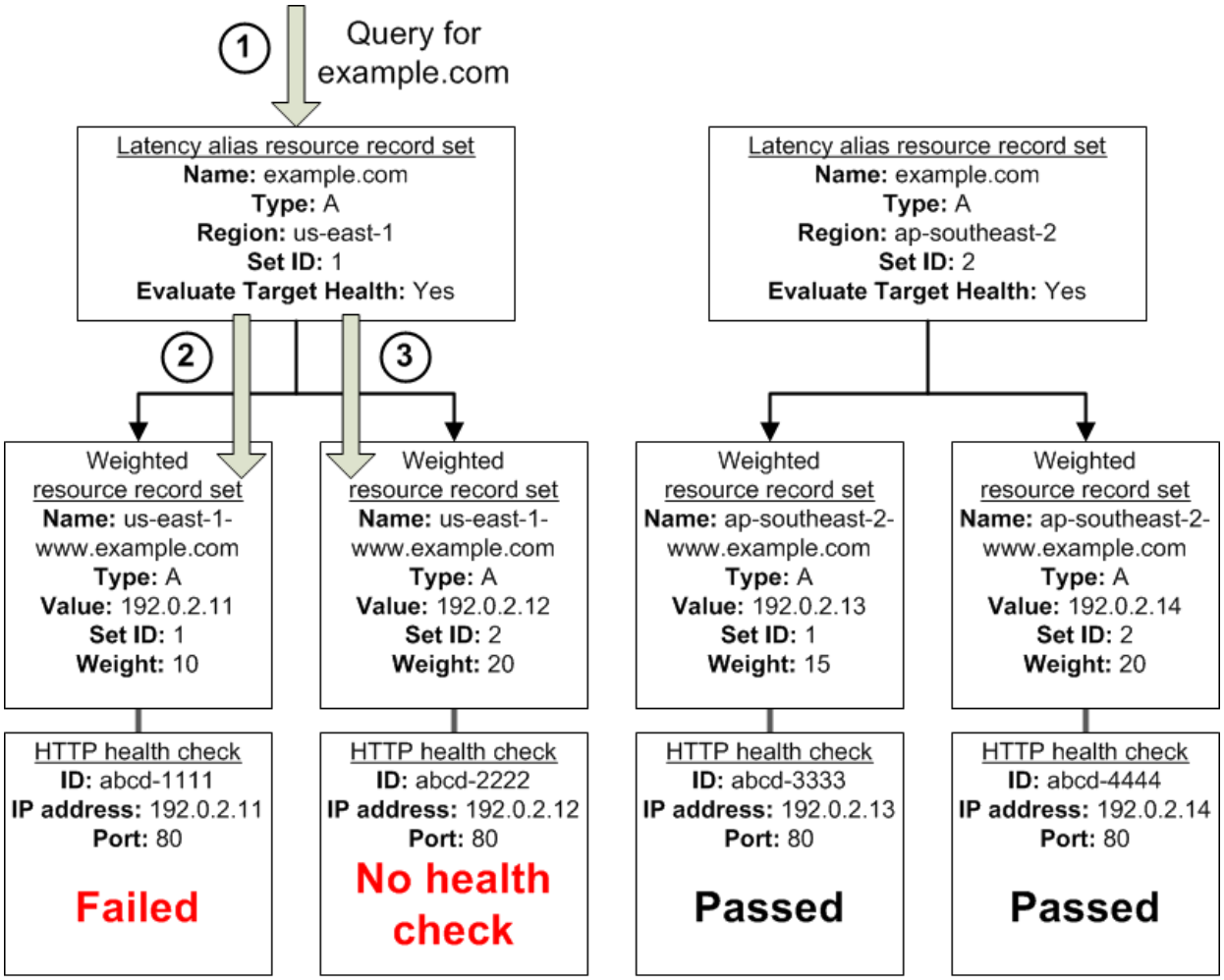
You assign a health check to a latency alias resource record set for which the alias target is a group of weighted resource record sets.

You set the value of Evaluate Target Health to Yes for the latency alias resource record set.

In this configuration, both of the following must be true before Amazon Route 53 will return the applicable value for a weighted resource record set:
The health check associated with the latency alias resource record set must pass.
At least one weighted resource record set must be considered healthy, either because it's associated with a health check that passes or because it's not associated with a health check. In the latter case, Amazon Route 53 always considers the weighted resource record set healthy.



If the health check for the latency alias resource record set fails, Amazon Route 53 stops responding to queries using any of the weighted resource record sets in the alias target, even if they're all healthy. Amazon Route 53 doesn't know the status of the weighted resource record sets because it never looks past the failed health check on the alias resource record set.
What Happens When You Omit Health Checks?
In a complex configuration, it's important to associate health checks with all of the non-alias resource record sets. Let's return to the preceding example, but assume that a health check is missing on one of the weighted resource record sets in the us-east-1 region:



Here's what happens when you omit a health check on a non-alias resource record set in this configuration:

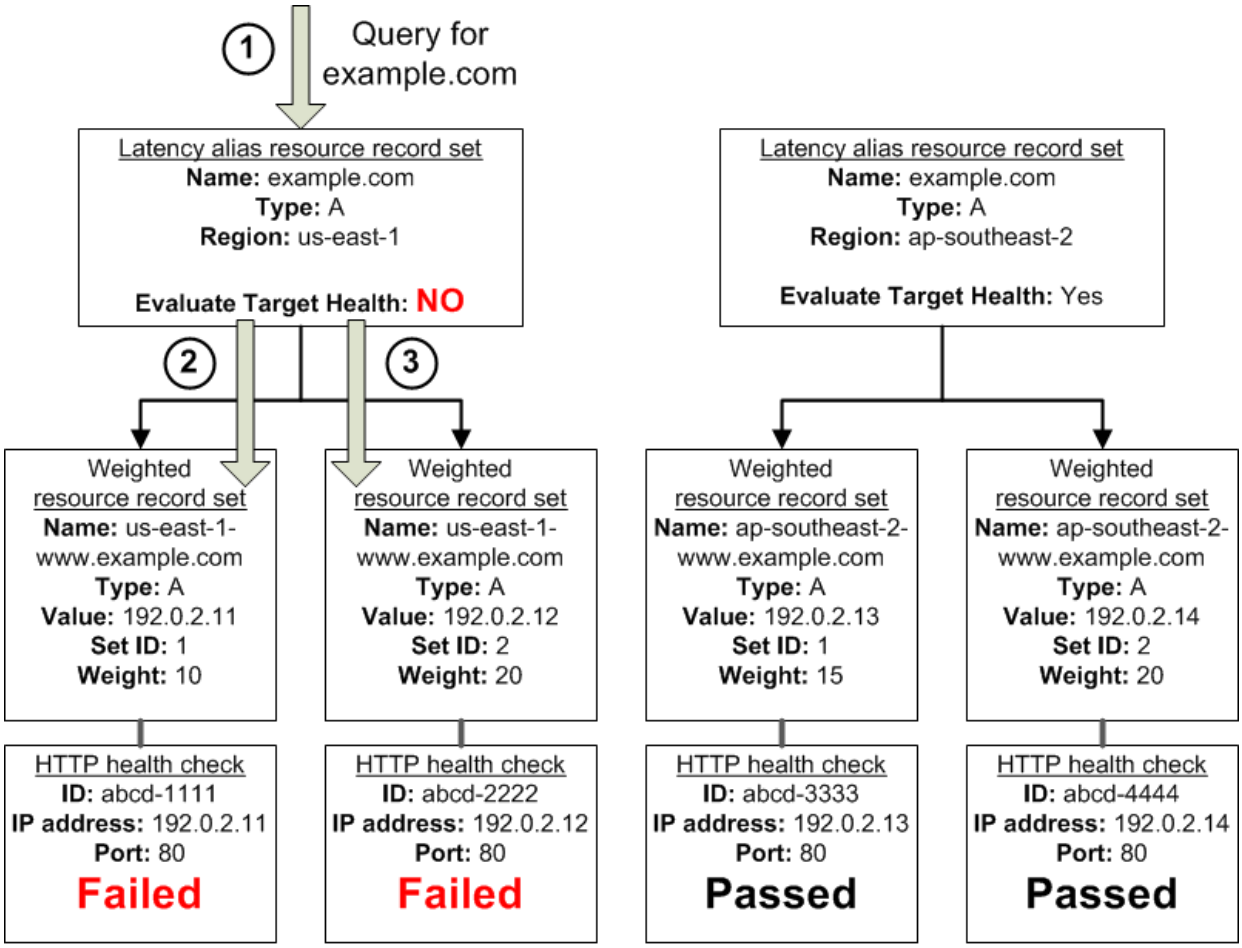
Amazon Route 53 receives a query for example.com. Based on the latency for the user making the request, Amazon Route 53 selects the latency alias resource record set for the us-east-1 region.

Amazon Route 53 looks up the alias target for the latency alias resource record set, and checks the status of the corresponding health checks. The health check for one weighted resource record set failed, so that resource record set is omitted from consideration.

The other weighted resource record set in the alias target for the us-east-1 region has no health check. The corresponding resource might or might not be healthy, but without a health check, Amazon Route 53 has no way to know. Amazon Route 53 assumes that the resource is healthy and returns the applicable value in response to the query.

What Happens When You Set Evaluate Target Health to No?

In general, you also want to set Evaluate Target Health to Yes for all of the alias resource record sets. In the following example, all of the weighted resource record sets have associated health checks, but Evaluate Target Health is set to No for the latency alias resource record set for the us-east-1 region:



Here's what happens when you set Evaluate Target Health to No for an alias resource record set in this configuration:

Amazon Route 53 receives a query for example.com. Based on the latency for the user making the request, Amazon Route 53 selects the latency alias resource record set for the us-east-1 region.

Amazon Route 53 determines what the alias target is for the latency alias resource record set, and checks the corresponding health checks. They're both failing.

Because the value of Evaluate Target Health is No for the latency alias resource record set for the us-east-1 region, Amazon Route 53 must choose one resource record set in this branch instead of backing out of the branch and looking for a healthy resource record set in the ap-southeast-2 region.

Question: 23

Your company hosts a social media site supporting users in multiple countries. You have been asked to provide a highly available design tor the application that leverages multiple regions tor the most recently accessed content and latency sensitive portions of the wet) site The most latency sensitive component of the application involves reading user preferences to support web site personalization and ad selection.

In addition to running your application in multiple regions, which option will support this application's requirements?

A. Serve user content from S3. CloudFront and use Route53 latency-based routing between ELBs in each region Retrieve user preferences from a local DynamoDB table in each region and leverage SQS

to capture changes to user preferences with SOS workers for propagating updates to each table.
B. Use the S3 Copy API to copy recently accessed content to multiple regions and serve user content from S3. CloudFront with dynamic content and an ELB in each region Retrieve user preferences from an ElasticCache cluster in each region and leverage SNS notifications to propagate user preference changes to a worker node in each region.
C. Use the S3 Copy API to copy recently accessed content to multiple regions and serve user content from S3 CloudFront and Route53 latency-based routing Between ELBs In each region Retrieve user preferences from a DynamoDB table and leverage SQS to capture changes to user preferences with SOS workers for propagating DynamoDB updates.
D. Serve user content from S3. CloudFront with dynamic content, and an ELB in each region Retrieve user preferences from an ElastiCache cluster in each region and leverage Simple Workflow (SWF) to manage the propagation of user preferences from a centralized OB to each ElastiCache cluster.

Answer: A

Question: 24

Your system recently experienced down time during the troubleshooting process. You found that a new administrator mistakenly terminated several production EC2 instances.
Which of the following strategies will help prevent a similar situation in the future?
The administrator still must be able to:
- launch, start stop, and terminate development resources.
- launch and start production instances.

A. Create an IAM user, which is not allowed to terminate instances by leveraging production EC2 termination protection.
B. Leverage resource based tagging along with an IAM user, which can prevent specific users from terminating production EC2 resources.
C. Leverage EC2 termination protection and multi-factor authentication, which together require users to authenticate before terminating EC2 instances
D. Create an IAM user and apply an IAM role which prevents users from terminating production EC2 instances.

Answer: B

Explanation:
Working with volumes
When an API action requires a caller to specify multiple resources, you must create a policy statement that allows users to access all required resources. If you need to use a Condition element with one or more of these resources, you must create multiple statements as shown in this example.
The following policy allows users to attach volumes with the tag "volume_user=iam-user-name" to instances with the tag "department=dev", and to detach those volumes from those instances. If you attach this policy to an IAM group, the aws:username policy variable gives each IAM user in the group permission to attach or detach volumes from the instances with a tag named volume_user that has his or her IAM user name as a value.

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "ec2:AttachVolume",
      "ec2:DetachVolume"
    ],
    "Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*",
    "Condition": {
      "StringEquals": {
        "ec2:ResourceTag/department": "dev"
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "ec2:AttachVolume",
      "ec2:DetachVolume"
    ],
    "Resource": "arn:aws:ec2:us-east-1:123456789012:volume/*",
    "Condition": {
      "StringEquals": {
        "ec2:ResourceTag/volume_user": "${aws:username}"
      }
    }
  }
 ]
}
```

Launching instances (RunInstances)

The RunInstances API action launches one or more instances. RunInstances requires an AMI and creates an instance; and users can specify a key pair and security group in the request. Launching into EC2-VPC requires a subnet, and creates a network interface. Launching from an Amazon EBS-backed AMI creates a volume. Therefore, the user must have permission to use these Amazon EC2 resources. The caller can also configure the instance using optional parameters to RunInstances, such as the instance type and a subnet. You can create a policy statement that requires users to specify an optional parameter, or restricts users to particular values for a parameter. The examples in this section demonstrate some of the many possible ways that you can control the configuration of an instance that a user can launch.

Note that by default, users don't have permission to describe, start, stop, or terminate the resulting instances. One way to grant the users permission to manage the resulting instances is to create a specific tag for each instance, and then create a statement that enables them to manage instances with that tag. For more information, see 2: Working with instances.

a. AMI

The following policy allows users to launch instances using only the AMIs that have the specified tag, "department=dev", associated with them. The users can't launch instances using other AMIs because the Condition element of the first statement requires that users specify an AMI that has this tag. The users also can't launch into a subnet, as the policy does not grant permissions for the subnet and network interface resources. They can, however, launch into EC2-Classic. The second statement uses

a wildcard to enable users to create instance resources, and requires users to specify the key pair project_keypair and the security group sg-1a2b3c4d. Users are still able to launch instances without a key pair.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": [
      "arn:aws:ec2:region::image/ami-*"
    ],
    "Condition": {
      "StringEquals": {
        "ec2:ResourceTag/department": "dev"
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": [
      "arn:aws:ec2:region:account:instance/*",
      "arn:aws:ec2:region:account:volume/*",
      "arn:aws:ec2:region:account:key-pair/project_keypair",
      "arn:aws:ec2:region:account:security-group/sg-1a2b3c4d"
      ]
    }
  ]
}
```

Alternatively, the following policy allows users to launch instances using only the specified AMIs, ami-9e1670f7 and ami-45cf5c3c. The users can't launch an instance using other AMIs (unless another statement grants the users permission to do so), and the users can't launch an instance into a subnet.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": [
      "arn:aws:ec2:region::image/ami-9e1670f7",
      "arn:aws:ec2:region::image/ami-45cf5c3c",
      "arn:aws:ec2:region:account:instance/*",
      "arn:aws:ec2:region:account:volume/*",
      "arn:aws:ec2:region:account:key-pair/*",
      "arn:aws:ec2:region:account:security-group/*"
    ]
   }
  ]
}
```

Alternatively, the following policy allows users to launch instances from all AMIs owned by Amazon. The Condition element of the first statement tests whether ec2:Owner is amazon. The users can't launch an instance using other AMIs (unless another statement grants the users permission to do so). The users are able to launch an instance into a subnet.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": [
      "arn:aws:ec2:region::image/ami-*"
    ],
    "Condition": {
      "StringEquals": {
        "ec2:Owner": "amazon"
        }
    }
  },
  {
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": [
      "arn:aws:ec2:region:account:instance/*",
      "arn:aws:ec2:region:account:subnet/*",
      "arn:aws:ec2:region:account:volume/*",
      "arn:aws:ec2:region:account:network-interface/*",
      "arn:aws:ec2:region:account:key-pair/*",
      "arn:aws:ec2:region:account:security-group/*"
      ]
    }
  ]
}
```

b. Instance type

The following policy allows users to launch instances using only the t2.micro or t2.small instance type, which you might do to control costs. The users can't launch larger instances because the Condition element of the first statement tests whether ec2:InstanceType is either t2.micro or t2.small.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": [
      "arn:aws:ec2:region:account:instance/*"
    ],
    "Condition": {
      "StringEquals": {
        "ec2:InstanceType": ["t2.micro", "t2.small"]
```

```
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": [
      "arn:aws:ec2:region::image/ami-*",
      "arn:aws:ec2:region:account:subnet/*",
      "arn:aws:ec2:region:account:network-interface/*",
      "arn:aws:ec2:region:account:volume/*",
      "arn:aws:ec2:region:account:key-pair/*",
      "arn:aws:ec2:region:account:security-group/*"
      ]
    }
  ]
}
```

Alternatively, you can create a policy that denies users permission to launch any instances except t2.micro and t2.small instance types.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Action": "ec2:RunInstances",
    "Resource": [
      "arn:aws:ec2:region:account:instance/*"
    ],
    "Condition": {
      "StringNotEquals": {
        "ec2:InstanceType": ["t2.micro", "t2.small"]
        }
    }
  },
  {
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": [
      "arn:aws:ec2:region::image/ami-*",
      "arn:aws:ec2:region:account:network-interface/*",
      "arn:aws:ec2:region:account:instance/*",
      "arn:aws:ec2:region:account:subnet/*",
      "arn:aws:ec2:region:account:volume/*",
      "arn:aws:ec2:region:account:key-pair/*",
      "arn:aws:ec2:region:account:security-group/*"
      ]
    }
  ]
}
```

c. Subnet
The following policy allows users to launch instances using only the specified subnet, subnet-12345678. The group can't launch instances into any another subnet (unless another statement grants the users permission to do so). Users are still able to launch instances into EC2-Classic.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": [
      "arn:aws:ec2:region:account:subnet/subnet-12345678",
      "arn:aws:ec2:region:account:network-interface/*",
      "arn:aws:ec2:region:account:instance/*",
      "arn:aws:ec2:region:account:volume/*",
      "arn:aws:ec2:region::image/ami-*",
      "arn:aws:ec2:region:account:key-pair/*",
      "arn:aws:ec2:region:account:security-group/*"
    ]
   }
  ]
}
```

Alternatively, you could create a policy that denies users permission to launch an instance into any other subnet. The statement does this by denying permission to create a network interface, except where subnet subnet-12345678 is specified. This denial overrides any other policies that are created to allow launching instances into other subnets. Users are still able to launch instances into EC2-Classic.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Action": "ec2:RunInstances",
    "Resource": [
      "arn:aws:ec2:region:account:network-interface/*"
    ],
    "Condition": {
      "ArnNotEquals": {
        "ec2:Subnet": "arn:aws:ec2:region:account:subnet/subnet-12345678"
        }
    }
  },
  {
    "Effect": "Allow",
    "Action": "ec2:RunInstances",
    "Resource": [
      "arn:aws:ec2:region::image/ami-*",
      "arn:aws:ec2:region:account:network-interface/*",
      "arn:aws:ec2:region:account:instance/*",
      "arn:aws:ec2:region:account:subnet/*",
```

```
      "arn:aws:ec2:region:account:volume/*",
      "arn:aws:ec2:region:account:key-pair/*",
      "arn:aws:ec2:region:account:security-group/*"
      ]
    }
  ]
}
```

Question: 25

A customer has established an AWS Direct Connect connection to AWS. The link is up and routes are being advertised from the customer's end, however the customer is unable to connect from EC2 instances inside its VPC to servers residing in its datacenter.
Which of the following options provide a viable solution to remedy this situation? (Choose 2 answers)

A. Add a route to the route table with an iPsec VPN connection as the target.
B. Enable route propagation to the virtual pinnate gateway (VGW).
C. Enable route propagation to the customer gateway (CGW).
D. Modify the route table of all Instances using the 'route' command.
E. Modify the Instances VPC subnet route table by adding a route back to the customer's on-premises environment.

Answer: A, C