Exam AWS Certified Developer - Associate DVA-C02 All Questions

View all questions & answers for the AWS Certified Developer - Associate DVA-C02 exam

Go to Exam

EXAM AWS CERTIFIED DEVELOPER - ASSOCIATE DVA-C02 TOPIC 1 QUESTION 1 DISCUS...

Exam question from Amazon's AWS Certified Developer - Associate DVA-C02
Question #: 1
Topic #: 1
[All AWS Certified Developer - Associate DVA-C02 Questions]

A company is implementing an application on Amazon EC2 instances. The application needs to process incoming transactions. When the application detects a transaction that is not valid, the application must send a chat message to the company's support team. To send the message, the application needs to retrieve the access token to authenticate by using the chat API.
A developer needs to implement a solution to store the access token. The access token must be encrypted at rest and in transit. The access token must also be accessible from other AWS accounts.
Which solution will meet these requirements with the LEAST management overhead?

A. Use an AWS Systems Manager Parameter Store SecureString parameter that uses an AWS Key Management Service (AWS KMS) AWS managed key to store the access token. Add a resource-based policy to the parameter to allow access from other accounts. Update the IAM role of the EC2 instances with permissions to access Parameter Store. Retrieve the token from Parameter Store with the decrypt flag enabled. Use the decrypted access token to send the message to the chat.

B. Encrypt the access token by using an AWS Key Management Service (AWS KMS) customer managed key. Store the access token in an Amazon DynamoDB table. Update the IAM role of the EC2 instances with permissions to access DynamoDB and AWS KMS. Retrieve the token from DynamoDDecrypt the token by using AWS KMS on the EC2 instances. Use the decrypted access token to send the message to the chat.

C. Use AWS Secrets Manager with an AWS Key Management Service (AWS KMS) customer managed key to store the access token. Add a resource-based policy to the secret to allow access from other accounts. Update the IAM role of the EC2 instances with permissions to access Secrets Manager. Retrieve the token from Secrets Manager. Use the decrypted access token to send the message to the chat.

D. Encrypt the access token by using an AWS Key Management Service (AWS KMS) AWS managed key. Store the access token in an Amazon S3 bucket. Add a bucket policy to the S3 bucket to allow access from other accounts. Update the IAM role of the EC2 instances with permissions to access Amazon S3 and AWS KMS. Retrieve the token from the S3 bucket. Decrypt the token by using AWS KMS on the EC2 instances. Use the decrypted access token to send the massage to the chat.

Show Suggested Answer

by good_ at *March 16, 2023, 9:21 a.m.*

Disclaimers:
- ExamTopics website is **not** related to, affiliated with, endorsed or authorized by Amazon.
- Trademarks, certification & product names are used for reference only and belong to Amazon.

Comments

Untamables Highly Voted 2 years, 1 month ago

Selected Answer: C
The correct answer is C.
https://aws.amazon.com/premiumsupport/knowledge-center/secrets-manager-share-between-accounts/
https://docs.aws.amazon.com/secretsmanager/latest/userguide/auth-and-access_examples_cross.html
Option A is wrong. It seems to be a good solution. However, AWS managed keys cannot be used for cross account accessing.
upvoted 30 times

CyberBaby803 2 years, 1 month ago
Based on this AWS managed keys can be used for cross account accessing.
https://docs.aws.amazon.com/kms/latest/developerguide/key-policy-modifying-external-accounts.html
upvoted 2 times

AgboolaKun 1 year, 11 months ago
I am not sure if the documentation you provided specifically say that AWS managed keys can be used for cross account accessing.

However, @Untamables explanation is on point. Please see this Stack Overflow thread - https://stackoverflow.com/questions/63420732/sharing-an-aws-managed-kms-key-with-another-account
upvoted 1 times

jipark 1 year, 9 months ago
cross account, rotate is key for 'Security Manager'
upvoted 2 times

anandkg Most Recent 1 month ago

Selected Answer: C
resource based policy works with secrets manger not with SSM
upvoted 1 times

sumanshu 4 months, 4 weeks ago

Selected Answer: C

For cross-account access, the AWS managed key (Option A) will be difficult to manage because it doesn't allow you to directly manage cross-account access. Therefore, Option C (AWS Secrets Manager with a customer-managed KMS key) is the recommended solution for cross-account access and security.

upvoted 1 times

sumanshu 4 months, 4 weeks ago

B) Eliminated - Creating and managing the DynamoDB table create overhead

upvoted 1 times

sumanshu 4 months, 4 weeks ago

D) Eliminated - Complexity and overhead for Managing an S3 bucket with access policies for cross-account access

upvoted 1 times

trieudo 5 months, 1 week ago

Selected Answer: C

keywords: LEAST management overhead

==> Discard B, D: you must do many steps to config with Storage, DB with KMS, IAM Role
==> Discard A: Pretty correct, but in use, you may write some script. It can work but requires additional configuration and doesn't offer some of the benefits tailored for secrets management like automatic rotation.

D: The solution with AWS Secrets Manager (option C) provides the least management overhead because:

Secrets Manager is specifically designed for storing and managing sensitive information like access tokens.
It natively integrates with AWS KMS for encryption and decryption.
It simplifies access control and auditing.
By adding a resource-based policy, cross-account access is easily managed without the need for additional configurations like DynamoDB tables or S3 bucket policies.

upvoted 1 times

trieudo 5 months, 1 week ago

D: ==> C: ... sr for inconsistent

upvoted 1 times

Tee400 7 months, 3 weeks ago

Selected Answer: A

Use an AWS Systems Manager Parameter Store SecureString parameter that uses an AWS Key Management Service (AWS KMS) AWS managed key to store the access token. This option allows you to securely store the access token in the Parameter Store, which automatically encrypts the data at rest and in transit. By adding a resource-based policy, you can also grant access to the access token from other AWS accounts. The IAM role of the EC2 instances can be updated to allow permissions to access the Parameter Store, and the access token can be retrieved with the decrypt flag enabled for use in sending the chat message. This option requires minimal setup and management compared to the other choices.

upvoted 1 times

cgpt 7 months, 3 weeks ago

Selected Answer: A

By default, AWS Systems Manager Parameter Store does not natively support cross-account access for SecureString parameters. However, you can configure cross-account access to SecureString parameters by sharing the KMS key with the target AWS accounts. To do this, you need to create a resource-based KMS key policy that allows access to the key by the external AWS account(s). After configuring the KMS key policy to allow the necessary cross-account access, you can grant IAM roles in the target accounts permission to access the SecureString parameters that are encrypted using that KMS key.

upvoted 2 times

tomchandler077 7 months, 3 weeks ago

AWS Secrets Manager (Option C) is designed for exactly this kind of use case, providing built-in functionality for secure storage and retrieval of secrets with minimal management overhead, especially for managing access tokens and cross-account access.
Amazon S3 with KMS (Option D), while familiar and powerful, requires more manual work to set up and manage the security aspects, which can lead to increased overhead in comparison to Secrets Manager.
Given that the goal is to have the least management overhead, Option C is the best fit because it is purpose-built for managing secrets and automates much of the complexity involved in secure storage and retrieval.

upvoted 1 times

Anandesh 10 months ago

Selected Answer: C

https://docs.aws.amazon.com/secretsmanager/latest/userguide/data-protection.html
https://docs.aws.amazon.com/secretsmanager/latest/userguide/auth-and-access_resource-policies.html
https://docs.aws.amazon.com/secretsmanager/latest/userguide/security-encryption.html

upvoted 2 times

nkroker 10 months, 2 weeks ago

C is the correct answer as the Secrets Manager supports resource-based policies, allowing you to grant access to other AWS accounts easily.

upvoted 1 times

NagaoShingo 11 months, 2 weeks ago

Selected Answer: C

C is correct answer.

upvoted 1 times

65703c1 12 months ago

Selected Answer: C

C is the correct answer.

upvoted 1 times

shabeebcoder 1 year, 1 month ago

Selected Answer: C

This is the correct answer for lease overhead to manage the secret key

upvoted 1 times

ibratoev 1 year, 1 month ago

Selected Answer: C

It is C

upvoted 1 times

certplan 1 year, 2 months ago

- Option A involves using AWS Systems Manager Parameter Store, which can work but requires additional configuration and doesn't offer some of the benefits tailored for secrets management like automatic rotation.

- Option B involves storing the access token in DynamoDB, which is not specifically designed for secrets management, and managing encryption and decryption manually using AWS KMS.
- Option D involves using S3, which again is not designed for secrets management, and adds complexity in managing access policies and permissions. Additionally, accessing the token would involve reading from S3, decrypting it, and then using it, which is less straightforward compared to using a service like Secrets Manager.

upvoted 2 times

SD_CS 1 year, 3 months ago

Selected Answer: A

I think this would be A as this is cheaper than C. Any reason why A can not be the answer?

upvoted 2 times

TheFivePips 1 year, 2 months ago

From what I can find, You can apply resource-based policies at the Parameter Store level to control access to the entire Parameter Store service. However, you cannot apply resource-based policies directly to individual parameters within the Parameter Store.
That is seemingly the only reason I would choose C over A.
But were also not looking for whats cheapest, were looking for whats easiest to manage

upvoted 2 times

tsdsmth 1 year, 3 months ago

The answer would be C if an AWS-managed key was used, as Secrets Manager and KMS are good for situations like this. However, the use of a customer-managed key increases management overhead. So the best answer is D, not C.

upvoted 2 times

gilleep_17 1 year, 4 months ago

You cannot use a resource-based policy with a parameter in the Parameter Store. The stephen answser Option C is correct
Practise paper3

upvoted 1 times