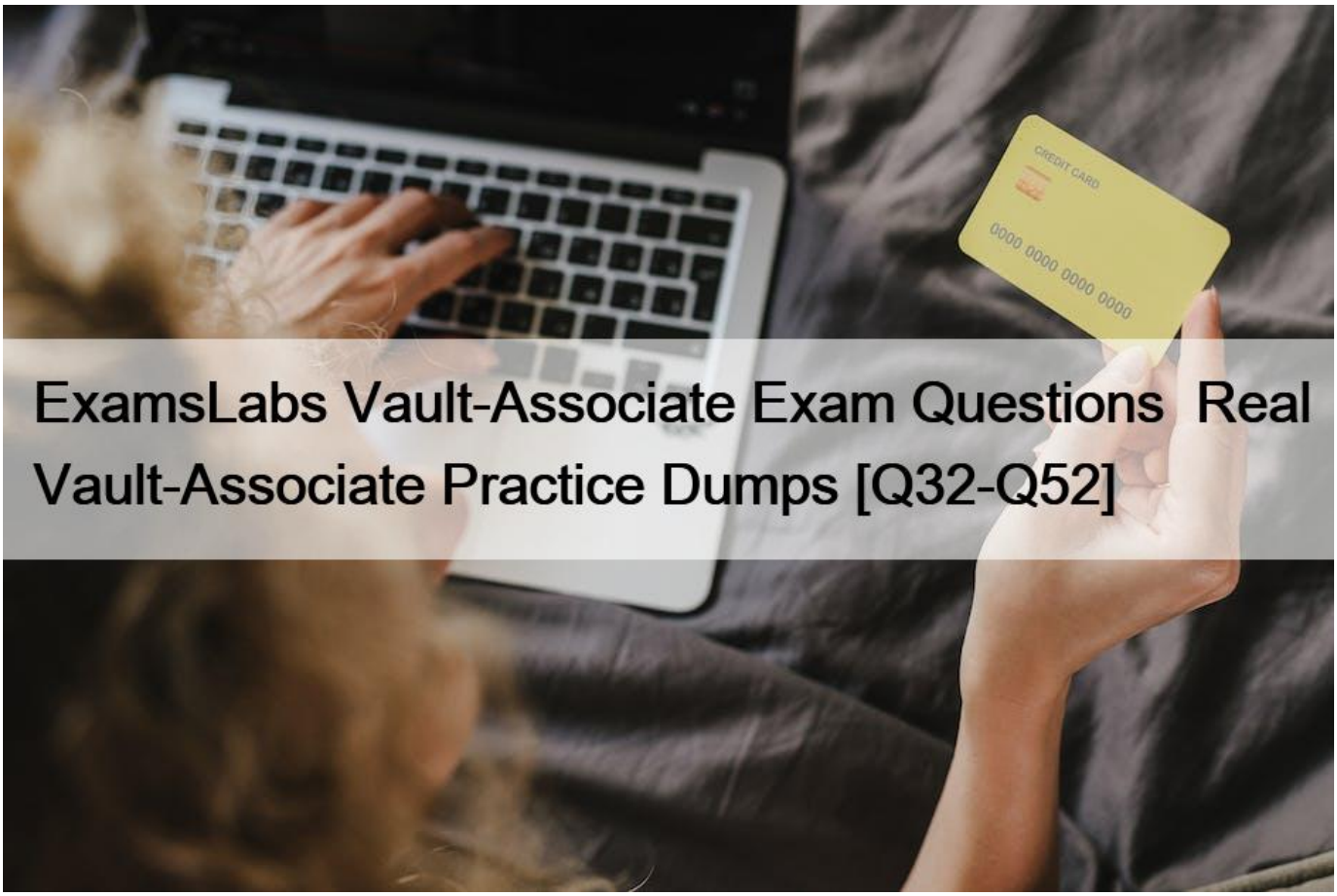
# ExamsLabs Vault-Associate Exam Questions  Real Vault-Associate Practice Dumps [Q32-Q52




ExamsLabs Vault-Associate Exam Questions | Real Vault-Associate Practice Dumps
Verified Vault-Associate Exam Dumps Q&As - Provide Vault-Associate with Correct Answers

## HashiCorp Vault-Associate Exam Syllabus Topics:

TopicDetailsTopic 1- Configure authentication methods-  Describe the encryption of data stored by VaultTopic 2- Describe root token uses and lifecycle-  Craft a Vault policy based on requirementsTopic 3- Choose a secret method based on use case-  Explain the purpose of a lease IDTopic 4- Describe authentication methods-  Illustrate the value of Vault policyTopic 5- Configure authentication methods-  Describe Vault policy syntax: capabilitiesTopic 6- Describe Shamir secret sharing and unsealing-  Differentiate between service and batch tokens. Choose one based on use-caseTopic 7- Compare and configure Vault secrets engines-  Contrast dynamic secrets vs. static secrets and their use casesTopic 8- Differentiate human vs. system auth methods-  Choose an authentication method based on use caseTopic 9- Be aware of identities and groups-  Explain the value of short-lived, dynamically generated secrets

**NEW QUESTION 32**

Security requirements demand that no secrets appear in the shell history. Which command does not meet this requirement?
* generate-password | vault kv put secret/password value
* vault kv put secret/password value-itsasecret
* vault kv put secret/password value=@data.txt
* vault kv put secret/password value-SSECRET_VALUE

The command that does not meet the security requirement of not having secrets appear in the shell history is B. vault kv put secret/password value-itsasecret. This command would store the secret value “itsasecret” in the key/value secrets engine at the path secret/password, but it would also expose the secret value in the shell history, which could be accessed by other users or malicious actors. This is not a secure way of storing secrets in Vault.

The other commands are more secure ways of storing secrets in Vault without revealing them in the shell history.

A). generate-password | vault kv put secret/password value would use a pipe to pass the output of the generate-password command, which could be a script or a tool that generates a random password, to the vault kv put command, which would store the password in the key/value secrets engine at the path secret/password. The password would not be visible in the shell history, only the commands.

C). vault kv put secret/password value=@data.txt would use the @ syntax to read the secret value from a file named data.txt, which could be encrypted or protected by file permissions, and store it in the key/value secrets engine at the path secret/password. The file name would be visible in the shell history, but not the secret value.

D). vault kv put secret/password value-SSECRET_VALUE would use the -S syntax to read the secret value from the environment variable SECRET_VALUE, which could be set and unset in the shell session, and store it in the key/value secrets engine at the path secret/password. The environment variable name would be visible in the shell history, but not the secret value.

Reference:

[Write Secrets | Vault | HashiCorp Developer]

**NEW QUESTION 33**

Which of these is not a benefit of dynamic secrets?
* Supports systems which do not natively provide a method of expiring credentials
* Minimizes damage of credentials leaking
* Ensures that administrators can see every password used
* Replaces cumbersome password rotation tools and practices

Dynamic secrets are generated on-demand by Vault and have a limited time-to-live (TTL). They do not ensure that administrators can see every password used, as they are often encrypted and ephemeral. The benefits of dynamic secrets are:

They support systems that do not natively provide a method of expiring credentials, such as databases, cloud providers, SSH, etc. Vault can revoke the credentials when they are no longer needed or when the lease expires.

They minimize the damage of credentials leaking, as they are short-lived and can be easily rotated or revoked. If a credential is compromised, the attacker has a limited window of opportunity to use it before it becomes invalid.

They replace cumbersome password rotation tools and practices, as Vault can handle the generation and revocation of credentials automatically and securely. This reduces the operational overhead and complexity of managing secrets.

**NEW QUESTION 34**

The following three policies exist in Vault. What do these policies allow an organization to do?

**app.hcl**

```
path "transit/encrypt/my_app_key" {
  capabilities = ["update"]
}
```

**callcenter.hcl**

```
path "transit/decrypt/my_app_key" {
  capabilities = ["update"]
}
```



**rewrap.hcl**

```
path "transit/keys/my_app_key" {
  capabilities = ["read"]
}

path "transit/rewrap/my_app_key" {
  capabilities = ["update"]
}
```

* Separates permissions allowed on actions associated with the transit secret engine
* Nothing, as the minimum permissions to perform useful tasks are not present
* Encrypt, decrypt, and rewrap data using the transit engine all in one policy
* Create a transit encryption key for encrypting, decrypting, and rewrapping encrypted data

The three policies that exist in Vault are:

admins: This policy grants full access to all secrets and operations in Vault. It can be used by administrators or operators who need to manage all aspects of Vault.

default: This policy grants access to all secrets and operations in Vault except for those that require specific policies. It can be used as a fallback policy when no other policy matches.

transit: This policy grants access only to the transit secrets engine, which handles cryptographic functions on data in-transit. It can be used by applications or services that need to encrypt or decrypt data using Vault.

These policies allow an organization to perform useful tasks such as:

Encrypting, decrypting, and rewrapping data using the transit engine all in one policy: This policy grants access to both the transit secrets engine and the default policy, which allows performing any operation on any secret in Vault.

Creating a transit encryption key for encrypting, decrypting, and rewrapping encrypted data: This policy grants access only to the transit secrets engine and its associated keys, which are used for encrypting and decrypting data in transit using AES-GCM with a 256-bit AES key or other supported key types.

Separating permissions allowed on actions associated with the transit secret engine: This policy grants access only to specific actions

related to the transit secrets engine, such as creating keys or wrapping requests. It does not grant access to other operations or secrets in Vault.

**NEW QUESTION 35**

Which of the following describes usage of an identity group?
* Limit the policies that would otherwise apply to an entity in the group
* When they want to revoke the credentials for a whole set of entities simultaneously
* Audit token usage
* Consistently apply the same set of policies to a collection of entities

An identity group is a collection of entities that share some common attributes. An identity group can have one or more policies attached to it, which are inherited by all the members of the group. An identity group can also have subgroups, which can further refine the policies and attributes for a subset of entities.

One of the use cases of an identity group is to consistently apply the same set of policies to a collection of entities. For example, an organization may have different teams or departments, such as engineering, sales, or marketing. Each team may have its own identity group, with policies that grant access to the secrets and resources that are relevant to their work. By creating an identity group for each team, the organization can ensure that the entities belonging to each team have the same level of access and permissions, regardless of which authentication method they use to log in to Vault. Reference: Identity: entities and groups | Vault | HashiCorp Developer, vault_identity_group | Resources | hashicorp/vault | Terraform | Terraform Registry

**NEW QUESTION 36**

When using Integrated Storage, which of the following should you do to recover from possible data loss?
* Failover to a standby node
* Use snapshot
* Use audit logs
* Use server logs

Integrated Storage is a Raft-based storage backend that allows Vault to store its data internally without relying on an external storage system. It also enables Vault to run in high availability mode with automatic leader election and failover. However, Integrated Storage is not immune to data loss or corruption due to hardware failures, network partitions, or human errors. Therefore, it is recommended to use the snapshot feature to backup and restore the Vault data periodically or on demand. A snapshot is a point-in-time capture of the entire Vault data, including the encrypted secrets, the configuration, and the metadata. Snapshots can be taken and restored using the vault operator raft snapshot command or the sys/storage/raft/snapshot API endpoint. Snapshots are encrypted and can only be restored with a quorum of unseal keys or recovery keys. Snapshots are also portable and can be used to migrate data between different Vault clusters or storage backends. Reference:
https://developer.hashicorp.com/vault/docs/concepts/integrated-storage1,
https://developer.hashicorp.com/vault/docs/commands/operator/raft/snapshot2,
https://developer.hashicorp.com/vault/api-docs/system/storage/raft/snapshot3

**NEW QUESTION 37**

When an auth method is disabled all users authenticated via that method lose access.
* True
* False

The statement is true. When an auth method is disabled, all users authenticated via that method lose access. This is because the tokens issued by the auth method are automatically revoked when the auth method is disabled. This prevents the users from performing any operation in Vault using the revoked tokens. To regain access, the users have to authenticate again using a different auth method that is enabled and has the appropriate policies attached. Reference: Auth Methods | Vault | HashiCorp Developer, auth disable – Command | Vault | HashiCorp Developer

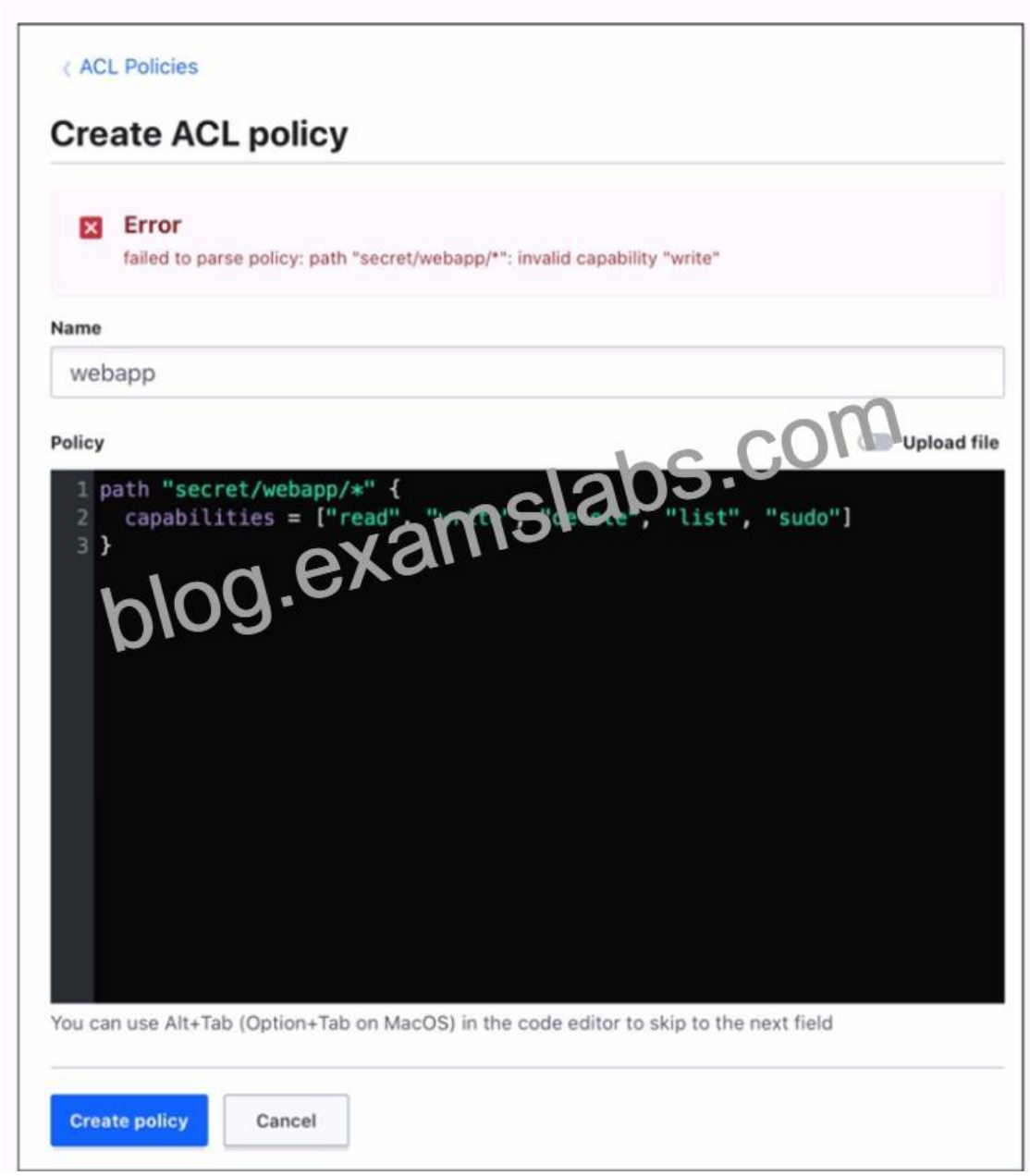
**NEW QUESTION 38**

Which Vault secret engine may be used to build your own internal certificate authority?
* Transit
* PKI
* PostgreSQL
* Generic

The Vault secret engine that can be used to build your own internal certificate authority is the PKI secret engine. The PKI secret engine generates dynamic X.509 certificates on-demand, without requiring manual processes of generating private keys and CSRs, submitting to a CA, and waiting for verification and signing. The PKI secret engine can act as a root CA or an intermediate CA, and can issue certificates for various purposes, such as TLS, code signing, email encryption, etc. The PKI secret engine can also manage the certificate lifecycle, such as rotation, revocation, renewal, and CRL generation. The PKI secret engine can also integrate with external CAs, such as Venafi or Entrust, to delegate the certificate issuance and management. Reference: PKI – Secrets Engines | Vault | HashiCorp Developer, Build Your Own Certificate Authority (CA) | Vault – HashiCorp Learn

**NEW QUESTION 39**

When creating a policy, an error was thrown:




Which statement describes the fix for this issue?

* Replace write with create in the capabilities list
* You cannot have a wildcard (” * “) in the path
* sudo is not a capability

The error was thrown because the policy code contains an invalid capability, “write”. The valid capabilities for a policy are “create”, “read”, “update”, “delete”, “list”, and “sudo”. The “write” capability is not recognized by Vault and should be replaced with “create”, which allows creating new secrets or overwriting existing ones. The other statements are not correct, because the wildcard (*) and the sudo capability are both valid in a policy. The wildcard matches any number of characters within a path segment, and the sudo capability allows performing certain operations that require root privileges.

Reference:

[Policy Syntax | Vault | HashiCorp Developer]

[Policy Syntax | Vault | HashiCorp Developer]

**NEW QUESTION 40**

A user issues the following cURL command to encrypt data using the transit engine and the Vault AP:

```
curl \
--header "X-Vault-Token: c4f280f6-fdb2-18eb-89d3-589e2e834edb" \
--request POST \<
--data @payload.json \
http://127.0.0.1:8200/v1/transit/encrypt/my-key
```



Which payload.json file has the correct contents?

*
```
{
  "plaintext": "dGhlIHF1aWNrIGJyb3duIGZveA=="
}
```

*
```
{
  "ciphertext": "vault:v1:abcdefgh"
}
```

*
```
{
  "data": {
    "plaintext": "dGhlIHF1aWNrIGJyb3duIGZveA=="
  }
}
```

*
```
{
  "data": {
    "ciphertext": "vault:v1:abcdefgh"
  }
}
```

The payload.json file that has the correct contents is C. This file contains a JSON object with a single key, “plaintext”, and a value that is the base64-encoded string of the data to be encrypted. This is the format that the Vault

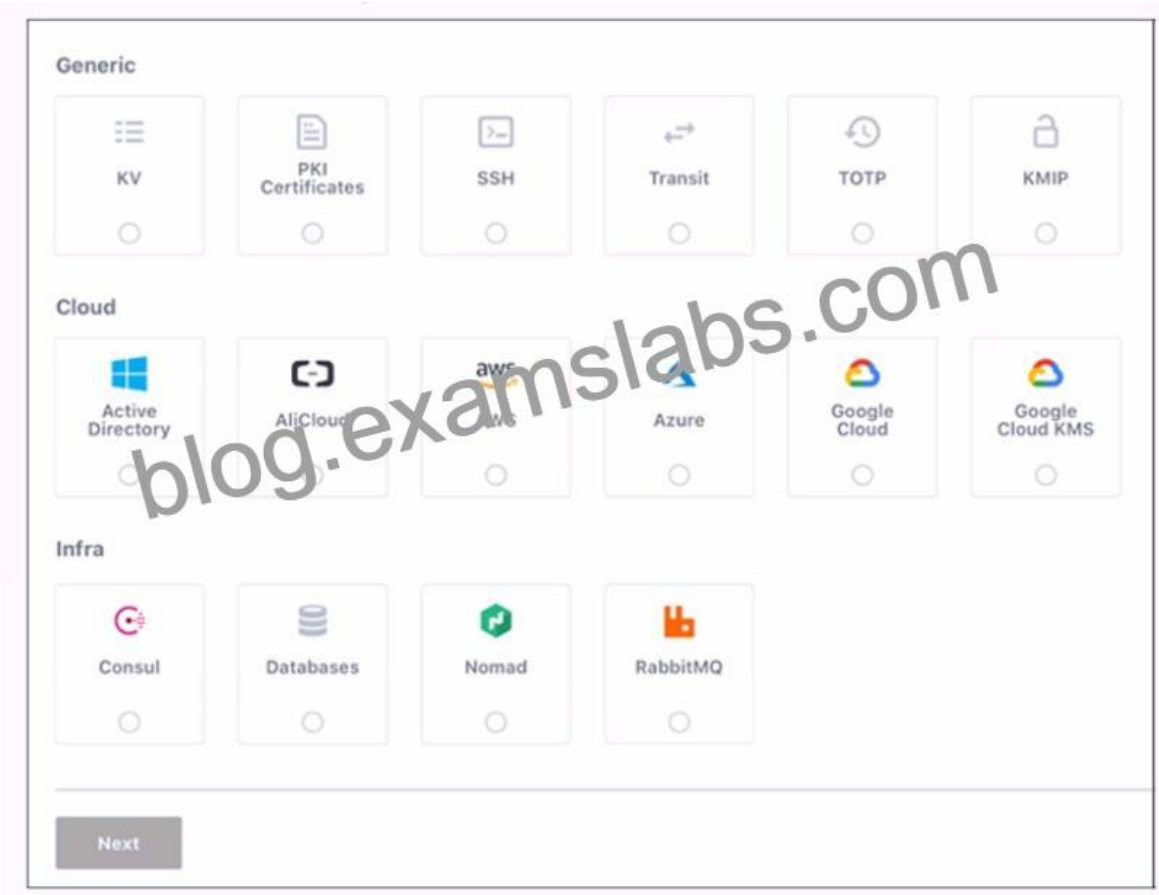
API expects for the transit encrypt endpoint1. The other files are not correct because they either have the wrong key name, the wrong value format, or the wrong JSON syntax.

Reference:

Encrypt Data – Transit Secrets Engine | Vault | HashiCorp Developer

**NEW QUESTION 41**

Use this screenshot to answer the question below:






When are you shown these options in the GUI?
* Enabling policies
* Enabling authentication engines
* Enabling secret engines
* Enabling authentication methods

This screenshot is shown when you are enabling authentication methods in the GUI. Authentication methods are the ways users and applications authenticate with Vault. Vault supports many different authentication methods, including username and password, GitHub, and more. You can enable one or more authentication methods from the grid of options, which are divided into three categories: Generic, Cloud, and Infra. Each option has a name, a description, and a logo. You can also enable authentication methods using the Vault CLI or API.

Enabling policies, authentication engines, and secret engines are different tasks that are not related to this screenshot. Policies are

rules that govern the access to Vault resources, such as secrets, authentication methods, and audit devices. Authentication engines are components of Vault that perform authentication and assign policies to authenticated entities. Secret engines are components of Vault that store, generate, or encrypt data. These tasks have different GUI pages and options than the screenshot.

Reference:

[Authentication | Vault | HashiCorp Developer]

[Policies | Vault | HashiCorp Developer]

[Authentication | Vault | HashiCorp Developer]

[Secrets Engines | Vault | HashiCorp Developer]

**NEW QUESTION 42**

Which of these are a benefit of using the Vault Agent?
* Vault Agent allows for centralized configuration of application secrets engines
* Vault Agent will auto-discover which authentication mechanism to use
* Vault Agent will enforce minimum levels of encryption an application can use
* Vault Agent will manage the lifecycle of cached tokens and leases automatically
Vault Agent is a client daemon that provides the following features:

Auto-Auth – Automatically authenticate to Vault and manage the token renewal process for locally-retrieved dynamic secrets.

API Proxy – Allows Vault Agent to act as a proxy for Vault’s API, optionally using (or forcing the use of) the Auto-Auth token.

Caching – Allows client-side caching of responses containing newly created tokens and responses containing leased secrets generated off of these newly created tokens. The agent also manages the renewals of the cached tokens and leases.

Templating – Allows rendering of user-supplied templates by Vault Agent, using the token generated by the Auto-Auth step.

Process Supervisor Mode – Runs a child process with Vault secrets injected as environment variables.

One of the benefits of using the Vault Agent is that it will manage the lifecycle of cached tokens and leases automatically. This means that the agent will handle the token renewal and revocation logic, as well as the lease renewal and revocation logic for the secrets that are cached by the agent. This reduces the burden on the application developers and operators, and ensures that the tokens and secrets are always valid and up-to-date. Reference: Vault Agent | Vault | HashiCorp Developer, Caching – Vault Agent | Vault | HashiCorp Developer

**NEW QUESTION 43**

Where can you set the Vault seal configuration? Choose two correct answers.
* Cloud Provider KMS
* Vault CLI
* Vault configuration file
* Environment variables
* Vault API

The Vault seal configuration can be set in two ways: through the Vault configuration file or through environment variables. The Vault configuration file is a text file that contains the settings and options for Vault, such as the storage backend, the listener, the telemetry, and the seal. The seal stanza in the configuration file specifies the seal type and the parameters to use for additional data protection, such as using HSM or Cloud KMS solutions to encrypt and decrypt the root key. The seal configuration can also be set through environment variables, which will take precedence over the values in the configuration file. The environment variables are prefixed with VAULT_SEAL_ and followed by the seal type and the parameter name. For example, VAULT_SEAL_AWSKMS_REGION sets the region for the AWS KMS seal. Reference: Seals – Configuration | Vault | HashiCorp Developer, Environment Variables | Vault | HashiCorp Developer

**NEW QUESTION 44**

When looking at Vault token details, which key helps you find the paths the token is able to access?
* Meta
* Path
* Policies
* Accessor

When looking at Vault token details, the policies key helps you find the paths the token is able to access. Policies are a declarative way to grant or forbid access to certain paths and operations in Vault. Policies are written in HCL or JSON and are attached to tokens by name. Policies are deny by default, so an empty policy grants no permission in the system. A token can have one or more policies associated with it, and the effective policy is the union of all the individual policies. You can view the token details by using the vault token lookup command or the auth/token/lookup API endpoint. The output will show the policies key with a list of policy names that are attached to the token. You can also view the contents of a policy by using the vault policy read command or the sys/policy API endpoint. The output will show the rules key with the HCL or JSON representation of the policy. The rules will specify the paths and the capabilities (such as create, read, update, delete, list, etc.) that the policy allows or denies. Reference: https://developer.hashicorp.com/vault/docs/concepts/policies4, https://developer.hashicorp.com/vault/docs/commands/token/lookup5, https://developer.hashicorp.com/vault/api-docs/auth/token#lookup-a-token6, https://developer.hashicorp.com/vault/docs/commands/policy/read7, https://developer.hashicorp.com/vault/api-docs/system/policy8

**NEW QUESTION 45**

You are performing a high number of authentications in a short amount of time. You’re experiencing slow throughput for token generation. How would you solve this problem?
* Increase the time-to-live on service tokens
* Implement batch tokens
* Establish a rate limit quota
* Reduce the number of policies attached to the tokens

Batch tokens are a type of tokens that are not persisted in Vault’s storage backend, but are encrypted blobs that carry enough information to perform Vault actions. Batch tokens are extremely lightweight and scalable, and can improve the throughput for token generation. Batch tokens are suitable for high-volume and ephemeral workloads, such as containers or serverless functions, that require short-lived and non-renewable tokens. Batch tokens can be created by using the -type=batch flag in the vault token create command, or by configuring the token_type parameter in the auth method’s role or mount options. Batch tokens have some limitations compared to service tokens, such as the lack of renewal, revocation, listing, accessor, and cubbyhole features. Therefore, batch tokens should be used with caution and only when the trade-offs are acceptable. Reference: https://developer.hashicorp.com/vault/tutorials/tokens/batch-tokens1, https://developer.hashicorp.com/vault/docs/commands/token/create2, https://developer.hashicorp.com/vault/docs/concepts/tokens#token-types3

**NEW QUESTION 46**

What environment variable overrides the CLI’s default Vault server address?
* VAULT_ADDR
* VAULT_HTTP_ADORESS
* VAULT_ADDRESS
* VAULT _HTTPS_ ADDRESS

The environment variable VAULT_ADDR overrides the CLI’s default Vault server address. The VAULT_ADDR environment variable specifies the address of the Vault server that is used to communicate with Vault from other applications or processes. By setting this variable, you can avoid hard-coding the Vault server address in your code or configuration files, and you can also use different addresses for different environments or scenarios. For example, you can use a local development server for testing purposes, and a production server for deploying your application. Reference: Commands (CLI) | Vault | HashiCorp Developer, Vault Agent – secrets as environment variables | Vault | HashiCorp Developer

**NEW QUESTION 47**

The vault lease renew command increments the lease time from:
* The current time
* The end of the lease

The vault lease renew command increments the lease time from the current time, not the end of the lease. This means that the user can request a specific amount of time they want remaining on the lease, termed the increment. This is not an increment at the end of the current TTL; it is an increment from the current time. For example, vault lease renew -increment=3600 my-lease-id would request that the TTL of the lease be adjusted to 1 hour (3600 seconds) from now. Having the increment be rooted at the current time instead of the end of the lease makes it easy for users to reduce the length of leases if they don’t actually need credentials for the full possible lease period, allowing those credentials to expire sooner and resources to be cleaned up earlier. The requested increment is completely advisory. The backend in charge of the secret can choose to completely ignore it1. Reference:

Lease, Renew, and Revoke | Vault | HashiCorp Developer

**NEW QUESTION 48**

Which of the following statements are true about Vault policies? Choose two correct answers.
* The default policy can not be modified
* You must use YAML to define policies
* Policies provide a declarative way to grant or forbid access to certain paths and operations in Vault
* Vault must be restarted in order for a policy change to take an effect
* Policies deny by default (empty policy grants no permission)

Vault policies are written in HCL or JSON format and are attached to tokens or roles by name. Policies define the permissions and restrictions for accessing and performing operations on certain paths and secrets in Vault. Policies are deny by default, which means that an empty policy grants no permission in the system, and any request that is not explicitly allowed by a policy is implicitly denied1. Some of the features and benefits of Vault policies are:

Policies are path-based, which means that they match the request path to a set of rules that specify the allowed or denied capabilities, such as create, read, update, delete, list, sudo, etc2.

Policies are additive, which means that if a token or a role has multiple policies attached, the effective policy is the union of all the individual policies. The most permissive capability is granted if there is a conflict3.

Policies can use glob patterns, such as * and +, to match multiple paths or segments with a single rule. For example, path “secret/*” matches any path starting with secret/, and path “secret/+/config” matches any path with two segments after secret/ and ending with config4.

Policies can use templating to interpolate certain values into the rules, such as identity information, time, randomness, etc. For example, path “secret/{{identity.entity.id}}/*” matches any path starting with secret/ followed by the entity ID of the requester5.

Policies can be managed by using the vault policy commands or the sys/policy API endpoints. You can write, read, list, and delete policies by using these interfaces6.

The default policy is a built-in policy that is attached to all tokens by default and cannot be deleted. However, the default policy can be modified by using the vault policy write command or the sys/policy API endpoint. The default policy provides common permissions for tokens, such as renewing themselves, looking up their own information, creating and managing response-wrapping tokens, etc7.

You do not have to use YAML to define policies, as Vault supports both HCL and JSON formats. HCL is a human-friendly configuration language that is also JSON compatible, which means that JSON can be used as a valid input for policies as well8.

Vault does not need to be restarted in order for a policy change to take effect, as policies are stored and evaluated in memory. Any change to a policy is immediately reflected in the system, and any token or role that has that policy attached will be affected by the change.

**NEW QUESTION 49**

Vault supports which type of configuration for source limited token?
* Cloud-bound tokens
* Domain-bound tokens
* CIDR-bound tokens
* Certificate-bound tokens

Vault supports CIDR-bound tokens, which are tokens that can only be used from a specific set of IP addresses or network ranges. This is a way to limit the scope and exposure of a token in case it is compromised or leaked. CIDR-bound tokens can be created by specifying the bound_cidr_list parameter when creating or updating a token role, or by using the -bound-cidr option when creating a token using the vault token create command. CIDR-bound tokens can also be created by some auth methods, such as AWS or Kubernetes, that can automatically bind the tokens to the source IP or network of the client. Reference: Token – Auth Methods | Vault | HashiCorp Developer, vault token create – Command | Vault | HashiCorp Developer

**NEW QUESTION 50**

Your DevOps team would like to provision VMs in GCP via a CICD pipeline. They would like to integrate Vault to protect the credentials used by the tool. Which secrets engine would you recommend?
* Google Cloud Secrets Engine
* Identity secrets engine
* Key/Value secrets engine version 2
* SSH secrets engine

The Google Cloud Secrets Engine is the best option for the DevOps team to provision VMs in GCP via a CICD pipeline and integrate Vault to protect the credentials used by the tool. The Google Cloud Secrets Engine can dynamically generate GCP service account keys or OAuth tokens based on IAM policies, which can be used to authenticate and authorize the CICD tool to access GCP resources. The credentials are automatically revoked when they are no longer used or when the lease expires, ensuring that the credentials are short-lived and secure. The DevOps team can configure rolesets or static accounts in Vault to define the scope and permissions of the credentials, and use the Vault API or CLI to request credentials on demand. The Google Cloud Secrets Engine also supports generating access tokens for impersonated service accounts, which can be useful for delegating access to other service accounts without storing or managing their keys1.

The Identity Secrets Engine is not a good option for this use case, because it does not generate GCP credentials, but rather generates identity tokens that can be used to access other Vault secrets engines or namespaces2. The Key/Value Secrets Engine version 2 is also not a good option, because it does not generate dynamic credentials, but rather stores and manages static secrets that the user provides3. The SSH Secrets Engine is not a good option either, because it does not generate GCP credentials, but rather generates SSH keys or OTPs that can be used to access remote hosts via SSH4.

Reference:

Google Cloud – Secrets Engines | Vault | HashiCorp Developer

Identity – Secrets Engines | Vault | HashiCorp Developer

KV – Secrets Engines | Vault | HashiCorp Developer

SSH – Secrets Engines | Vault | HashiCorp Developer

**NEW QUESTION 51**

You have a 2GB Base64 binary large object (blob) that needs to be encrypted. Which of the following best describes the transit secrets engine?
* A data key encrypts the blob locally, and the same key decrypts the blob locally.
* To process such a large blob. Vault will temporarily store it in the storage backend.
* Vault will store the blob permanently. Be sure to run Vault on a compute optimized machine
* The transit engine is not a good solution for binaries of this size.

The transit secrets engine is not a good solution for binaries of this size, because it is designed to handle cryptographic functions on data in-transit, not data at-rest. The transit secrets engine does not store any data sent to it, so it would require sending the entire 2GB blob to Vault for encryption or decryption, which would be inefficient and impractical. A better solution would be to use the transit secrets engine to generate a data key, which is a high-entropy key that can be used to encrypt or decrypt data locally. The data key can be returned in plaintext or wrapped by another key, depending on the use case. This way, the transit secrets engine only handles the encryption or decryption of the data key, not the data itself, and the data can be stored in any primary data store.
Reference: Transit – Secrets Engines | Vault | HashiCorp Developer, Encryption as a service: transit secrets engine | Vault | HashiCorp Developer

**NEW QUESTION 52**

How would you describe the value of using the Vault transit secrets engine?
* Vault has an API that can be programmatically consumed by applications
* The transit secrets engine ensures encryption in-transit and at-rest is enforced enterprise wide
* Encryption for application data is best handled by a storage system or database engine, while storing encryption keys in Vault
* The transit secrets engine relieves the burden of proper encryption/decryption from application developers and pushes the burden onto the operators of Vault

The transit secrets engine relieves the burden of proper encryption/decryption from application developers and pushes the burden onto the operators of Vault. The transit secrets engine provides encryption as a service, which means that it performs cryptographic operations on data in-transit without storing any data. This allows developers to delegate the responsibility of managing encryption keys and algorithms to Vault operators, who can define and enforce policies on the transit secrets engine. This way, developers can focus on their application logic and data, while Vault handles the encryption and decryption of data in a secure and scalable manner.
Reference: Transit – Secrets Engines | Vault | HashiCorp Developer, Encryption as a service: transit secrets engine | Vault | HashiCorp Developer

**Get Top-Rated HashiCorp Vault-Associate Exam Dumps Now:**
https://www.examslabs.com/HashiCorp/HashiCorp-Security-Automation/best-Vault-Associate-exam-dumps.html]