DUMPSBOSS.

HashiCorp Certified: Vault Associate (003)Exam

HashiCorp HCVA0-003

Version Demo

Total Demo Questions: 34

Total Premium Questions: 348

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Vault Architecture	59
Topic 2, Vault Deployment	7
Topic 3, Vault Configuration	17
Topic 4, Vault Security	103
Topic 5, Vault Operations	81
Topic 6, Vault Integration	81
Total	348

QUESTION NO: 1

From the options below, select the benefits of using a batch token over a service token (select four).

A. Often used for ephemeral, high-performance workloads

B. Can be a root token

C. Can be used on performance replication clusters (if orphan)

D. Has accessors

E. Lightweight and scalable

F. No storage cost for token creation

ANSWER: A C E F

Explanation:

Batch tokens are intended for situations where Vault needs to issue tokens at very high volume without the storage and write overhead associated with service-token creation. “Often used for ephemeral, high-performance workloads” is correct because batch tokens are designed for workloads such as short-lived application jobs where lightweight authentication is more important than token-management features. “Lightweight and scalable” is correct because a batch token is a cryptographically protected blob containing its authentication information, rather than a token whose full state must be persisted and indexed like a service token.

“No storage cost for token creation” is also a key benefit: Vault does not store batch-token data, which substantially reduces the storage burden and replication overhead of issuing large numbers of tokens. Finally, “Can be used on performance replication clusters (if orphan)” reflects a replication benefit. An orphan batch token has no parent-token dependency that must be resolved from the originating cluster, enabling its use across performance replication clusters. These characteristics make batch tokens particularly suitable when an application only needs policy-based access for a limited lifetime and does not require stateful token-management capabilities. See the Vault token concepts documentation and the Vault Enterprise replication documentation.

QUESTION NO: 2

Which is true about Vault authentication responses when using the Vault API?

A. The Vault API cannot be used for authentication.

B. The returned tokens are not needed as all API endpoints are unauthenticated.

C. The returned tokens should be deleted to avoid any use in future requests.

D. The returned token must be passed in the request header.

ANSWER: D

Explanation:

When a client successfully authenticates to Vault through an API authentication endpoint, Vault includes an `auth` object in its response. This object contains a client token and associated details such as its policies, lease duration, renewability, and token type. The client must retain that token and present it on subsequent requests to endpoints that require authorization. The standard method is to send it in the `X-Vault-Token` HTTP request header, for example `X-Vault-Token: s.example`. Vault also supports the standard `Authorization: Bearer <token>` header format. This explicit token handling is fundamental to direct HTTP API use: Vault returns credentials at authentication time, and the API caller supplies those credentials with later requests. Token lifecycle controls, including renewal and revocation, can then be applied as appropriate to the workload. HashiCorp documents the client-token response structure and the required token headers in its Token authentication API documentation and describes how Vault clients authenticate and use tokens in the authentication concepts documentation.

QUESTION NO: 3

During a service outage, you must ensure all current tokens and leases are copied to another Vault cluster for failover so applications don't need to authenticate. How can you accomplish this?

A. Have Vault write all the tokens and leases to a file so you have a second copy of them

B. Configure all applications to use the auto-auth feature of the Vault Agent

C. Configure Disaster Recovery replication and promote the secondary cluster during an outage

D. Replicate to another cluster using Performance Replication and promote the secondary cluster during an outage

ANSWER: C

Explanation:

Configure Disaster Recovery replication and promote the secondary cluster during an outage. Vault Disaster Recovery (DR) replication is designed for failover: a DR primary continuously replicates its data to DR secondary clusters so that a secondary can be promoted if the primary cluster becomes unavailable. The replicated state includes the information needed to preserve the validity and management of existing client tokens and leases, allowing applications to continue using credentials issued before the outage rather than requiring a full reauthentication workflow solely because the active cluster changed.

During an outage, an operator promotes the DR secondary, making it the active cluster that serves client requests. This is specifically the resilience model required when the goal is continuity of the Vault security state, including token and lease data. DR replication is an Enterprise feature and should be configured, tested, and operated with an established failover procedure, including appropriate client routing or DNS changes so applications reach the promoted cluster. HashiCorp documents the intended DR use case and promotion workflow in its Disaster Recovery Replication documentation, with broader operational context in the Vault Replication overview.

QUESTION NO: 4

Which of the following statements are true about HCP Vault Dedicated? (Select three)

A. Provides 100% feature parity compared to Vault self-managed clusters

B. Helps reduce operational overhead for organizations with push-button deployment and fully managed upgrades

C. Increases reliability and ease of use so you can onboard applications and teams easily

D. Increases security across clouds and machines through a single interface

ANSWER: B C D

Explanation:

HCP Vault Dedicated reduces operational overhead through its managed-service model. HashiCorp provisions and operates the Vault Enterprise deployment, including managed upgrades and backups, allowing platform and security teams to devote more time to adopting Vault and integrating it with applications rather than maintaining its underlying infrastructure. This operational model is a core benefit for organizations that need enterprise Vault capabilities without assuming all cluster-administration responsibilities.

HCP Vault Dedicated is also designed to simplify cloud security automation and accelerate adoption. Its managed delivery and HashiCorp's operational expertise make it easier to get started and support onboarding applications and teams. The service provides the reliability benefits of an offering operated by HashiCorp, which has extensive experience operating Vault Enterprise environments.

Finally, Vault provides centralized control of sensitive data and access across infrastructure environments. HCP Vault Dedicated enables organizations to secure secrets and manage access through a unified Vault interface, supporting consistent security controls across clouds and machines. This makes it well suited to organizations managing distributed

applications and infrastructure while seeking centralized policy-based access management. See HashiCorp's HCP Vault Dedicated documentation and the HashiCorp Vault product overview.

QUESTION NO: 5

Which statement best explains how Vault handles data encryption?

A. Vault uses encryption to secure data at rest and in transit, using an encryption key protected by the root key.

B. Vault encrypts data using a root key stored in plain text on the server's filesystem.

C. Vault stores data in plaintext on disk but encrypts it only when transmitting it over the network.

D. Vault offloads all encryption to third-party services, so no secret data is ever processed by Vault.

ANSWER: A

Explanation:

Vault uses encryption to secure data at rest and in transit, using an encryption key protected by the root key. Vault encrypts the values it persists to its configured storage backend, so the backend holds encrypted Vault data rather than usable plaintext secrets. The encryption key material used for this protection is managed in Vault's keyring. That keyring is encrypted by the root key, creating a hierarchy in which the root key protects the keys that protect stored data. During initialization, Vault is sealed; it cannot decrypt its encrypted storage until the unseal process makes the root key available. With Shamir sealing, unseal key shares reconstruct the root key in memory; with auto-unseal, a configured external key-management system protects the root key. Vault also protects client and API communication in transit through TLS. These separate protections are fundamental to Vault's security model: encrypted storage protects persisted information, root-key protection controls access to the encryption keyring, and TLS protects data while it travels over the network. See HashiCorp's seal/unseal concepts documentation and Vault architecture documentation for the encryption hierarchy and security model.

QUESTION NO: 6

A security team is configuring a PKI role for internal service certificates. The role must restrict which DNS names can be issued and prevent certificates from being valid longer than the organization permits.

Which role parameters help meet these requirements? (Select three)

A. `allowed_domains`

B. `allow_subdomains`

C. `max_ttl`

D. `no_store`

E. `issuer_ref`

ANSWER: A B C

Explanation:

`allowed_domains` restricts the DNS domains for which the role can issue certificates. `allow_subdomains` controls whether subdomains of those allowed domains are accepted. `max_ttl` limits the maximum validity period that can be issued through the role.

`no_store` controls whether issued certificates are stored by the PKI secrets engine, rather than restricting names or certificate lifetime. `issuer_ref` selects the issuer used by the role, but does not itself constrain the requested DNS names or TTL.

QUESTION NO: 7

What is the Vault CLI command to query information about the token the client is currently using?

A. vault lookup token

B. vault token lookup

C. vault lookup self

D. vault self-lookup

ANSWER: B

Explanation:

The correct command is `vault token lookup`. The `vault token lookup` command reads metadata about a Vault token, including properties such as its token ID or accessor, policies, creation time, time-to-live, renewal status, display name, entity association, and token type. When it is run without a token argument, Vault uses the token supplied by the current client session—normally the value of `VAULT_TOKEN` or the token stored by `vault login`. This makes it the standard CLI command for inspecting the credentials currently authenticating Vault requests. The command maps to Vault's token self-lookup capability and is useful for verifying which policies and lifetime constraints apply before performing privileged operations. A token can also be explicitly supplied to the command when authorized, but omitting that argument specifically queries the currently used token. HashiCorp documents this behavior in the Vault CLI command reference: Vault token lookup command. Related token concepts, including TTLs, policies, and renewable tokens, are described in the Vault token concepts documentation.

QUESTION NO: 8

Based on the following output, what command can Steve use to determine if the KV store is configured for versioning?

text

CollapseWrapCopy

$ vault secrets list

Path Type Accessor Description

---- ---- -------- -----------

automation/ kv kv_56f991b9 Automation team for CI/CD

cloud/ kv kv_4426c541 Cloud team for static secrets

cubbyhole/ cubbyhole cubbyhole_9bd538e per-token priv secret storage

data_team/ kv kv_96d57692 Data warehouse KV for certs

identity/ identity identity_0042595e identity store

network/ kv kv_3e53aaab Network team secret storage

secret/ kv kv_d66e2adc key/value secret storage

sys/ system system_d6f218a9 system endpoints

A. vault secrets list -all

B. vault kv get automation

C. vault secrets list -detailed

D. vault kv list

ANSWER: C

Explanation:

`vault secrets list -detailed` is the appropriate command because the standard secrets-engine listing shows that a mount uses the `kv` secrets engine but does not display its mount configuration in sufficient detail to identify the KV engine version. The detailed listing includes additional mount metadata, notably the mount `Options`. For a KV version 2 mount, this output includes an option such as `version=2`, which confirms that the engine supports versioned secret values. KV version 2 retains prior versions of a secret and exposes version-aware API paths and commands, whereas KV version 1 does not provide this versioning behavior. Steve can run the command and inspect the row for a particular mount, such as `automation/`, to see whether its options identify version 2. HashiCorp documents that the KV secrets engine has distinct version 1 and version 2 behaviors, with version 2 adding secret versioning and related metadata capabilities. The Vault CLI reference also documents the `-detailed` flag for displaying detailed information about enabled secrets engines. See the KV secrets engine documentation and the vault secrets list command reference.

QUESTION NO: 9

What artifacts allow you to regenerate a root token after you have revoked it?

Pick the 2 correct responses below.

A. Access to the OS root user.

B. Policy with sudo access.

C. Initial root token.

D. Unseal keys.

E. Recovery keys.

ANSWER: D E

Explanation:

Unseal keys and **Recovery keys** are the artifacts used in Vault's controlled root-token generation procedure. A revoked root token is permanently invalidated, so generating a replacement requires the `vault operator generate-root` workflow rather than restoring or reusing the revoked token. This workflow creates a one-time-password-protected encoded token and requires multiple authorized key holders to provide key shares, enforcing a quorum-based break-glass process instead of allowing one person to create a root token alone.

For a Vault cluster using Shamir sealing, the required shares are unseal keys. Vault combines the configured threshold of those shares during the generate-root operation. For a cluster using auto-unseal, unseal keys are held or managed through the configured seal mechanism, while Vault provides recovery keys to authorize sensitive recovery operations. A threshold of recovery-key shares is therefore used to generate a root token in the auto-unseal case. The generated token can then be decoded with the OTP chosen when initialization began. This design preserves separation of duties and ensures that recovery of top-level administrative access is auditable and dependent on a quorum. See the official Vault generate-root command documentation and Vault seal and recovery key documentation.

QUESTION NO: 10

You would like to provision virtual machines (VMs) using infrastructure as code (IaC). The VMs require an OAuth token to access GCP services during provisioning. You are required to use tokens that can be generated and revoked automatically.

Which secrets engine would meet this need?

A. Identity secrets engine

B. Key/Value secrets engine version 2

C. Google Cloud secrets engine

D. SSH secrets engine

ANSWER: C

Explanation:

The Google Cloud secrets engine meets this requirement because it dynamically generates short-lived Google Cloud credentials, including OAuth 2.0 access tokens, for a configured GCP service account. A client such as an IaC workflow can request an access token from Vault immediately before provisioning or configuring a VM, then use that token to call Google Cloud APIs. This avoids placing a long-lived service-account key or manually managed token in source control, configuration files, or the provisioning environment.

Vault returns the generated credential with a lease. Its lifetime is controlled through the role and Vault lease settings, and Vault can revoke the lease when it is no longer needed. This supports automated credential lifecycle management: credentials are issued on demand, expire after a limited period, and can be revoked through Vault. The Google Cloud secrets engine is therefore appropriate when workloads need temporary OAuth access to GCP services during automation. HashiCorp documents the engine's access-token endpoint and its service-account impersonation workflow in the Google Cloud secrets engine documentation. Details for generating OAuth access tokens are available in the Generate OAuth 2.0 access token API reference.

QUESTION NO: 11

Which of the following are replication methods available in Vault Enterprise? Choose two correct answers.

A. Cluster sharding

B. Namespaces

C. Performance Replication

D. Disaster Recovery Replication

ANSWER: C D

Explanation:

Vault Enterprise provides Performance Replication and Disaster Recovery Replication as its two replication capabilities. Performance Replication is designed to scale Vault horizontally across geographic regions or separate environments. A performance primary replicates selected or eligible data to performance secondary clusters, which can serve client read and write requests locally. This allows organizations to reduce latency, distribute client traffic, and maintain regional Vault access while centrally managing replicated configuration and secrets data. Performance replication also supports path filters to control which data is replicated to individual secondary clusters.

Disaster Recovery Replication provides a standby copy of a Vault cluster for business continuity. A disaster recovery primary continuously replicates its data and state to a disaster recovery secondary. If the primary becomes unavailable because of a major outage, an authorized operator can promote the secondary, allowing it to become the active cluster and resume service using the replicated Vault state. These features are distinct Enterprise replication modes intended for workload scaling and recovery planning. See HashiCorp's Vault Enterprise Replication documentation and Performance Replication documentation.

QUESTION NO: 12

An application uses a KV version 2 secrets engine mounted at `secret/`. A deployment process reads `secret/app/config` at version 4, then prepares an update. Before it writes, another process creates version 5.

The deployment process must not overwrite version 5 with its stale update.

Which command option should the deployment process use when writing its update?

A. `vault kv put -cas=4 secret/app/config setting=new-value`

B. `vault kv put -version=4 secret/app/config setting=new-value`

C. `vault kv put -force secret/app/config setting=new-value`

D. `vault kv put -metadata-version=4 secret/app/config setting=new-value`

ANSWER: A

Explanation:

Use the `-cas=4` option with `vault kv put`. KV version 2 supports check-and-set (CAS) writes, which permit a write only when the current version matches the version expected by the client. Since the deployment process read version 4, a write using `-cas=4` succeeds only if no newer version has been written.

After another process creates version 5, the CAS condition fails and Vault rejects the stale write. The deployment process can then read the current value and decide how to merge or retry its change. A normal `vault kv put` operation would create a new version regardless of the intervening update.

QUESTION NO: 13

Vault enables the generation of dynamic credentials against many different platforms. When generating these credentials, what Vault feature is used to track the credentials?

A. namespace

B. role

C. token

D. lease_id

ANSWER: D

Explanation:

`lease_id` is the identifier Vault uses to track a dynamically generated credential throughout its lifecycle. When Vault creates a dynamic secret—such as a database username and password, cloud access keys, or other temporary credentials—it associates that secret with a lease. The lease contains lifecycle metadata, including the lease duration (TTL), whether it is renewable, and the unique lease ID. Clients and operators use this `lease_id` when interacting with the generated secret's lifecycle, including looking up lease details, renewing the lease when permitted, or revoking it before its natural expiration. This model allows Vault to centrally manage temporary access and ensure that credentials issued by a secrets engine remain valid only for their intended period. When a lease expires or is revoked, Vault instructs the relevant secrets engine to remove or invalidate the associated dynamic credential where supported. HashiCorp describes leases as the mechanism attached to dynamic secrets and service tokens, with the lease ID serving as the unique identifier for managing that issued secret. See the Vault leases documentation and the dynamic secrets overview.

QUESTION NO: 14

A web application uses Vault ' s transit secrets engine to encrypt data in-transit. If an attacker intercepts the data in transit which of the following statements are true? Choose two correct answers.

A. You can rotate the encryption key so that the attacker won't be able to decrypt the data

B. The keys can be rotated and min_decryption_version moved forward to ensure this data cannot be decrypted

C. The Vault administrator would need to seal the Vault server immediately

D. Even if the attacker was able to access the raw data, they would only have encrypted bits (TLS in transit)

ANSWER: B D

Explanation:

The keys can be rotated and `min_decryption_version` moved forward to ensure this data cannot be decrypted. Vault Transit keys are versioned: key rotation creates a new version without automatically removing the capability to decrypt ciphertext created by prior versions. Setting `min_decryption_version` to a newer version causes Vault to reject decryption requests for ciphertext that requires an older key version. This provides a deliberate cryptographic-retirement control when ciphertext encrypted under an earlier version must no longer be decryptable through Vault. HashiCorp documents this behavior in its Transit key rotation documentation.

Even if the attacker was able to access the raw data, they would only have encrypted bits (TLS in transit). The Transit secrets engine performs encryption as a service: an application submits plaintext to Vault and receives ciphertext, while key material remains managed by Vault rather than being returned to the application. Consequently, interception of properly protected encrypted traffic or ciphertext does not itself provide the plaintext or the managed encryption key. Access to Transit operations is separately governed by Vault authentication and policy authorization. The engine is designed for cryptographic operations and does not persist the application plaintext submitted for encryption. See HashiCorp's Transit secrets engine documentation.

QUESTION NO: 15

What API endpoint is used to manage secrets engines in Vault?

A. /secret-engines/

B. /sys/mounts

C. /sys/capabilities

D. /sys/kv

ANSWER: B

Explanation:

`/sys/mounts` is the Vault system API endpoint used to manage secrets engines, which Vault also calls auth methods and secrets engines "mounts" in its API structure. An administrator can use this endpoint to list enabled secrets engines, enable a new engine at a chosen mount path, inspect configuration, tune an existing mount, or disable a mount. For example, sending a request to `POST /sys/mounts/<path>` enables a secrets engine at that path, while `GET /sys/mounts` returns the currently enabled mounts and their configuration details. Access to these operations is governed by Vault policies, typically requiring capabilities such as `create`, `read`, `update`, or `delete` on the applicable `sys/mounts` path. This endpoint is part of Vault's system backend, which exposes administrative APIs under the `/sys` prefix. HashiCorp documents the complete mount-management API, including enable, disable, tune, and list operations, at Vault API: Mounts. The broader concept and operational use of mounts are also described in the Vault mounts documentation.

QUESTION NO: 16

What is the default value of the VAULT_ADDR environment variable?

A. http://127.0.0.1:8200

B. https://vault.example.com:8200

C. https://127.0.0.1:8200

D. http://vault.example.com:8200

ANSWER: C

Explanation:

The correct default value is `https://127.0.0.1:8200`. When a Vault CLI command needs to contact a Vault server and no address is supplied through the `VAULT_ADDR` environment variable or another applicable client setting, the CLI assumes this local HTTPS endpoint. The loopback address `127.0.0.1` means the client expects Vault to be reachable on the same machine, while port `8200` is Vault's standard API listener port. The `https` scheme is significant because Vault is designed to protect secrets in transit and its normal client configuration expects a TLS-secured connection. In practical deployments, administrators commonly set `VAULT_ADDR` to the DNS name and protocol of their organization's Vault service, but that is an explicit configuration change rather than the CLI default. HashiCorp documents the environment variable's default directly in the Vault CLI environment-variable reference. See the Vault CLI environment variables documentation and the Vault CLI command documentation.

QUESTION NO: 17

A Jenkins server is using the following token to access Vault. Based on the lookup shown below, what type of token is this?
$ vault token lookup hvs.FGP1A77Hxa1Sp6Pkp1yURcZB

Key Value

--- -----

accessor RnH8jtgrxBrYanizlyJ7Y8R

creation_time 1604604512

creation_ttl 24h

display_name token

entity_id n/a

expire_time 2025-11-06T14:28:32.8891566-05:00

explicit_max_ttl 0s

id hvs.FGP1A77Hxa1Sp6KRau5eNB

issue_time 2025-11-06T14:28:32.8891566-05:00

meta < nil >

num_uses 0

orphan false

path auth/token/create

period 24h

policies [admin default]

renewable true

ttl 23h59m50s

type service

A. Periodic token

B. Batch token

C. Orphaned token

D. Secondary token

ANSWER: A

Explanation:

Periodic token is correct because the token lookup output contains `period 24h`. In Vault, a periodic token is issued with a defined renewal period. Each successful renewal resets its TTL to that period—in this case, 24 hours—rather than extending the token indefinitely by adding time to its remaining lease. The output also shows `renewable true`, which confirms that Vault can renew the token, and the current `ttl` of nearly 24 hours is consistent with a newly issued or recently renewed periodic token.

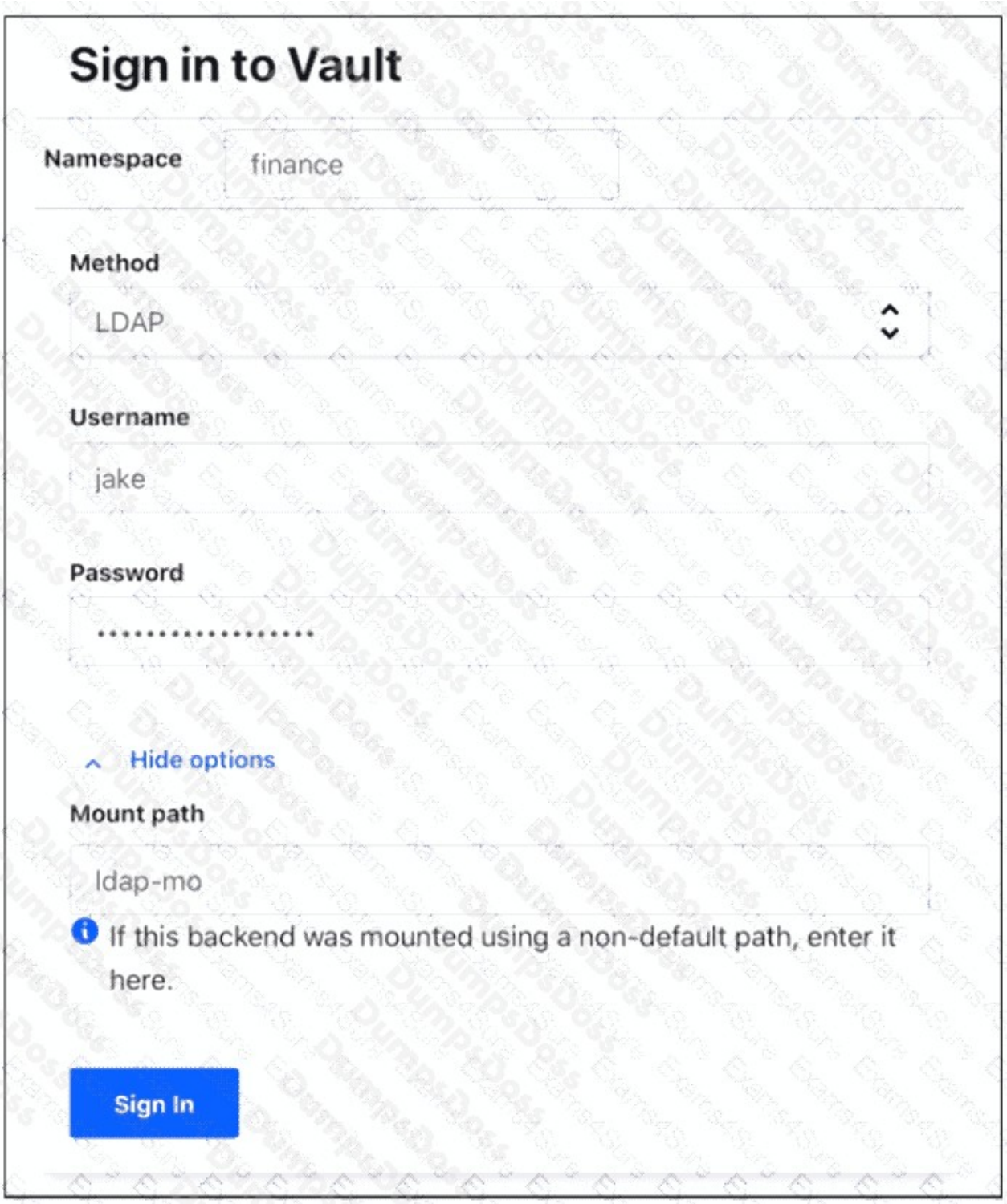
The `explicit_max_ttl` value of `0s` is also significant in this context: no explicit maximum TTL has been set for the token. Periodic tokens are commonly used by long-running automated workloads, such as a Jenkins server, because the workload can continue to authenticate with the same token as long as it renews the token before each period expires and remains authorized to do so. If renewal stops, the token expires after its current 24-hour TTL. Vault documents periodic tokens and their renewal behavior in the Vault token concepts documentation; the token lookup command reference describes the fields displayed by this command.

QUESTION NO: 18 - (SIMULATION)

Where do you define the Namespace to log into using the Vault UI?

To answer this question

Use your mouse to click on the screenshot in the location described above. An arrow indicator will mark where you have clicked. Click the " Answer " button once you have positioned the arrow to answer the question. You may need to scroll down to see the entire screenshot.



ANSWER: See the explanation for the answer

Explanation:

Click the `Namespace` field at the top of the Vault sign-in form (the field currently showing `finance`), approximately at `x=300, y=111`.

This is where the Vault UI namespace is selected/entered. The `Mount path` field is only for the non-default path of the selected authentication method.

QUESTION NO: 19

You have a CI/CD pipeline using Terraform to provision AWS resources with static privileged credentials. Your security team requests that you use Vault to limit AWS access when needed. How can you enhance this process and increase pipeline security?

A. Enable the SSH secrets engine and have Terraform generate dynamic credentials when deploying resources in AWS

B. Enable the Transit secrets engine to encrypt the AWS credentials and have Terraform retrieve these credentials when needed

C. Store the AWS credentials in the Vault KV store and use the Vault provider to obtain these credentials on each terraform apply

D. Enable the aws secrets engine and configure Terraform to dynamically generate a short-lived AWS credential on each terraform apply

ANSWER: D

Explanation:

Enable the aws secrets engine and configure Terraform to dynamically generate a short-lived AWS credential on each terraform apply. Vault's AWS secrets engine generates dynamic AWS credentials on demand rather than requiring a long-lived, highly privileged access key to be stored in the CI/CD system. Depending on the configured role, Vault can create IAM users with access keys or issue temporary STS credentials. The credentials are governed by a lease with a defined time-to-live, and Vault can revoke the associated access when the lease expires or is explicitly revoked. This implements least-privilege access by mapping the Terraform workflow to an AWS role or policy that grants only the permissions required for provisioning. It also reduces the impact of accidental credential exposure because a leaked credential is short-lived instead of remaining valid indefinitely. In a pipeline, Terraform can authenticate to Vault using an appropriate non-static authentication method, retrieve credentials from the AWS secrets engine immediately before a run, and provide them to the AWS provider through environment variables or provider configuration. This creates an auditable, centrally managed workflow for issuing and controlling AWS access. See the Vault AWS secrets engine documentation and the AWS STS credentials documentation.

QUESTION NO: 20

A deployment platform uses AppRole authentication. Each SecretID must be usable only once, must expire after 15 minutes if unused, and must be accepted only from the deployment network.

Which AppRole settings help enforce these SecretID requirements? (Select three)

A. `secret_id_num_uses`

B. `secret_id_ttl`

C. `secret_id_bound_cidrs`

D. `token_ttl`

E. `role_id`

ANSWER: A B C

Explanation:

`secret_id_num_uses` limits how many times a generated SecretID can be used to log in. Setting it to one supports single-use delivery. `secret_id_ttl` limits how long an unused SecretID remains valid, so it can enforce the 15-minute expiration requirement.

`secret_id_bound_cidrs` restricts use of SecretIDs to specified source network CIDR ranges. In contrast, `token_ttl` controls the lifetime of the Vault token issued after successful AppRole login, not the lifetime of the SecretID. A RoleID identifies the AppRole but does not itself impose single-use, expiration, or source-network restrictions on SecretIDs.

QUESTION NO: 21

All Vault instances, or clusters, include two built-in policies that are created automatically. Choose the two policies below and the correct information regarding each policy. (Select two)

A. The root policy is created automatically. This policy provides superuser privileges and cannot be deleted

B. The admin policy is created automatically. It provides administrative permissions but can be deleted if needed

C. The default policy is created automatically. This policy can be modified but not deleted

D. The default policy is created automatically. This policy cannot be modified but it can be deleted

ANSWER: A C

Explanation:

Vault automatically creates the `root` and `default` ACL policies during initialization. The root policy provides unrestricted superuser access: a token associated with it can perform any operation on any path in Vault. This policy is a core safety and recovery mechanism, so it cannot be deleted or altered. Root tokens should therefore be tightly controlled and used only when privileged administrative access is truly required.

The default policy is also built in and is automatically attached to every token unless the token is itself a root token. It provides baseline capabilities needed for ordinary token behavior, including access to token self-service endpoints such as looking up or renewing the token. Unlike the root policy, administrators may modify the default policy to tailor its baseline permissions. Vault does not permit deletion of this policy, because it is a required built-in policy. These built-in-policy characteristics are documented by HashiCorp in the Vault policies documentation and the Vault token documentation.

QUESTION NO: 22

Given the following policy, which command below would not result in a permission denied error (select two)?

path " secret/* " { capabilities = [" create " , " update "] allowed_parameters = { " student " = [" steve " , " frank " , " jamie " , " susan " , " gerry " , " damien "] } }

path " secret/apps/* " { capabilities = [" read "] }

path " secret/apps/results " { capabilities = [" deny "] }

A. vault kv put secret/apps/results student03=practice

B. vault kv put secret/apps/app01 student=bryan

C. vault kv put secret/common/results student=frank

D. vault kv get secret/apps/api_key

ANSWER: C D

Explanation:

`vault kv put secret/common/results student=frank` is authorized because `secret/common/results` matches the broad `secret/*` policy path, which grants the capabilities needed to write a previously absent secret or modify an existing one. The supplied parameter is `student`, and its value, `frank`, is explicitly included in that path's allowed-values list. Thus, the request satisfies both the capability requirement and the parameter constraint.

`vault kv get secret/apps/api_key` is authorized because its path matches `secret/apps/*`, which grants the `read` capability. A KV get operation is a read request, so the policy permits retrieving the secret at that application path. The more specific deny rule applies only to the exact `secret/apps/results` path and does not affect this request. Vault evaluates policy paths and combines matching policy rules, with explicit deny taking precedence where it applies. The allowed-parameters restriction applies to write requests governed by the create/update path and does not change the authorization of this read request.

See HashiCorp's Vault policy documentation for path capabilities and precedence, and its allowed_parameters documentation for parameter restrictions.

QUESTION NO: 23

A new Vault administrator is writing a CURL command (shown below) to retrieve a secret stored in a KV v2 secrets engine at secret/audio/soundbooth but is receiving an error. What could be the cause of the error?

$ curl \

--header " X-Vault-Token: hvs.rffHw0iXqkRo19b2cjf93DM39WjpbN3J " \

https://vault.unlimited.com:8200/v1/secret/audio/soundbooth

A. The VAULT_ADDR environment variable wasn't set, so it should be configured: export VAULT_ADDR= " https://vault.unlimited.com:8200 "

B. The request is being made on the incorrect endpoint and should be:$ curl \--header " X-Vault-Token: hvs.rffHw0iXqkRo19b2cjf93DM39WjpbN3J " \https://vault.unlimited.com:8200/v1/secret/data/audio/soundbooth

C. The user's token doesn't permit access to the Vault API, only the UI

D. The endpoint should point to v2 since this is a KV v2 secrets engine:$ curl \--header " X-Vault-Token: hvs.rffHw0iXqkRo19b2cjf93DM39WjpbN3J " \https://vault.unlimited.com:8200/v2/secret/audio/soundbooth

ANSWER: B

Explanation:

The request is being made on the incorrect endpoint and should include `data/` after the KV v2 mount path. Vault's HTTP API always begins with `/v1/`; that segment identifies the Vault API version and does not change to `/v2/` for a KV version 2 engine. Instead, KV v2 exposes versioned secret values through a distinct data endpoint. Given a KV v2 engine mounted at `secret/` and a logical secret path of `audio/soundbooth`, the read request must target `/v1/secret/data/audio/soundbooth`.

The `data/` segment is essential because KV v2 separates secret data operations from its metadata, version-history, deletion, and undelete operations. A successful read also requires that the supplied token have `read` capability for the policy path `secret/data/audio/soundbooth`, but the URL shown omits the required KV v2 data prefix before authorization against the intended secret path can occur. The appropriate command therefore uses `https://vault.unlimited.com:8200/v1/secret/data/audio/soundbooth` with the token passed in the `X-Vault-Token` header. HashiCorp documents KV v2 API paths at KV v2 API documentation and describes its versioned key/value behavior in the KV secrets engine documentation.

QUESTION NO: 24

Which two interfaces automatically assume the token for subsequent requests after successfully authenticating? (Select two)

A. CLI

B. API

C. UI

ANSWER: A C

Explanation:

CLI and UI automatically assume the authentication token for subsequent Vault requests. When a user runs `vault login`, the Vault CLI receives a client token and uses its configured token helper to persist that token locally. Later CLI commands automatically retrieve and use the stored token, so the user ordinarily does not need to supply it again. In the Vault UI, a

successful login establishes the UI session using the returned token, allowing subsequent actions performed through the browser interface to be authenticated automatically during that session. This behavior makes the interactive CLI and browser-based UI convenient for routine Vault use while retaining Vault's token-based authorization model. HashiCorp's Vault documentation explicitly distinguishes these interfaces from direct API use: following authentication, the UI and CLI automatically assume the token for later requests. The CLI token-helper behavior, including how Vault stores and retrieves a token for commands, is documented separately. See Vault token concepts and Vault CLI token helpers.

QUESTION NO: 25

What features are offered by the Vault Agent? (Select three)

A. Auditing

B. Templating

C. Auto-auth

D. Secret caching

ANSWER: B C D

Explanation:

Vault Agent is a client-side daemon designed to simplify how applications authenticate to Vault and consume secrets. **Auto-auth** is a core capability: the agent can authenticate through a configured auth method, write the resulting token to a sink, and renew the token when appropriate. This avoids requiring application code to implement Vault login and token lifecycle handling itself.

Templating lets Vault Agent retrieve secret values and render them into user-defined templates. Rendered output can be written to files, allowing applications to consume secrets through familiar configuration files, certificates, environment-style files, or other structured formats. The agent can also re-render templates as leased secrets are renewed or changed.

Secret caching enables an agent listener to cache eligible Vault responses locally. Applications can send requests to the local agent, which forwards cache misses to Vault and serves valid cached responses when available. This can reduce repeated requests to Vault and improve resilience for workloads that repeatedly obtain the same secrets. These capabilities are documented in the Vault Agent documentation and the Vault Agent caching documentation.

QUESTION NO: 26

Based on the output below, how many policies have been added to Vault?

$ vault policy list

base

default

root

web-app-1

automation-team

A. 3

B. 4

C. 1

D. 2

ANSWER: A

Explanation:

3 is correct because the policy list contains five policy names, but two of these are Vault's built-in ACL policies: `default` and `root`. Vault automatically includes the `default` policy and attaches it to all tokens unless that behavior is explicitly disabled for a token. The `root` policy is also built in and grants unrestricted access to Vault; it is associated with root tokens rather than representing a user-created policy.

After excluding those two built-in policies from the five names returned by `vault policy list`, the remaining policies are `base`, `web-app-1`, and `automation-team`. Those three names represent policies that have been added to the Vault instance. The CLI command lists both built-in and user-defined ACL policies, so interpreting its output requires distinguishing the built-in policy names from policies created for an organization's applications, teams, or baseline access controls.

HashiCorp documents the built-in `root` and `default` policies in its Policies documentation. The command behavior is also described in the vault policy list command reference.

QUESTION NO: 27

Your organization is integrating its legacy application with Vault to improve its security. However, you have discovered that the application has issues when the token changes for authentication during testing. What type of token could be used to help alleviate this issue without compromising security?

A. Periodic Service Token

B. Root Token

C. Orphan Service Token

D. Batch Token

ANSWER: A

Explanation:

Periodic Service Token is the appropriate choice because it lets the legacy application continue using the same token value while its validity is extended through renewal. A periodic token has a configured period rather than a fixed maximum lifetime. As long as an authorized process renews it before the current TTL expires, Vault extends the token's TTL by that period and the token can remain valid indefinitely. This avoids forcing an application that cannot easily handle token rotation to repeatedly receive and store a replacement token.

It also preserves Vault's security controls: the token remains governed by its attached policies, has a finite TTL at any given time, and stops being usable if it is not renewed in time. Administrators can revoke it when necessary, and renewal requires appropriate access to Vault. For a legacy integration, this supports a gradual modernization path: use a tightly scoped policy and a short, appropriate renewal period, then later migrate toward an authentication workflow that obtains and rotates tokens automatically. HashiCorp documents periodic tokens and their renewable behavior in the Vault tokens documentation and describes token renewal in the Token API documentation.

QUESTION NO: 28

By default, what methods of authentication does Vault support? (Select four)

A. SSH

B. Kubernetes

C. VMware

D. LDAP

E. AppRole

F. JWT

ANSWER: B D E F

Explanation:

Vault includes several built-in authentication method plugins that can be enabled at an auth mount to verify a caller's identity and associate that identity with Vault policies. Kubernetes supports workload authentication using Kubernetes service-account JWTs, allowing applications running in a Kubernetes cluster to obtain Vault tokens without distributing static Vault credentials. LDAP authenticates users against an LDAP directory and can map directory users or groups to Vault policies. AppRole is designed for machine and application authentication: a role ID and a suitably protected secret ID are exchanged for a Vault token. JWT validates JSON Web Tokens from a configured issuer and can map JWT claims to Vault identities, groups, and policies. These mechanisms are documented as Vault auth methods and are available as built-in capabilities; an operator enables and configures the chosen method at a path appropriate for the organization. The selected methods therefore cover common human-directory, application, Kubernetes-workload, and token-issuer identity patterns. See the Vault authentication methods documentation and the AppRole authentication documentation.

QUESTION NO: 29

You are building a new CI/CD pipeline which integrates with Vault. You will be building multiple targets: on premises in vSphere, and in AWS. You have already selected the AWS authentication method for the AWS targets.

Which auth method can the CI/CD tool use to authenticate with the on-premises targets?

A. AWS

B. GitHub

C. AppRole

D. Userpass

ANSWER: C

Explanation:

AppRole is the appropriate authentication method for a CI/CD tool accessing Vault from on-premises vSphere targets. Vault's AppRole auth method is specifically intended for machines, applications, and automated workflows that need to authenticate without an interactive human login or a cloud-provider-native workload identity. The CI/CD system authenticates using a RoleID, which identifies the role and associated policies, together with a SecretID, which acts as a credential and can be tightly controlled through settings such as use limits, time-to-live values, CIDR bindings, and response wrapping. This gives pipeline operators a practical way to issue narrowly scoped, short-lived Vault tokens to automation running in environments such as vSphere. AppRole also supports different SecretID delivery patterns, allowing teams to choose an approach appropriate for their bootstrap and secret-distribution controls. Once authenticated, the resulting Vault token receives only the capabilities granted by the policies attached to the AppRole. HashiCorp documents AppRole as an auth method for "machines or apps," including automated deployment and CI/CD scenarios. See the Vault AppRole authentication documentation and the Vault authentication concepts documentation.

QUESTION NO: 30

Which of the following statements are true about Vault policies? Choose two correct answers.

A. The default policy can not be modified

B. You must use YAML to define policies

C. Policies provide a declarative way to grant or forbid access to certain paths and operations in Vault

D. Vault must be restarted in order for a policy change to take an effect

E. Policies deny by default (empty policy grants no permission)

ANSWER: C E

Explanation:

“Policies provide a declarative way to grant or forbid access to certain paths and operations in Vault” is correct because Vault uses policies to authorize requests against paths. A policy contains path rules and capabilities that describe which operations, such as reading, creating, updating, deleting, listing, or performing privileged operations, a token is permitted to request. This lets administrators express access control as managed configuration rather than embedding permissions directly into applications or manually deciding access for each request. Vault evaluates the policies associated with the requesting token whenever it handles an API request.

“Policies deny by default (empty policy grants no permission)” is also correct. Vault follows a default-deny authorization model: a request must be explicitly granted an applicable capability by policy before it can succeed. Therefore, an empty policy does not authorize access to any path. This model supports least-privilege access by ensuring that permissions arise only from intentionally written policy rules. When multiple policies are attached to a token, their granted permissions are combined, but the absence of an applicable allow rule still results in denial. See the Vault policy concepts documentation and the Vault identity and policy documentation.

QUESTION NO: 31

Assuming default configurations, which of the following operations require a threshold of key shares to perform? (Select three)

A. Rotating the Vault encryption key to adhere to internal security policies

B. Unsealing Vault after a scheduled maintenance to install patches

C. Generating a new root token as a break-glass procedure

D. Creating a new set of unseal keys due to an employee leaving the organization

ANSWER: B C D

Explanation:

With Vault’s default Shamir seal configuration, the master key is split into a configured number of unseal key shares, and the configured threshold is required to reconstruct it. Therefore, unsealing Vault after maintenance requires enough current unseal key shares to bring the server back to an unsealed state. Vault also protects root-token generation with a multi-party authorization process: generating a root token requires the threshold number of unseal key shares (or the equivalent authorization mechanism when using a different seal model). This prevents a single key holder from independently creating a highly privileged token. Finally, changing the Shamir key set through a rekey operation requires the threshold of the current unseal key shares. Rekeying is the appropriate action when key holders change, such as after an employee leaves, because it produces a replacement set of unseal shares and can change the threshold. HashiCorp documents both the threshold-based unseal process and the requirement for existing key shares during rekey and root-token generation. See the Vault seal documentation and the vault operator rekey command reference.

QUESTION NO: 32

A Vault server is initialized using Shamir's Secret Sharing with five key shares and an unseal threshold of three.

Which statements are true? (Select three)

A. Any three distinct unseal key shares can unseal Vault.

B. The initial root token is highly privileged and must be protected carefully.

C. All five key shares are required to unseal Vault.

D. A holder can use one unseal key share as a token to read secrets.

E. Vault can still be unsealed if two of the five shares are unavailable.

ANSWER: A B E

Explanation:

Any three distinct unseal key shares are sufficient to meet the configured threshold and unseal Vault. Because the threshold is three, loss of up to two of the five shares does not prevent unsealing, provided the remaining three shares are available.

Initialization also returns an initial root token. It is highly privileged and should be protected carefully and used only for initial administrative setup or controlled emergency operations. Individual unseal shares are not client authentication credentials and cannot be used in place of a Vault token to read secrets.

QUESTION NO: 33

You have TBs of data encrypted by Vault stored in a database and are worried about Vault becoming unavailable and not being able to decrypt the data. Is it possible to export the encryption key to store it somewhere else in the event Vault becomes unavailable?

A. Yes, as long as the key was configured to be exportable when it was created

B. No, you cannot export the encryption key from Vault

ANSWER: A

Explanation:

Yes, as long as the key was configured to be exportable when it was created. Vault's Transit secrets engine normally keeps encryption key material protected within Vault and does not expose it. However, a Transit key can be created with its `exportable` attribute enabled. This explicitly permits authorized clients to export the key material through the Transit key-export endpoint, including a specified key version. The exported material can then be protected and retained according to an organization's recovery and key-management procedures.

The exportable setting is a deliberate security decision made at key creation time. It cannot be enabled later for an existing non-exportable key, because doing so would weaken the original protection boundary selected for that key. Access to the export operation must also be tightly controlled with Vault policies, since anyone able to retrieve plaintext key material can potentially decrypt data independently of Vault. For resilience, teams should additionally plan for highly available Vault deployment, backups or snapshots appropriate to their storage backend, tested recovery procedures, and secure handling of any exported key material. HashiCorp documents the Transit key configuration and export operation at Transit API: exportable key configuration and Transit API: export key.

QUESTION NO: 34

What is true about the output of the following command (select three)?

A. The admin never sees all the unseal keys and cannot unseal Vault by themselves

B. All three users, Jane/John/Student01, will receive all unseal keys and can unseal Vault

C. The admin will receive the unseal keys and be able to unseal Vault themselves

D. The keys will be returned encrypted

E. Each individual can only decrypt their own unseal key using their private PGP key

ANSWER: A D E

Explanation:

When Vault is initialized with three key shares, a threshold of two, and a PGP public key supplied for each recipient, Vault uses Shamir's Secret Sharing to create three distinct unseal-key shares. Any two decrypted shares are required to unseal

Vault. The initializer's command output contains each share encrypted to its corresponding supplied PGP public key rather than displaying the share in plaintext. This allows the encrypted output to be safely delivered to the intended key holders.

Consequently, the administrator does not see all usable plaintext unseal shares and cannot unseal Vault alone when they do not possess the recipients' PGP private keys. Each recipient uses the private key corresponding to the public key supplied at initialization to decrypt that recipient's own unseal-key share. After decryption, the required threshold of key holders can provide their shares to complete the unseal process. HashiCorp documents that the `-pgp-keys` parameter encrypts generated unseal keys with the provided PGP keys, and that the matching private key is required to decrypt each resulting value. See the Vault operator init command documentation and the Vault seal and unseal concepts documentation.