DUMPSBOSS.

Developing Applications using Cisco Core Platforms and APIs (DEVCOR)

Cisco 350-901

Version Demo

Total Demo Questions: 64

Total Premium Questions: 641

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Software Development and Design	81
Topic 2, Understanding and Using APIs	200
Topic 3, Cisco Platforms and Application Development	97
Topic 4, Application Deployment and Security	176
Topic 5, Infrastructure and Automation	86
Topic 6, Network Fundamentals	1
Total	641

QUESTION NO: 1

Which two security benefits are provided by OAuth 2.0 Authorization Code flow with PKCE? (Choose two.)

A. It binds the authorization request to the token request.

B. It reduces authorization-code interception attacks for public clients.

C. It encrypts access tokens that are sent to resource servers.

D. It removes the need to validate redirect URIs.

E. It allows a client to use the resource owner's password directly.

ANSWER: A B

Explanation:

It binds the authorization request to the token request and **it reduces authorization-code interception attacks for public clients**. In PKCE, the client generates a high-entropy `code_verifier` and derives a `code_challenge` that is sent with the authorization request. When exchanging the authorization code at the token endpoint, the client must provide the original verifier. The authorization server validates that the verifier corresponds to the earlier challenge.

An intercepted authorization code is therefore insufficient for an attacker to redeem the code without the verifier. This is particularly important for public clients, such as native and single-page applications, that cannot securely maintain a client secret. PKCE is defined in RFC 7636, and current OAuth security guidance recommends PKCE for authorization-code flows in RFC 9700.

QUESTION NO: 2

Which two statements describe advantages of static code analysis over unit tests? (Choose two.)

A. It checks for potential tainted data where input is not checked.

B. It enforces proper coding standards and style.

C. It performs a quick analysis of whether tests will pass or fail when run.

D. It checks for race conditions in threaded applications.

E. It estimates the performance of the code when run.

ANSWER: A B

Explanation:

Static code analysis examines source code without executing it, so it can identify categories of defects that may not be exercised by the limited inputs, execution paths, or assertions present in unit tests. “It checks for potential tainted data where input is not checked.” is an important advantage because security-focused static analysis can trace untrusted input through code and identify data-flow paths that reach sensitive operations without appropriate validation or sanitization. This supports earlier discovery of potential injection and input-handling vulnerabilities, before runtime testing is available or comprehensive.

“It enforces proper coding standards and style.” is also correct because static-analysis tools can apply coding rules consistently across an entire codebase. Automated rule enforcement helps teams detect maintainability issues, unsafe patterns, complexity concerns, and deviations from agreed conventions during development and in CI pipelines. Unit tests primarily validate expected runtime behavior for the cases they execute; static analysis provides complementary, broad source-level feedback without requiring each rule to have a dedicated test case. OWASP describes static application security testing as analysis of an application’s source, bytecode, or binaries without executing it, including identification of security weaknesses. Sonar’s documentation also describes the use of code rules to identify code-quality and security issues. See OWASP Static Application Security Testing and SonarQube rules overview.

QUESTION NO: 3

What are two steps in the OAuth2 protocol flow? (Choose two.)

A. The user is authenticated by the authorization server and granted an access token.

B. The user's original credentials are validated by the resource server and authorization is granted.

C. The client indirectly requests authorization from the resource owner through the authorization server.

D. The client requests an access token by presenting the authorization grant.

E. The user requests the protected resource from the resource server using the original credentials.

ANSWER: C D

Explanation:

"The client indirectly requests authorization from the resource owner through the authorization server" is a core OAuth authorization-flow step. The client directs the resource owner's user agent to the authorization server, where the resource owner can authenticate and approve the requested scope. This approach lets the client obtain delegated access without handling the resource owner's password. "The client requests an access token by presenting the authorization grant" is the subsequent token-exchange step. After authorization is granted, the client sends that grant to the authorization server's token endpoint; depending on the grant type and client authentication requirements, the authorization server returns an access token representing the approved authorization. The client then uses that token when calling the protected resource service. These steps express OAuth's separation of responsibilities: the authorization server manages authentication and consent, while the resource server accepts a valid access token for protected API access. This is the authorization-code-style interaction defined in the OAuth framework and commonly used by Cisco platform integrations. See RFC 6749, OAuth roles and protocol flow and RFC 6749 authorization code grant.

QUESTION NO: 4

What are two benefits of using distributed log collectors? (Choose two.)

A. supports multiple transport protocols such as TCP/UDP

B. improves performance and reduces resource consumption

C. provides flexibility due to a wide range of plugins and accepted log formats

D. enables extension of logs with fields and export to backend systems

E. buffers and resends data when the network is unavailable

ANSWER: B E

Explanation:

Distributed log collectors improve performance and reduce resource consumption because collection and forwarding are performed close to the log-producing workloads. This avoids requiring each application to establish separate, potentially expensive connections to a centralized logging platform and permits lightweight local processing such as parsing, filtering, batching, and compression. Distributing collectors also allows log ingestion capacity to scale with the environment, reducing bottlenecks at individual hosts or at a single collection point.

They also buffer and resend data when the network is unavailable. A collector can persist log records locally or in an in-memory/disk-backed buffer when its destination cannot be reached. It then retries delivery after connectivity or the backend service is restored. This resilience protects valuable operational and security events from being lost during transient network failures, planned maintenance, or destination-side overload. Reliable buffering and retry behavior are fundamental characteristics of production log-forwarding pipelines. Fluent Bit documents buffering, storage, and retry behavior in its Buffering and storage documentation, and its Scheduling and retries documentation describes how failed deliveries are retried.

QUESTION NO: 5

Into which two areas are AppDynamics APIs categorized? (Choose two.)

A. application-centric

B. analytics-events

C. database-visibility

D. platform-side

E. agent-side

ANSWER: D E

Explanation:

AppDynamics APIs are broadly organized by where the integration or extension operates: the AppDynamics platform or the monitored application's agent. **platform-side** APIs support interaction with platform services, particularly the Controller and associated AppDynamics services. They enable programmatic access to operational data and platform functions, such as querying application-performance information, managing configuration, and integrating AppDynamics data with external tools and workflows. **agent-side** APIs are available within, or in direct conjunction with, an AppDynamics language agent. Their purpose is to let developers extend instrumentation and reporting from the application runtime itself, including identifying business transactions, adding custom metrics, or supplying contextual data that the agent reports to the Controller. This separation is useful because platform-side integrations communicate with centralized AppDynamics services, whereas agent-side extensions work close to the instrumented application code. Cisco's AppDynamics documentation describes APIs and extensions in the context of the Controller platform and application agents, and the agent documentation covers custom instrumentation capabilities exposed at runtime. See AppDynamics APIs and App Server Agents.

QUESTION NO: 6

Which command is used 10 enable application hosting on a Cisco IOS XE device?

A. iox

B. iox-service

C. application -honing

D. app- hosting

ANSWER: A

Explanation:

The `iox` command is the global-configuration command that enables the Cisco IOx infrastructure on a Cisco IOS XE device. Cisco IOx supplies the application-hosting framework used to deploy and operate containerized applications directly on supported network platforms. Entering `iox` enables the underlying IOx services and makes the device ready for the subsequent application-hosting workflow, such as configuring application hosting resources, installing an application package, activating it, and starting it. This command is therefore the foundational device-level enablement step rather than an application-specific action. After IOx is enabled, operational status can be checked with IOS XE show commands, and applications can be managed through the application-hosting CLI or applicable APIs. The command must be entered from global configuration mode, typically after entering `configure terminal`. Cisco documents IOx as the IOS XE capability that provides the application hosting environment and identifies its enablement and configuration as part of the IOS XE programmability workflow. See the Cisco IOS XE IOx configuration documentation and the Cisco IOx developer documentation for the supported application-hosting architecture and procedures.

QUESTION NO: 7

```
curl --insecure -H "Accept: application/json" \
-H "Content-Type:application/json" \
-d @token_data \
https://ast0072-pod.cisco.com:33333//api/fdm/latest/fdm/token
```

Refer to the exhibit. This cURL POST request creates an OAuth access token for authentication with FDM API requests.

What is the purpose of the file "@token_data" that cURL is handling?

A. This file is given as input to store the access token received from FDM.

B. This file is used to send authentication-related headers.

C. This file contains raw data that is needed for token authentication.

D. This file is a container to log possible error responses in the request.

ANSWER: C

Explanation:

This file contains raw data that is needed for token authentication. In cURL, the `@` prefix used with a data option, such as `--data` or `-d`, instructs cURL to read the request body from the named local file rather than treating the filename itself as literal request data. Therefore, `@token_data` means that cURL opens the `token_data` file and sends its contents in the POST body to the OAuth token endpoint.

For an FDM access-token request, that body supplies the authentication payload expected by the token service, commonly including credentials or other token-request parameters in the format required by the endpoint. The file is an input source for the POST payload; it enables the request to keep token data separate from the command line, rather than embedding that data directly in the cURL command. cURL documents this file-reading behavior for the `--data` option, including the use of `@filename` to post the contents of a file. Cisco's FTD/FDM API documentation describes obtaining an access token before making authenticated API calls. See the cURL man page for --data and the Cisco Secure Firewall Threat Defense API documentation.

QUESTION NO: 8

How is an OAulh2 three-legged authentication flow initiated?

A. The user makes a request to the OAuth client.

B. Exchange the authorization code for an access token.

C. Construct an API call to retrieve the credentials.

D. Get the authorization code.

ANSWER: A

Explanation:

The user makes a request to the OAuth client is correct because a three-legged OAuth 2.0 flow begins when a resource owner uses an application, which acts as the OAuth client, and requests access to a protected resource or a feature requiring authorization. The client then creates an authorization request and redirects the user's browser to the authorization server's authorization endpoint. This browser-mediated interaction is what gives the flow its "three-legged" nature: the resource owner, client application, and authorization server all participate. After the user authenticates and grants consent, the authorization server redirects the browser back to the client with an authorization code. The client can then use that code at the token endpoint to obtain an access token for API calls. Therefore, the initial user request to the client starts the sequence; it establishes the user's intent to use the application and causes the client to begin the authorization redirect. The OAuth 2.0 authorization-code flow is defined in RFC 6749, section 4.1. Cisco's DevNet documentation also describes OAuth

authorization as a user-authorized process involving a client redirect to an authorization service: Cisco DevNet OAuth 2.0 documentation.

QUESTION NO: 9

An enterprise refactors its monolithic application into a modem cloud-native application that is based on microservices. A key requirement of the application design is to ensure that the IT team is aware of performance issues or bottlenecks in the new application Which two approaches must be part of the design considerations" (Choose two.)

A. Periodically scale up the resources of the host machines when the application starts to experience high loads

B. Instrument the application code to gather telemetry data from logs, metrics or tracing

C. Adopt a service-oriented architecture to handle communication between the services that make up the application

D. Deploy infrastructure monitoring agents into the operating system of the host machines

E. Implement infrastructure monitoring to ensure that pipeline components interoperate smoothly and reliably

ANSWER: B D

Explanation:

Instrument the application code to gather telemetry data from logs, metrics or tracing is essential because a microservices application distributes a single business transaction across numerous independently deployed services. Application instrumentation provides the observability signals needed to follow a request through service boundaries, measure latency and error rates, and correlate a user-facing slowdown with the service or operation responsible. Distributed tracing, structured logs, and metrics therefore make bottlenecks actionable rather than merely visible.

Deploy infrastructure monitoring agents into the operating system of the host machines is also required to establish the execution context for those application signals. Host-level agents collect resource and operating-system telemetry such as processor utilization, memory pressure, disk activity, network behavior, and process health. Combining that data with application telemetry lets operations personnel determine whether degraded service performance is caused by application behavior or by constrained underlying compute infrastructure. This end-to-end visibility is a core observability design practice for cloud-native workloads. Cisco AppDynamics describes application monitoring and infrastructure visibility at Cisco AppDynamics Application Monitoring; the relationship among logs, metrics, and traces is also described in the OpenTelemetry observability primer.

QUESTION NO: 10

Refer to the exhibit. The Dockerfile is placed in the current working directory. Which command builds an image named application:1.0.0 that contains application-1.0.0.jar and also includes any updates to the parent image?

A. docker build --put=VERSION="1.0.0" --tag application:1.0.0

B. docker build --build-arg VERSION="1.0.0" --tag application:1.0.0

C. docker build --pull --build-arg VERSION="1.0.0" --tag application:1.0.0 .

D. docker build --file=application-1.0.0.jar --tag application:1.0.0

ANSWER: C

Explanation:

`docker build --pull --build-arg VERSION="1.0.0" --tag application:1.0.0 .` is correct because it supplies all required build inputs in one command. The `--build-arg VERSION="1.0.0"` setting provides the value used by the Dockerfile's build-time `VERSION` variable, allowing its file-selection instruction to include `application-1.0.0.jar` in the resulting image. The `--tag application:1.0.0` setting assigns the requested repository name and version tag to the completed image. Crucially, `--pull` instructs Docker to attempt to download a newer version of the image referenced by

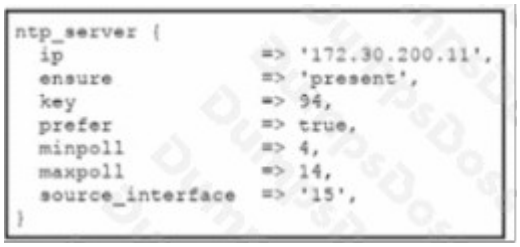
the Dockerfile's `FROM` instruction before building. This ensures that available parent-image updates are considered rather than relying solely on an already cached local parent image. Finally, the trailing period supplies the current directory as the build context. Because the Dockerfile and JAR are in that directory, Docker can read the Dockerfile and copy the required artifact into the image. Docker documents `--pull` and `--build-arg` in its docker build reference, and explains build contexts in the Docker Build context documentation.

QUESTION NO: 11

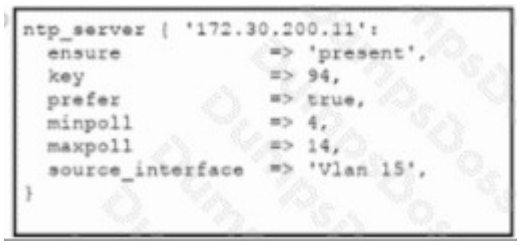
Which Puppet manifest changes the NTP server and generates the traffic from VLAN 15?

A)

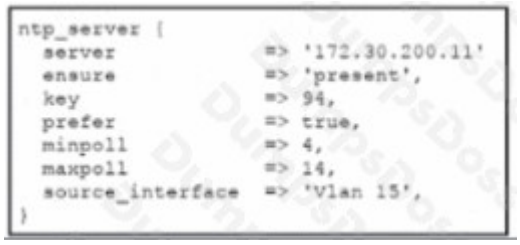
B)



C)



D)



A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: C

Explanation:

The correct manifest is the one that declares both required NTP attributes in the device configuration: the replacement NTP server and `Vlan15` as the NTP source interface. Configuring an NTP server defines the destination to which the device sends NTP requests, while configuring the source interface causes the device to originate those packets using the IP address assigned to VLAN 15. On Cisco IOS and IOS XE, this corresponds to the operational intent of the `ntp server` and `ntp source Vlan15` configuration commands. A Puppet manifest must express both pieces of desired state through the applicable Cisco NTP resource/provider so that Puppet can enforce the NTP peer/server setting and source-interface setting idempotently. This ensures that NTP traffic is sourced from VLAN 15 rather than from an interface selected implicitly by routing. Cisco documents NTP configuration, including selection of a source interface, in its IOS XE system-management guidance: Cisco IOS XE Time and Calendar Services Configuration Guide. Cisco's Puppet integration resources are available in the Cisco Puppet module repository.

QUESTION NO: 12

```
$ docker image push cisco-dev/test-image:latest
The push refers to repository [docker.io/cisco-dev/test-image]
tag does not exist: dhirotsu/test-image:latest

$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
cisco-dev/test-image    1.0                 cdec93a89ebb        2 hours ago         189MB
```

Refer to the exhibit. Which command publishes the Docker image to the private repository named cisco-dev with the latest tag?

A.

B.

C.

D.

ANSWER: C

Explanation:

The command represented by `<p><img src="D:\dumps_data\dumps_files\dumpsboss.co\site_temp_images\all-latest-b91076d194774e48bb4fbd6b7975f2a9\worker-2\4089_7.9/img-0536b43e7b8ce3d234fc9202.png"></p>` is correct because Docker publishes an image to a registry by using the `docker push` command and specifying the image repository and tag in the image reference. The required reference is `cisco-dev:latest`: `cisco-dev` identifies the target private repository, while `latest` explicitly identifies the tag to publish. Before pushing, the local image must already be tagged with that exact repository-and-tag name, and the Docker client must be authenticated to the registry when authentication is required. Docker's image naming convention supports the form `[REGISTRY_HOST[:PORT]/]NAME[:TAG]`, so a repository name and tag together uniquely select the local image manifest and layers that Docker uploads. Explicitly including `:latest` is appropriate here because the requested destination tag is stated directly rather than relying on Docker's default-tag behavior. Docker documents the push syntax as `docker image push [OPTIONS] NAME[:TAG]` and explains that a tag can be pushed as part of the image name. See the Docker image push command reference and Docker's image tagging reference.

QUESTION NO: 13

Why is end-to-end encryption deployed when exposing sensitive data through APIs?

A. Traffic is encrypted and decrypted at every hop in the network path.

B. Data transfers are untraceable from source to destination.

C. Data cannot be read or modified other than by the true source and destination.

D. Server-side encryption enables the destination to control data protection.

ANSWER: C

Explanation:

Data cannot be read or modified other than by the true source and destination. End-to-end encryption protects sensitive API payloads by encrypting data before it leaves the authorized sender and allowing decryption only by the intended recipient. Intermediary systems that forward, route, proxy, or store the encrypted payload do not have access to its plaintext content or the private keys needed to decrypt it. In addition to confidentiality, properly designed end-to-end protection uses authenticated encryption or integrity mechanisms so that an unauthorized alteration of the ciphertext is detected rather than accepted as valid data. This is particularly important for APIs carrying credentials, personal information, payment details, or business-critical commands across networks and services that may include infrastructure not controlled by both endpoint owners. Cisco's API security guidance emphasizes encrypting data in transit with TLS, while the security properties of TLS include confidentiality and integrity protection for communications between its authenticated endpoints. For API architectures, the precise endpoints and key-management design must be chosen carefully so that only the intended source and destination can access the protected information. See Cisco: What Is API Security? and RFC 8446: The Transport Layer Security Protocol Version 1.3.

QUESTION NO: 14

Which two encryption principles should be applied to secure APIs? (Choose two.)

A. Use temporary files as part of the encryption and decryption process

B. Transmit authorization information by using digitally signed payloads

C. Use encrypted connections to protect data in transit

D. Reuse source code that contain existing UUIDs

E. Embed keys in code to simplify the decryption process

ANSWER: B C

Explanation:

Using encrypted connections to protect data in transit is essential for API security because TLS encrypts the HTTP session between a client and the API endpoint. This protects confidential request and response content, such as credentials, access tokens, personal data, and application payloads, from interception or modification while traversing untrusted networks. API endpoints should therefore be exposed through HTTPS and configured to use current TLS versions and strong cipher suites.

Transmitting authorization information by using digitally signed payloads provides verifiable authorization context. A signed token or payload, such as a JSON Web Token, includes a cryptographic signature that allows the API to validate that the authorization data was issued by a trusted party and has not been altered. This supports stateless authorization decisions while preserving the integrity and authenticity of claims such as the subject, scopes, roles, audience, and expiry. The signature must be validated by the receiving API before the claims are trusted. Cisco API-security guidance emphasizes protecting API traffic with TLS and using secure token-based authentication and authorization mechanisms. See Cisco API Security and RFC 7519: JSON Web Token.

QUESTION NO: 15

What are two benefits of using a centralized logging service? (Choose two.)

A. reduces the time required to query log data across multiple hosts

B. reduces the loss of logs after a single disk failure

C. improves application performance by reducing CPU usage

D. improves application performance by reducing memory usage

E. provides compression and layout of log data

ANSWER: A B

Explanation:

A centralized logging service reduces the time required to query log data across multiple hosts because it collects events from distributed systems into a shared repository. Instead of connecting to each server, network device, or application instance separately, an operator can search, filter, correlate, and investigate relevant events from one location. This substantially improves operational visibility during troubleshooting and incident response, particularly in environments with many ephemeral or geographically distributed workloads.

It also reduces the loss of logs after a single disk failure by transmitting or forwarding records away from the host that generated them. If the local disk or the host itself subsequently fails, previously exported events remain available in the centralized logging platform, subject to the platform's retention and durability configuration. Central collection is therefore a key part of resilient observability and audit logging architectures. Cisco describes centralized logging as enabling log collection and analysis across infrastructure, while the Syslog protocol supports sending event messages to remote collectors. See Cisco ISE Monitoring and Logging documentation and RFC 5424: The Syslog Protocol.

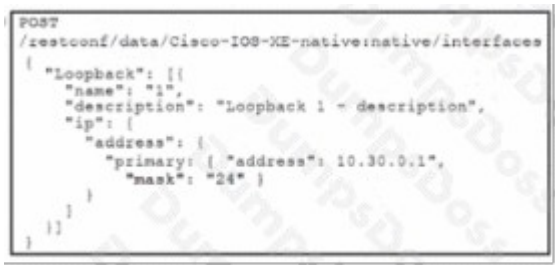
QUESTION NO: 16

Refer to the exhibit.

Interface Lookback 1 must be created with IP address 10.30.0.1/24 in a Cisco IOS XE device using RESTCONF. The schema that is defined by the exhibit must be used. Which body and URI should be used for this operation?

A)

B)



C)

```
POST
/restconf/data/Cisco-IOS-XE-native:native/interface
{
  "Loopback": [{
    "name": "1",
    "description": "Loopback 1 - description",
    "ip": {
      "address": {
        "primary: { "address": 10.30.0.1",
          "mask": "255.255.255.0" }
      }
    }
  }]
}
```

D)

```
PUT
/restconf/data/Cisco-IOS-XE-native:native/interface
{
  "Loopback": [{
    "name": "1",
    "description": "Loopback 1 - description",
    "ip": {
      "address": {
        "primary: { "address": 10.30.0.1",
          "mask": "24" }
      }
    }
  }]
}
```

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: A

Explanation:

The correct request targets the `Loopback` list under the Cisco IOS XE native interface container and supplies the list entry as JSON using the module-qualified `Cisco-IOS-XE-native:Loopback` member. The URI ends at the list collection, `/restconf/data/Cisco-IOS-XE-native:native/interface/Loopback`, so that a POST can create a new list instance. The payload provides `name` with the loopback number and follows the YANG hierarchy for the configured IPv4 primary address: `ip`, then `address`, then `primary`, containing the address and dotted-decimal mask. Therefore, the body must specify `10.30.0.1` and `255.255.255.0` beneath that hierarchy. RESTCONF resources are addressed from the data root according to the YANG data tree, and JSON property names are qualified with the defining module name when required. Cisco IOS XE exposes native configuration through the `Cisco-IOS-XE-native` YANG model, including interface configuration. See Cisco's IOS XE RESTCONF documentation and the RESTCONF specification for RESTCONF data-resource and JSON-encoding behavior.

QUESTION NO: 17

Refer to the exhibit.

```
[all:vars]
ansible_connection=
ansible_user=admin
ansible_network_os=ios
```

An engineer is configuring Ansible to run playbooks against Cisco IOS XE Software. What

should be configured in ansible.cfg as the connection type?

A. network_cli

B. ssh

C. shell

D. command

ANSWER: A

Explanation:

`network_cli` is the appropriate connection type for managing Cisco IOS XE devices through Ansible network modules. It uses the Ansible network connection framework to establish an SSH-based CLI session to the device, handle network-device login behavior, and maintain the persistent connection required by IOS-oriented modules. This lets playbooks use Cisco IOS collection modules, such as operational command and configuration modules, without treating the network device as a conventional Linux host.

For IOS XE automation, the connection is commonly defined as `ansible_connection=ansible.netcommon.network_cli` in inventory or group variables; depending on the configuration context and Ansible version, the short name `network_cli` identifies the same network CLI connection plugin. The plugin is specifically intended for network operating systems whose automation is performed through their command-line interface. Cisco IOS modules document `network_cli` as the CLI connection mechanism, and Ansible documents the plugin's persistent SSH CLI behavior and authentication settings.

See the Ansible network_cli connection documentation and the Cisco IOS command module documentation.

QUESTION NO: 18

Which two statements about a stateless application are true? (Choose two.)

A. Different requests can be processed by different servers.

B. Requests are based only on information relayed with each request.

C. Information about earlier requests must be kept and must be accessible.

D. The same server must be used to process all requests that are linked to the same state.

E. No state information can be shared across servers.

ANSWER: A B

Explanation:

A stateless application treats every client request as an independent transaction. The server does not need to retain client-session context from a prior request in order to understand and fulfill the next one; instead, the request includes all information necessary for processing, such as authentication credentials, parameters, and the resource being requested. This design aligns with the REST constraint that client-server communication be stateless, which improves visibility, reliability, and scalability because each request can be evaluated independently.

Because no client-specific session state must remain on a particular application instance, different requests from the same client can be routed to different servers by a load balancer. This makes horizontal scaling straightforward: instances can be added, removed, or replaced without requiring session affinity or synchronization of per-client state. Stateless services may still access shared durable resources, such as databases or caches, when needed; the defining characteristic is that the application does not depend on server-local conversational state from earlier requests. Cisco application-development guidance commonly uses RESTful APIs as an example of this request-oriented model. See the HTTP Semantics RFC and Cisco DevNet REST API learning materials.

QUESTION NO: 19 - (DRAG DROP)

Refer to the exhibit.

```
import request
import json
import sys

token = ""

def get_dnac_devices():
    <item 1>:
        url = "https://sanboxdnac.cisco.com/dna/intent/api/v1/network-device"

        print(token)
        payload = {}
        headers = {
        'Content-Type': 'application/json',
        "Accept": 'application/json',
        'x-auth-token': token
        }

        response = requests.request("GET", url, headers=headers, data = payload)
        response.raise_for_status()
        return response.text

    <item 2>:
        print(e)
        if str(<item 3>) in str(e):
            create_dnac_token()

def create_dnac_token():
    try:
        url = "https://sanboxdnac.cisco.com/dna/system/api/v1/auth/token"

        payload = {}
        headers ={
        '<item4>': 'Basic ZGV2bmV0dXNlcjpDaXNjbzEyMyE=',
        'Content-Type': 'application/json'
        }

        response = requests.request("POST", url, headers=headers, data = payload)
        response.raise_for_status()
        return response.json()["Token"]
    except Exception as e:
        print(e)
        if str(<item 5>) in str(e):
            sys.exit("DNAC Service is not reachable")

if __name__ == "__main__":
    token = create_dnac_token()
    print(get_dnac_devices())
```

Drag and drop the code snippets from the left onto the item numbers on the right that match the missing sections in the exhibit to complete the script to implement control flow.

except Exception as e	<item 1>
try	<item 2>
Authorization	<item 3>
request.status_codes.codes.SERVER_ERROR	<item 4>
request.status_codes.codes.UNAUTHORIZED	<item 5>

ANSWER:

Explanation:

The completed script uses Python exception control flow to handle failures from the Cisco DNA Center REST API cleanly. The `try` statement starts the protected portion of `get_dnac_devices()`, which includes constructing the request, sending the GET operation, checking its result, and returning the response text. The corresponding `except Exception as e` block receives an exception raised during that work. Converting the exception to text allows the script to identify an unauthorized condition. When the error contains `request.status_codes.codes.UNAUTHORIZED`, the script calls `get_dnac_token()` so that authentication can be renewed before further API activity.

In `create_dnac_token()`, the HTTP header name `Authorization` is required for the Basic credential value used to request an access token. This is the standard HTTP header that carries authentication credentials. Cisco DNA Center API requests use a token obtained from the authentication endpoint, and later requests include that token in the appropriate authentication header. Cisco documents this token-based workflow in its DNA Center authentication documentation.

The second exception handler prints the received exception and checks for `request.status_codes.codes.SERVER_ERROR`. A server error means the service cannot successfully process the request at that time, so terminating with the message that the DNA Center service is not reachable is consistent with the script's intended control flow. Together, these placements create two complete `try`/`except` structures: one that recovers from an authorization failure by obtaining a token, and another that exits when a server-side service failure is encountered. Python's documented exception-handling model supports this exact structure, as described in the Python errors and exceptions tutorial.

QUESTION NO: 20

While working with the Webex Teams API, on an application that uses end-to-end encryption, a webhook has

been received. What must be considered to read the message?

A. Webhook information cannot be used to read the message because of end-to-end encryption. The API key is needed to decrypt the message.

B. Webhook returns the full unencrypted message. Only the body is needed to query the API.

C. Webhook returns a hashed version of the message that must be unhashed with the API key.

D. Webhook returns message identification. To query, the API is needed for that message to get the decrypted information.

ANSWER: D

Explanation:

Webhook returns message identification. To query, the API is needed for that message to get the decrypted information.

This is correct because Webex webhook notifications are deliberately minimal and do not include sensitive message content. A webhook identifies the affected resource, including its message identifier, so the receiving application can determine which message changed or was created. Since Webex initiates delivery of the webhook to the application's endpoint, that delivery does not carry the application user's OAuth access token and therefore cannot provide protected message text as part of the notification.

To read the content, the application uses the message ID supplied in the webhook payload to make an authenticated Webex REST API request for the specific message resource, such as `GET /messages/{messageId}`. The request must include a valid Bearer access token with authorization to access that space and message. Webex then returns the complete message representation, including the protected content available to that authorized identity. This two-step design lets an application react quickly to events while preserving message confidentiality and enforcing authorization when sensitive data is retrieved. See the Webex webhooks guide and the Get a Message Details API reference.

QUESTION NO: 21

An application is developed in order to communicate with Cisco Webex. For reporting, the application must retrieve all the messages sent to a Cisco Webex room on a monthly basis.

Which action calls /v1/messages directly?

A. Set up a webhook that has messages as the resource type and store the results locally.

B. Utilize the pagination functionality by defining the max property.

C. Recursively call the /v1/messages endpoint by using the beforeMessage property.

D. Filter the response results by specifying the created property in the request.

ANSWER: C

Explanation:

Recursively call the `/v1/messages` endpoint by using the `beforeMessage` property. The Webex Messages API lists messages in reverse chronological order and returns only a page of results per request. For a reporting application that must collect a room's full monthly message history, it should first issue a `GET /v1/messages` request scoped to the relevant room, typically using `roomId` and an appropriate page size. After processing the returned messages, the application uses the identifier of the oldest message received as the `beforeMessage` value in the next request. Repeating this request moves backward through the room's message history until the required monthly boundary has been reached or no additional messages are returned.

This approach directly invokes the Messages REST resource for each page and provides a reliable cursor-like mechanism for traversing historical results. The reporting process can store each page locally, deduplicate by message ID if necessary, and stop once it has collected messages for the intended reporting interval. Cisco documents the available query parameters, including `beforeMessage`, for the list operation at Webex List Messages API. General request authentication and API usage are documented in the Webex API Getting Started guide.

QUESTION NO: 22

```
module: ietf-interfaces
  +--rw interfaces
  |  +--rw interface* [name]
  |  +--rw name                          string
  |  +rw description?                    string
  |  +--rw type                          identityref
  |  +--rw enabled?                      boolean
  |  +--rw link-up-down-trap-enable?     enumeration {if-mib}?
  +--ro   interfaces-state
    +--ro interface* [name]
     +--ro name                 string
     +--ro type                 identityref
     +--ro admin-status         enumeration {if-mib}?
     +--ro oper-status          enumeration
     +--ro last-change?         yang:date-and-time
     +--ro if-index             int32 {if-mib}?
     +--ro phys-address?        yang:phys-address
     +--ro higher-layer-if*     interface-state-ref
     +--ro lower-layer-if*      interface-state-ref
     +--ro speed?               yang:gauge64
     +--ro statistics
              +--ro discontinuity-time     yang:date-and-time
              +--ro in-octets?             yang:counter64
              +--ro in-unicast-pkts?       yang:counter64
              +--ro in-broadcast-pkts?     yang:counter64
              +--ro in-multicast-pkts?     yang:counter64
              +--ro in-discards?           yang:counter32
              +--ro in-errors?             yang:counter32
              +--ro in-unknown-protos?     yang:counter32
              +--ro out-octets?            yang:counter64
              +--ro out-unicast-pkts?      yang:counter64
              +--ro out-broadcast-pkts?    yang:counter64
              +--ro out-multicast-pltas?   yang:counter64
              +--ro out-discards?          yang:counter32
              +--ro out-errors?            yang:counter32
```

```
import requests
url = ("https://ios-xe-mgmt.cisco.com:9443/restconf/data/ietf-interfaces:" +
    "interfaces/interface=GigabitEthernet2")

headers = {
    'Accept': "application/yang-data+json",
    'Authorization': "Basic cm9vdDpEX1ZheSFfMTAm",
    'Content-Type': "application"
}

response = requests.request(rest_operation, url, data=payload,
    headers = headers, verify=False)

print (response.text)
```

Refer to the exhibits. An interface named “GigabitEthernet2” has been configured on a Cisco IOS XE device. Using RESTCONF APIs as defined by the ietf-interfaces@2014-05-08.yang model, which two combinations of “rest_operation” and “payload” must be added to the Python script to set the “description” to “Configured by RESTCONF”? (Choose two.)

A.

```
rest_operation = "PATCH"

payload = " {\n    \"ietf-interfaces:interface\": {\n
     \"name\": \"GigabitEthernet2\", \n
     \ "description\": \"Configured by RESTCONF\" \n
     }\n}"
```

B.

```
rest_operation = "PUT"

payload = " {\n    \"ietf-interfaces:interface\": {\n
     \"name\": \"GigabitEthernet2\", \n
     \ "description\": \"Configured by RESTCONF\" \n
     }\n}"
```

C.

```
rest_operation = "PUT"

payload = "{\n \"ietf=interfaces:interface\": {\n
     \"name\": \"GigabitEthernet2\",\n
     \"description\": \"Configured by RESTCONF\", \n
     \"type\": \"iana-if-type:ethernetCsmacd\", \n
     \"enabled\" true,\n \"ietf-ip:ipv4\": {\n
          \"address\": [\n{\n\"ip\": \"10.255.255.1\",
          \n"netmask\": \"255.255.255.0\"\n  }\n
     ]\n       }\n       }\n}"
```

D.

```
rest_operation = "POST"

payload = " {\n    \"ietf-interfaces:interface\": {\n
     \"name\": \"GigabitEthernet2\", \n
     \ "description\": \"Configured by RESTCONF\" \n
     }\n}"
```

E.

```
rest_operation = "POST"

payload = "{\n \"ietf=interfaces:interface\": {\n
     \"name\": \"GigabitEthernet2\",\n
     \"description\": \"Configured by RESTCONF\", \n
     \"type\": \"iana-if-type:ethernetCsmacd\", \n
     \"enabled\" true,\n \"ietf-ip:ipv4\": {\n
          \"address\": [\n{\n\"ip\": \"10.255.255.1\",
          \n"netmask\": \"255.255.255.0\"\n  }\n
     ]\n       }\n       }\n}"
```

A. Option A

B. Option B

C. Option C

D. Option D

E. Option E

ANSWER: A B

Explanation:

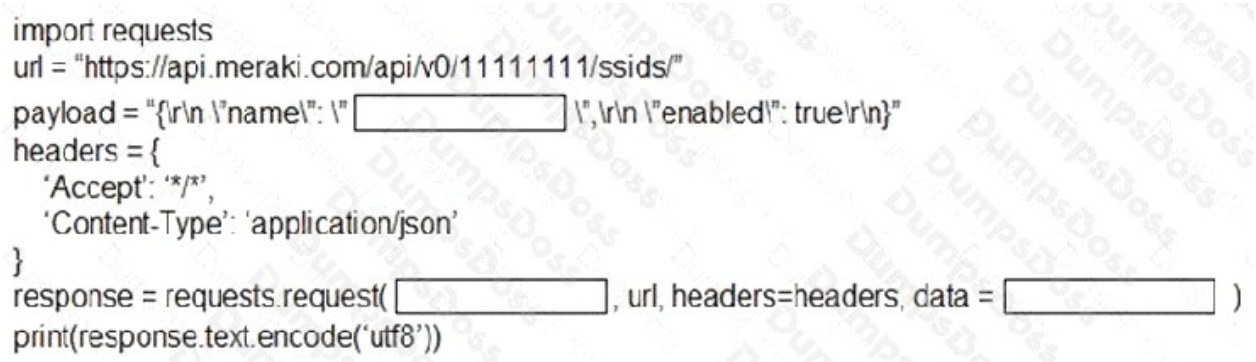
The two selected RESTCONF combinations correctly update the existing `GigabitEthernet2` interface's description through the RESTCONF data resource representing the interface configuration. RESTCONF supports `PATCH` for a partial modification of an existing resource, which is appropriate when changing only the `description` leaf while preserving the rest of the interface configuration. The JSON payload must use the YANG JSON encoding and identify the `ietf-interfaces` data node correctly, with the description value set to `Configured by RESTCONF`.

The selected combination using `PUT` is also valid when the request targets the description leaf resource directly and supplies that leaf's valid JSON representation. RESTCONF defines `PUT` as creating or replacing the resource addressed by the request URI; therefore, when the target is the description leaf rather than the complete interface object, it updates that one configuration value. The interface model defines `description` as a configurable leaf under an interface entry, so its value can be changed through the interface's RESTCONF configuration path. See RFC 8040, RESTCONF Protocol and RFC 7223, YANG Interface Management Model.

QUESTION NO: 23 - (SIMULATION)

FILL BLANK

Fill in the blanks to complete the Python script to enable the SSID with a name of "371767916" in the network resource "11111111" using the Meraki Dashboard API.



ANSWER: See the explanation for the answer

Explanation:

Fill the blanks as follows:

The completed script is:

A `PUT` request updates the SSID resource, and the JSON payload sets its name and enables it.

QUESTION NO: 24

A developer is working on an enhancement for an application feature and has made changes to a branch called 'devcor-432436127a-enhance4'. When merging the branch to production, conflicts occurred. Which Git command must the developer use to recreate the pre-merge state?

A. git merge -no-edit

B. git merge --abort

C. git merge -revert

D. git merge —comrmt

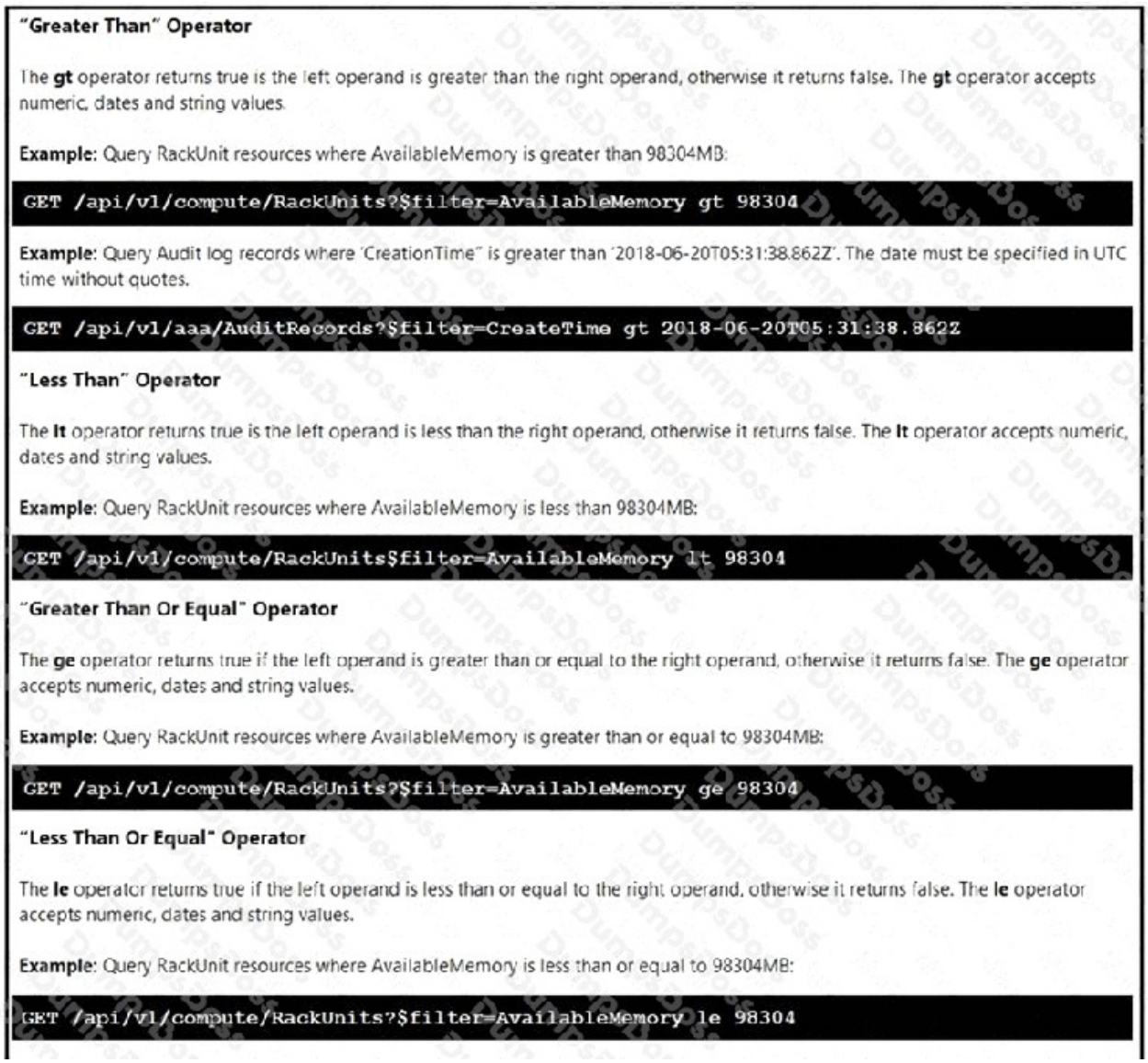
ANSWER: B

Explanation:

`git merge --abort` is the Git command used while a merge is in progress and conflicts have stopped it from completing. Git records the state that existed before it began the merge, including the original branch tip and working-tree/index state required to return to that point. Running this command cancels the unfinished merge operation, clears merge-conflict state such as `MERGE_HEAD`, and restores the branch to its pre-merge state. This lets the developer inspect or update the changes before attempting the merge again. The command is intended specifically for an uncommitted, conflicted merge; it does not create a new commit or rewrite completed merge history. As a practical precaution, developers should avoid starting a merge with unrelated uncommitted modifications, because Git may not always be able to reconstruct those pre-existing local changes exactly. Git documents that `git merge --abort` attempts to reconstruct the state before the merge began, and notes that it is equivalent to resetting the merge state when `MERGE_HEAD` exists. See the official Git merge documentation and the --abort command reference.

QUESTION NO: 25

Click on the *GET Resource* button above to view resources that will help with this question.



"And" Operator

The **and** operator returns true if both the left and right operands evaluate to true, otherwise it returns false.

Example: Query RackUnit resources where the Model property is equal to 'UCSC-C240- M5SN' and thy server has more than 64GB of memory:

```
GET /api/v1/compute/RackUnits?$filter=Model eq 'UCSC-C240-M5SN' and AvailableMemory gt
65000
```

"Or" Operator

The **or** operator returns true if either the left or right operand evaluate to true, otherwise it returns false.

Example: Query RackUnit resources where the Model property is equal to 'UCSC-C240-M5SN' **or** the Model property is equal to 'UCSC-C240-M5SN'. Use the $select keyword to reduce the size of the output JSON document.

"Not" Operator

The **not** operator returns true if the operand returns false, otherwise it returns false.

Example: Query RackUnit resources where the model property is not ('HX220C-M5SX' or 'HX220C-M5S'). The example shows how grouping parenthesis can be used to set the operator precedence.

```
GET /api/v1/compute/RackUnits?$select=Vendor,Model,Serial&top=10&$filter=not(Model eq
'HX220C-M5SX' or Model eq 'HX220C-M5S')
```

"In" Operator

The **in** operator returns true if the left operand is equal to one of the values specified in the right operand, otherwise it returns false. The **in** operator accepts numeric and string values.

Values must be specified as a comma-separated list enclosed in parenthesis.

Example: Query RackUnit resources where the Model is either 'HX220C-M5SX' or 'UCSC-C240-M5SN'.

```
GET /api/v1/compute/RackUnits?$filter=Model in ('HX220C-M5SX','UCSC-C240-M5SN')
```

String Functions

"contains" Function

The **contains** function has the following signature:

boolean contains(s string, subst string)

The **contains** function returns true if the second parameter string value is a substring of the first parameter string value, otherwise it returns false.

Example: Query RackUnit resources where the value of the 'Model' property contains 'C240'

```
GET /api/v1/RackUnits?$filter=contains(Model,'C240')
```

"startsWith" Function

The **startswith** function has the following signature:

boolean startswith(s string, subst string)

The **startswith** function returns true if the first parameter string value starts with the second parameter string value, otherwise it returns false.

Example: Query RackUnit resources where the value of the 'Model' property starts with the prefix 'UCSC-C240'

```
GET /api/v1/RackUnits?$filter=startswith(Model,'UCSC-C240')
```

"endswith" Function

The endswith function has the following signature:

boolean endswith(string, suffix string)

The **endswith** function returns true if the first parameter string value ends with the second parameter string value, otherwise it returns false.

Example: Query RackUnit resources where the value of the 'Model' property ends with the suffix 'M5'

```
GET /api/v1/RackUnits?$filter=endswith(Model,'M5')
```

"tolower" Function

The **tolower** function has the following signature:

string tolower(string)

An engineer is managing a DC with 6000 Cisco UCS servers installed and running. The engineer has been asked to identify all resources where the model is in the UCSB family and the available memory is less than or equal to 5 GB.

Which REST API call accomplishes this task?

A. GET/api/v1/compute/RackUnits?$select=Vendor,Model,Serial & $filter=not(Model eq 'UCSC') and AvailableMemory le 5000

B. GET/api/v1/compute/RackUnits?$select=Vendor,Model,Serial & $filter=Model eq 'UCSB' and AvailableMemory lt 5000

C. GET/api/v1/compute/RackUnits?$select=Vendor,Model,Serial & $filter=contains(Model, UCSB') and AvailableMemory lt 5000

D. GET/api/v1/compute/RackUnits?$select=Vendor,Model,Serial & $filter=contains(Model, 'UCSB') and AvailableMemory le 5000

ANSWER: D

Explanation:

`GET/api/v1/compute/RackUnits?$select=Vendor,Model,Serial & $filter=contains(Model, 'UCSB') and AvailableMemory le 5000` is correct because it combines server-side projection and filtering in a single REST request. The `$select` query parameter limits the returned properties to the requested vendor, model, and serial-number fields, which reduces unnecessary response data when querying a large inventory. The `$filter` expression uses `contains(Model, 'UCSB')` to identify models whose model value includes the UCSB family identifier. It then joins that test with `and` and applies `AvailableMemory le 5000`. In the supported OData-style filter syntax, `le` means "less than or equal to," so a resource reporting exactly 5,000 MB is included along with resources reporting less available memory. This makes the request satisfy both required conditions while having the API perform the filtering, rather than retrieving thousands of resources and filtering them locally. Cisco documents its REST API resources and query capabilities in the Cisco Intersight API documentation; additional Cisco developer guidance is available in the Cisco Intersight developer documentation.

QUESTION NO: 26

Which two statements about Kubernetes readiness probes are true? (Choose two.)

A. A failed readiness probe removes a Pod from Service endpoints.

B. A readiness probe can prevent traffic from reaching an application before it is ready.

C. A failed readiness probe always restarts the container.

D. A readiness probe is evaluated only when a Pod is created.

E. A readiness probe replaces the need for a Service.

ANSWER: A B

Explanation:

A failed readiness probe removes a Pod from Service endpoints and **a readiness probe can prevent traffic from reaching an application before it is ready**. A readiness probe determines whether a container is ready to accept requests. When a readiness check fails, Kubernetes marks the Pod as not ready and removes it from the endpoints used by Services, so normal Service traffic is not directed to that Pod.

Readiness is different from liveness. A liveness-probe failure can cause Kubernetes to restart a container, whereas a readiness failure primarily controls whether the Pod should receive traffic. Kubernetes documents probe behavior and Service endpoint handling in the Liveness, Readiness, and Startup Probes documentation.

QUESTION NO: 27

Which two design considerations should be considered when building a Cisco Meraki dashboard out of available APIs? (Choose two.)

A. If the API key is shared, it cannot be regenerated.

B. The API requests require the key and the user credentials.

C. API call volume is rate-limited to five calls per second per organization.

D. The API version does not need to be specified in the URL.

E. Access to the API must first be enabled by using the settings for an organization.

ANSWER: C E

Explanation:

API call volume is rate-limited to five calls per second per organization and access to the API must first be enabled by using the settings for an organization are key considerations for an application designed around the Meraki Dashboard API. A

dashboard commonly polls multiple networks, devices, and clients, so its requests must be paced, consolidated where possible, and designed to handle rate-limit responses gracefully. The stated five-requests-per-second value reflects the legacy Dashboard API guidance associated with this exam material. For current Meraki API v1 implementations, developers should verify the live rate-limit policy before deployment, because current limits and burst behavior are documented separately and can evolve. API access also depends on organization-level Dashboard API enablement. This is an administrative control: even a properly constructed integration using an API key requires the organization to permit Dashboard API access. Consequently, an application's deployment plan should identify the target organization, ensure API access has been enabled by an authorized administrator, and securely provision credentials with appropriate organization access. Cisco documents API rate-limit behavior at Meraki Dashboard API Rate Limits and organization-level API enablement and API-key use at Cisco Meraki Dashboard API documentation.

QUESTION NO: 28

A developer releases a new application for network automaton of Cisco devices deployed in a local data center. The application utilizes complex design patterns such as microservices that host multiple third-party libraries and programming languages. The development must be simplified by implementing an observability-driven development lifecycle. Which two considerations must be taken to meet the requirements? (Choose two.)

A. which monitoring tools to use

B. identifying customer priorities

C. relevant metrics to expose

D. description of low-level errors

E. which KPIs to monitor

ANSWER: D E

Explanation:

An observability-driven development lifecycle needs to define the outcomes that demonstrate whether the application is delivering reliable and useful service, as well as make failures understandable when they occur. **which KPIs to monitor** is essential because KPIs connect technical telemetry to the application's intended service outcomes, such as availability, latency, error rate, throughput, or successful network-automation transactions. Establishing these measures early lets developers instrument services consistently and gives operations teams meaningful signals for evaluating health across a distributed application.

A **description of low-level errors** is also required because microservices and third-party dependencies can fail at many layers. Errors should be emitted with actionable details and sufficient contextual information to identify the failed operation, dependency, and request path. This makes telemetry useful for diagnosing production incidents instead of merely reporting that a request failed. Together, KPI-focused monitoring and meaningful error descriptions provide the business-level and technical-level visibility required to operate complex distributed applications. Cisco describes full-stack observability as correlating application and infrastructure insights to improve operational outcomes; see Cisco Full-Stack Observability. The role of metrics, logs, and traces in producing this diagnostic context is also described in the OpenTelemetry Observability Primer.

QUESTION NO: 29

A developer plans to create a new bugfix branch to fix a bug that was found on the release branch.

Which command completes the task?

A. git checkout -t RELEASE BUGFIX

B. git checkout -b RELEASE BUGFIX

C. git checkout -t BUGFIX RELEASE

D. git checkout -b BUGFIX RELEASE

ANSWER: D

Explanation:

`git checkout -b BUGFIX RELEASE` is correct because the `-b` form of `git checkout` creates a new branch and immediately switches the working tree to that branch. In this command, `BUGFIX` is the name assigned to the new branch, while `RELEASE` is the start point from which the branch is created. Consequently, the new bugfix branch begins at exactly the commit currently identified by the release branch, preserving the proper release code context for the fix.

This is the appropriate workflow when a defect is discovered in code that belongs to a release line: create a separate branch based on that release branch, implement and validate the correction there, and then merge or otherwise apply the resulting commit according to the team's release process. Git documents the syntax as `git checkout -b <new_branch> [<start_point>]`; when a start point is supplied, it determines the commit at which the new branch is created. The equivalent modern command is `git switch -c BUGFIX RELEASE`, but the specified checkout command directly completes the requested task. See the official Git checkout documentation and the Git branching guide.

QUESTION NO: 30

Refer to the exhibit.

```
import requests
from requests.auth import HTTPBasicAuth
import json

url = 'https://na.cloudcenter.cisco.com/cloudcenter-ccm-backend/api/v1/apps'
response = requests.request("GET", url, auth=HTTPBasicAuth('32768', '2b...'),
verify=False)
result = json.loads(response.text)

page = 0
totalPages = 1
while(page < totalPages):
    url = 'https://na.cloudcenter.cisco.com/cloudcenter-ccm-backend/api/v1/
    virtualMachines?page={page}'
    response = requests.request("GET", url, auth=HTTPBasicAuth('32768',
    '2b...'), verify=False)
    result = json.loads(response.text)

    totalPages = result["details"]["totalPages"]
    page = page + 1
```

An application has been created to serve a whole enterprise. Based on use and department requirements, changes are requested on a quarterly basis. When evaluating the application design, which two actions improve code maintainability?

A. Replace the requests library with the http client library in the code.

B. Place all import statements on a single line at the top of the code.

C. Cache responses to API calls for later reuse on other code.

D. Parameterize similar code blocks inside functions and reuse within the code.

E. Add comments in appropriate locations to aid in understanding the code.

ANSWER: D E

Explanation:

Parameterize similar code blocks inside functions and reuse within the code. improves maintainability by applying the DRY (Don't Repeat Yourself) principle. Common behavior is defined in one reusable function, while parameters supply the values that vary between calls. When a quarterly change affects that behavior, a developer can update and test one implementation rather than locating and modifying multiple near-duplicate blocks. This reduces inconsistency, regression risk, and the cost of future enhancements.

Add comments in appropriate locations to aid in understanding the code. also supports long-term maintenance, particularly for an enterprise application that may be changed by developers who did not write the original implementation. Focused comments can document intent, business rules, design decisions, non-obvious constraints, and reasons for unusual behavior. This context helps maintainers safely assess the impact of requested changes and makes code reviews and troubleshooting more efficient. Comments should complement clear naming and modular functions rather than restate obvious syntax. Cisco DevNet emphasizes creating reusable, maintainable code and documenting application behavior as part of effective software-development practice. See the Cisco DevNet Associate learning track and the Python documentation on defining reusable functions.

QUESTION NO: 31

Click on the *GET Resource* button above to view resources that will help with this question.

An engineer is managing a DC with 6000 Cisco UCS servers installed and running. The engineer has been asked to identify all resources where the model is in the UCSB family and the available memory is less than or equal to 5 GB.

Which REST API call accomplishes this task?

A. GET/api/v1/compute/RackUnits?$select=Vendor,Model,Serial&$filter=not(Model eq 'UCSC') and AvailableMemory le 5000

B. GET/api/v1/compute/RackUnits?$select=Vendor,Model,Serial&$filter=Model eq 'UCSB' and AvailableMemory lt 5000

C. GET/api/v1/compute/RackUnits?$select=Vendor,Model,Serial&$filter=contains(Model, 'UCSB') and AvailableMemory lt 5000

D. GET/api/v1/compute/RackUnits?$select=Vendor,Model,Serial&$filter=contains(Model, 'UCSB') and AvailableMemory le 5000

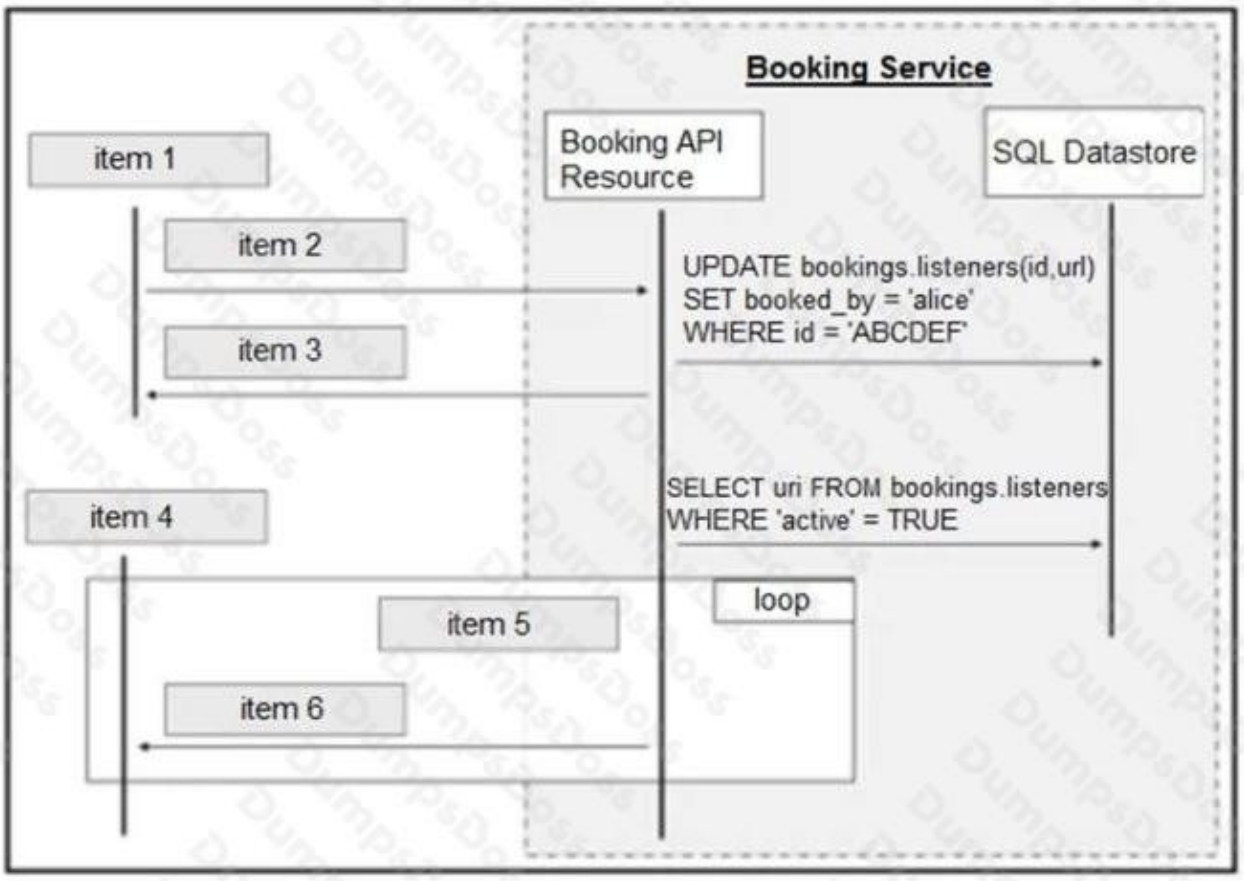
ANSWER: D

Explanation:

`GET/api/v1/compute/RackUnits?$select=Vendor,Model,Serial&$filter=contains(Model, 'UCSB') and AvailableMemory le 5000` is correct because it combines projection and server-side filtering in a single REST request. The `$select` parameter limits the returned fields to the vendor, model, and serial number, which is appropriate when the task is to identify matching resources rather than retrieve every property of each server. The `$filter` expression uses `contains(Model, 'UCSB')` to match model values belonging to the UCSB blade-server family, including model strings that contain additional platform or generation details. It also uses the OData relational operator `le`, meaning “less than or equal to,” so resources with exactly 5,000 MB of available memory are included along with those below that threshold. The logical `and` ensures that each returned resource satisfies both required conditions. Cisco Intersight REST resources support query parameters such as `$select` and `$filter` for reducing and filtering API results; see the Cisco Intersight query-parameter documentation and the Cisco Intersight API reference.

QUESTION NO: 32 - (DRAG DROP)

DRAG DROP



Refer to the exhibit above and click on the tab in the top left corner to view a diagram that describes the typical flow of requests involved when a webhook is created for a booking service. Drag and drop the requests from the left onto the item numbers on the right that match the missing sections in the sequence diagram to design the complete flow of requests involved as a booking is updated from a web application.

Select and Place:

Answer Area

Web Application	item 1
POST <listener.uri> {"bookingId": "ABCDEF"}	item 2
PATCH /bookings/ABCEDF {"bookedBy":"alice"}	item 3
Webhook Listener	item 4
204 NO CONTENT	item 5
[for each listener]	item 6

ANSWER:

Answer Area

Web Application	Web Application
POST <listener.uri> {"bookingId": "ABCDEF"}	PATCH /bookings/ABCEDF {"bookedBy":"alice"}
PATCH /bookings/ABCEDF {"bookedBy":"alice"}	[for each listener]
Webhook Listener	Webhook Listener
204 NO CONTENT	POST <listener.uri> {"bookingId": "ABCDEF"}
[for each listener]	204 NO CONTENT

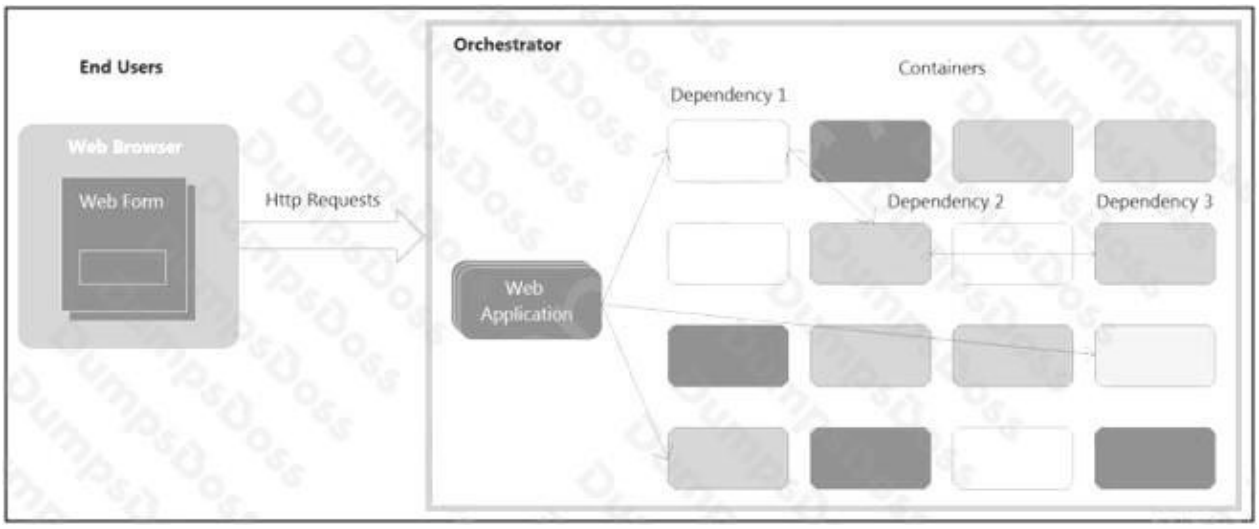
Explanation:

A booking update begins when the web application sends a PATCH request to the booking resource, carrying the change to the booking data. Once the booking service accepts and processes that update, it determines which webhook subscriptions must receive an event for the changed booking. The notification work is performed once for every registered listener, so the per-listener loop belongs immediately after the booking update and before any individual listener is contacted.

Within each pass through that loop, the flow identifies the webhook listener and sends an HTTP POST request to the listener URI. The POST body includes the booking identifier, allowing the receiving application to determine which booking event it needs to process. The listener then acknowledges successful receipt with an HTTP 204 No Content response. A 204 status is appropriate for an acknowledgement where the webhook receiver has accepted the notification but does not need to return a response body.

This ordering models the core webhook pattern: an originating application changes a resource, the service iterates over subscribed callback endpoints, and each endpoint receives a notification request followed by an acknowledgement. Cisco DevNet's webhook material describes webhooks as event-driven HTTP callbacks and provides the relevant callback and listener concepts in its Webhooks learning module. The HTTP semantics of the acknowledgement are defined by the IETF in RFC 9110, 204 No Content.

QUESTION NO: 33



Refer to the exhibit. The application follows a containerized microservices architecture that has one container per microservice. The microservices communicate with each other by using REST APIs. The double-headed arrows in the diagram display chains of synchronous HTTP calls needed for a single user request.

Which action ensures the resilience of the application in the scope of a single user request?

A. Implement retries with exponential backoff during HTTP API calls.

B. Set up multiple instances of each microservice in active/active mode by using the Orchestrator.

C. Redesign the application to be separated into these three layers: Presentation, API, and Data.

D. Create two virtual machines that each host an instance of the application and set up a cluster.

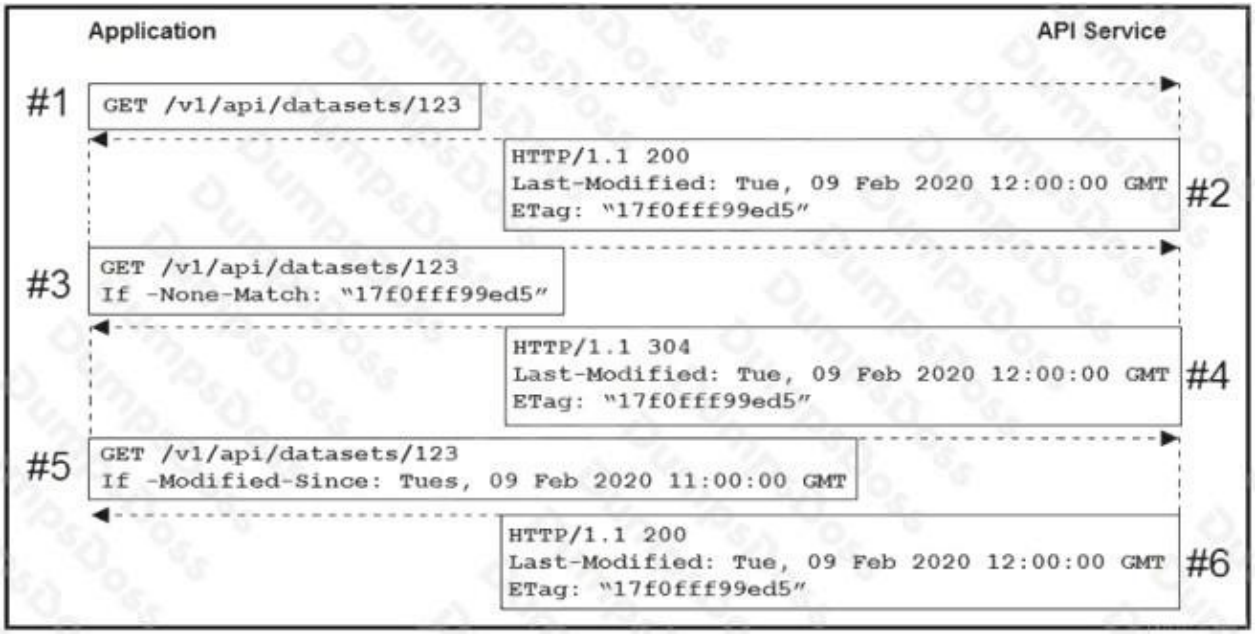
ANSWER: A

Explanation:

Implement retries with exponential backoff during HTTP API calls. is the appropriate resilience measure for the scope of one user request because each synchronous REST call in the request chain can encounter a temporary failure, such as a short network interruption, a connection reset, or a brief period of downstream overload. A retry allows the current request to recover when the failed operation is transient rather than permanent. Exponential backoff spaces subsequent attempts progressively farther apart, reducing the chance that many callers immediately retry at once and amplify load on an already stressed microservice.

In a microservices call chain, this behavior is applied at the client side of an outbound HTTP dependency, with carefully bounded retry counts and delays that fit the request timeout budget. The caller can therefore continue the same end-to-end user operation if a downstream service becomes available shortly afterward. This is a standard resiliency pattern for transient faults in distributed applications. The retry pattern and the importance of backoff are described in the Microsoft Azure Architecture Center Retry pattern. For HTTP behavior and the transient nature of certain server failures, see RFC 9110: HTTP Server Error 5xx.

QUESTION NO: 34



Refer to the exhibit. An application uses an API to periodically sync a large data set.

Based on the HTTP message sequence provided, which statements are true about the caching behavior seen in the scenario? (Choose two.)

A. The full dataset was transmitted to the client twice.

B. The dataset changed sometime between message #4 and #5.

C. A partial dataset was transmitted to the client in message #4.

D. The dataset did not change during the scenario.

E. Messages #3 and #5 are equivalent.

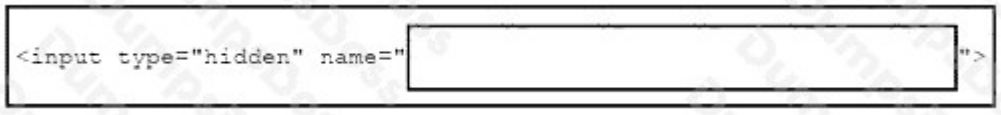
ANSWER: A D

Explanation:

The full dataset was transmitted to the client twice because the sequence shows two successful responses that carry the complete representation rather than a cache-validation-only response. A successful response containing the representation body transfers the data again even when that representation is identical to one obtained previously. The dataset did not change during the scenario because the cache validators shown in the exchange continue to identify the same version of the resource. Validators such as an entity tag allow a client and server to establish whether a stored representation corresponds to the current resource state. When the validator remains associated with the same resource version, the client can safely treat its cached dataset as current for that validation point. HTTP caching is intended to reduce unnecessary transfers, but a full response can still be sent when a request is not satisfied from cache or is not processed as a conditional validation. The relevant semantics for cache validation, stored responses, and validators are defined in RFC 9111: HTTP Caching. Conditional request handling and entity tags are also described in RFC 9110: HTTP Semantics.

QUESTION NO: 35

Refer to the exhibit.



A network engineer created a simple Python Flask application but must incorporate a CSRF token. Which code snippet must be added in the blank in the script to manually incorporate the token?

A. _access_tokenM value=M{{ csrf_token}}

B. _csrMoken" value="{{ csrf_grant()}}

C. _csrMoken" value="{{ csrf_token()}}

D. _xssjoken" value="{{ csrMoken}}

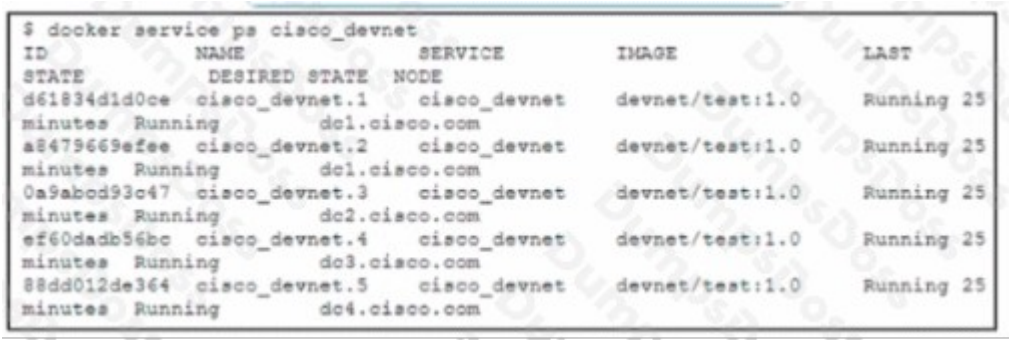
ANSWER: C

Explanation:

`<input type="hidden" name="csrf_token" value="{{ csrf_token() }}">` is the correct snippet for manually placing a CSRF token in an HTML form rendered by a Flask application using Flask-WTF CSRF protection. The hidden input ensures that the browser submits the generated token together with the form's normal fields, while the `csrf_token()` Jinja helper generates the token value associated with the current request/session context. On submission, Flask-WTF's `CSRFProtect` integration retrieves and validates that submitted value before allowing the protected state-changing request to proceed. This pattern is particularly useful when the form is written directly in an HTML/Jinja template rather than rendered from a Flask-WTF form object, where a call such as `form.hidden_tag()` would normally emit the hidden CSRF field automatically. The token field must use the expected `csrf_token` name and invoke the helper as a function so that an actual generated token, rather than a literal template reference, is sent. Flask-WTF documents this exact manual hidden-field approach for HTML forms at Flask-WTF CSRF Protection. This supports the broader CSRF defense guidance described in the OWASP Cross-Site Request Forgery overview.

QUESTION NO: 36

Refer to the exhibit.



```
$ docker service ps cisco_devnet
ID            NAME              SERVICE         IMAGE              LAST
STATE          DESIRED STATE  NODE
d61834d1d0ce  cisco_devnet.1    cisco_devnet    devnet/test:1.0     Running 25
minutes  Running        dc1.cisco.com
a8479669efee  cisco_devnet.2    cisco_devnet    devnet/test:1.0     Running 25
minutes  Running        dc1.cisco.com
0a9abcd93c47  cisco_devnet.3    cisco_devnet    devnet/test:1.0     Running 25
minutes  Running        dc2.cisco.com
ef60dadb56bc  cisco_devnet.4    cisco_devnet    devnet/test:1.0     Running 25
minutes  Running        dc3.cisco.com
88dd012de364  cisco_devnet.5    cisco_devnet    devnet/test:1.0     Running 25
minutes  Running        dc4.cisco.com
```

The cisco_devnet Docker swarm service runs across five replicas The development team tags and imports a new image named devnet'test 1 1 and requests that the image be upgraded on each container There must be no service outages during the upgrade process Which two design approaches must be used? (Choose two.)

A. Implement rolling upgrades by using the docker service update command.

B. Enable parallel upgrades by using the docker service update command.

C. Ensure that the service is hosted behind a VIP with no session persistence.

D. Update the restart policy of the containers to restart upon failure.

E. Ensure that the service replicas are set to a minimum of 5

ANSWER: A C

Explanation:

Implement rolling upgrades by using the docker service update command. is required because Docker Swarm performs a service update incrementally, replacing tasks in batches rather than removing every replica at once. The update behavior can be controlled with settings such as `--update-parallelism` and `--update-delay`, allowing the scheduler to keep healthy tasks serving traffic while replacement tasks using the new image are deployed and validated. For an availability-sensitive service, a small update batch size is the appropriate rolling-update design.

Ensure that the service is hosted behind a VIP with no session persistence. provides a single virtual service address for clients and lets Swarm distribute new connections among the currently available replicas. As individual tasks are replaced during the rolling update, requests can continue to reach replicas that remain healthy and active. Avoiding application dependence on a replica-specific persistent session enables a client to be served by another available task when a task is rotated. Together, incremental task replacement and VIP-based load distribution preserve service availability throughout the image upgrade. Docker documents Swarm rolling service updates at Docker Rolling Updates and service discovery with VIP load balancing at Docker Swarm Networking.

QUESTION NO: 37

An engineer is developing a client/server application for use by a very large number of clients. The engineer must optimize the application response time and lower the required network bandwidth when clients access the server. The engineer wants the server responses to be based on the resource version fetched by the clients. Which HTTP technique must be used?

A. middleware caching

B. API pagination

C. data compression

D. conditional requests

ANSWER: D

Explanation:

conditional requests are the HTTP mechanism that lets a client make a request conditional on the version of a resource it previously retrieved. The server supplies a validator for a representation, commonly an `ETag` value or a `Last-Modified` timestamp. When requesting the resource again, the client includes that validator in a header such as `If-None-Match` or `If-Modified-Since`. The server compares the supplied validator with the current resource version. If the representation has not changed, it can return `304 Not Modified` without sending the response body, allowing the client to reuse its stored copy. This substantially reduces transferred data and commonly improves perceived response time for large client populations. If the resource has changed, the server returns the current representation, enabling the client to refresh its cached version. Conditional requests can also use `If-Match` for state-changing operations when an update must apply only to a specific current version, helping prevent lost updates. HTTP defines these precondition headers and their evaluation rules in RFC 9110, Section 13. Practical validator and revalidation behavior is also described by MDN's Conditional Requests guide.

QUESTION NO: 38

Refer to the exhibit.

```
def init_tracer(service):
    logging.getLogger('').handlers = []
    logging.basicConfig(format='%(message)s', level=logging.DEBUG)
    config = Config(
        config={'sampler': {'type': 'const', 'param': 1,},'logging': True,},
        service_name=service,)
    return config.initialize_tracer()

base_url = 'https://sandboxdnac.cisco.com/'

        try:
            dnac = DNACenterAPI(username='devnetuser', password='Cisco123!',
                                base_url=base_url, version='1.3.3',
                                verify=False)
            print('auth passed')
        except Exception as e:
            print('failed')
    try:
        devices = dnac.devices.get_device_list()
        devices = devices['response']

        devices = [device['hostname'] for device in devices]
        print(devices)
    except Exception as e:
```

An application is created to serve the needs of an enterprise. Slow performance now impacts certain API calls, and the application design lacks observability. Which two commands improve observability and provide an output that is similar to the sample output? (Choose two.)

A)

```
dnac-tracer3 = init_tracer('dnac-tracer3')
```

B)

```
with dnac-tracer3.start_span('dnac-api-calls') as span:
```

C)

```
with tracer.start_span('dnac-api-calls') as span:
```

D)

```
dnac-tracer3.start_span('dnac-api-calls') as span:
```

E)

```
tracer = init_tracer('dnac-tracer3')
```

A. Option A

B. Option B

C. Option C

D. Option D

E. Option E

ANSWER: C E

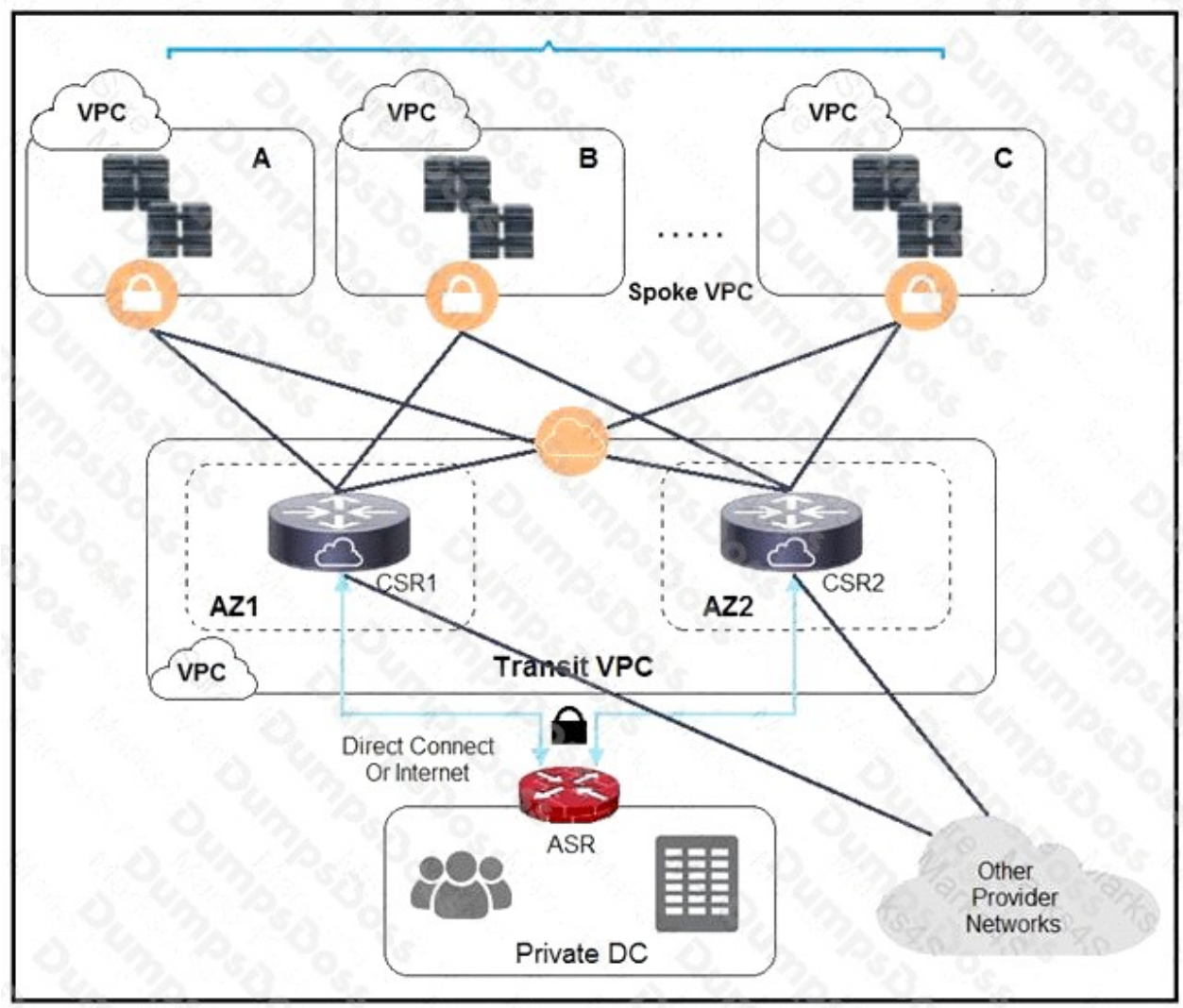
Explanation:

The commands represented by **Option C** and **Option E** are the appropriate selections because they add useful application-level observability for diagnosing slow API activity and generate output in the form shown in the exhibit. Effective observability requires telemetry that identifies what operation occurred and provides enough contextual information to correlate a request with its execution behavior. In an enterprise application, this enables developers and operators to isolate the API calls associated with elevated latency, inspect the relevant runtime details, and correlate events during troubleshooting rather than relying on an isolated symptom report.

Cisco's application-development guidance emphasizes using instrumentation and operational telemetry to make applications easier to monitor and troubleshoot. The selected commands support this practice by producing diagnostic information that can be inspected directly or collected by a logging and monitoring platform. Such telemetry is particularly valuable when performance degradation affects only a subset of calls, because request-specific records help establish which calls are slow and where to continue the investigation. Cisco DevNet resources on application development and APIs are available through Cisco DevNet Learning, and Cisco's observability guidance is available at Cisco Observability.

QUESTION NO: 39

Refer to the exhibit.



A company has extended networking from the data center to the cloud through Transit VPC.

Which two statements describe the benefits of this approach? (Choose two.)

A. Dynamic routing combined with multi-AZ- deployment creates a robust network infrastructure.

B. VPC virtual gateways provide highly available connections to virtual networks.

C. Dedicated VPC simplifies load balancing by combining internal and external web services.

D. VPC virtual gateways provide more secure connections to virtual networks.

E. Dedicated VPC simplifies routing by not combining this service with other shared services.

ANSWER: B D

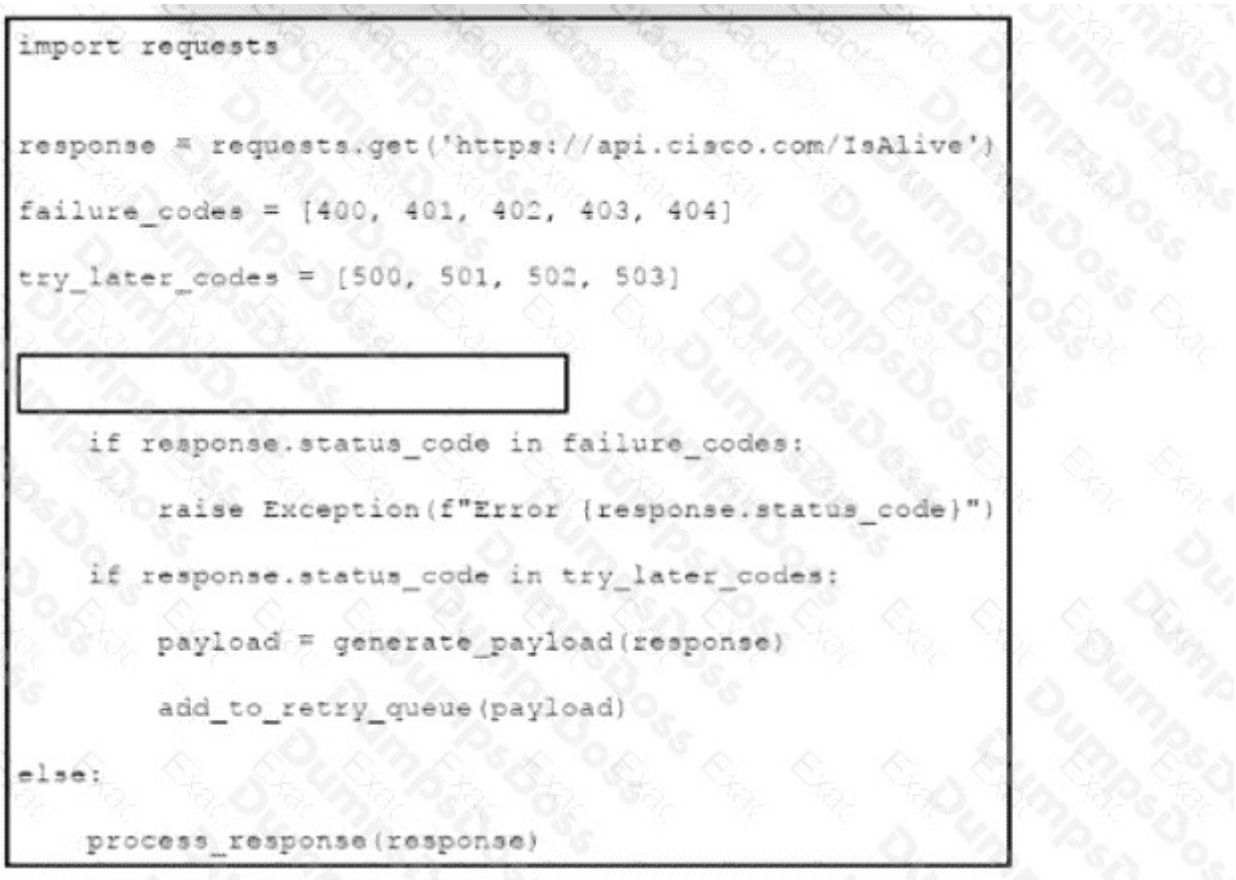
Explanation:

VPC virtual gateways provide highly available connections to virtual networks. A virtual private gateway is the AWS-side VPN endpoint for a VPC. AWS Site-to-Site VPN connections use redundant VPN tunnels, enabling resilient connectivity between the on-premises environment and VPC resources. In a Transit VPC design, these VPN connections provide the spoke VPCs with a managed and highly available attachment to the transit routing environment.

VPC virtual gateways provide more secure connections to virtual networks. Site-to-Site VPN connectivity through a virtual private gateway uses IPsec encryption to protect traffic sent between the data center and AWS VPCs over the public network. The Transit VPC pattern centralizes this encrypted connectivity and routing through the transit environment while retaining isolated VPC network boundaries. This allows workloads in separate VPCs to communicate with on-premises networks through controlled, encrypted paths rather than requiring direct, independently managed connectivity for every network relationship.

AWS documents the redundancy and tunnel behavior of Site-to-Site VPN connections in its AWS Site-to-Site VPN documentation and describes virtual private gateways in the virtual private gateway guide.

QUESTION NO: 40



```
import requests

response = requests.get('https://api.cisco.com/IsAlive')

failure_codes = [400, 401, 402, 403, 404]

try_later_codes = [500, 501, 502, 503]

[                                        ]

    if response.status_code in failure_codes:

        raise Exception(f"Error {response.status_code}")

    if response.status_code in try_later_codes:

        payload = generate_payload(response)

        add_to_retry_queue(payload)

else:

    process_response(response)
```

Refer to the exhibit. An engineer is developing a Python script to check if an API is live If the API is not live, the script must evaluate the response code and either raise an exception or add the request to a queue for retry. Which code snippet must be placed in the plank in the code to complete the script?

A.

B.

C.

D.

ANSWER: D

Explanation:

The code snippet shown in the fourth image is correct because it handles an unsuccessful API response according to the response-status category. HTTP client-error responses, represented by status codes in the 400 range, indicate that the request is invalid or cannot be fulfilled as submitted. Retrying the same request without correcting it is generally not useful, so the script must raise an exception and allow the caller or application to handle the failure. Server-error responses, represented by status codes in the 500 range, indicate that the server failed to process an otherwise valid request. These failures can be transient; therefore, placing the request in a retry queue is an appropriate resilience pattern. This conditional handling lets the application distinguish permanent request failures from potentially recoverable service failures rather than treating every non-successful response identically. Python's Requests library exposes the HTTP status through the response object and provides exception handling facilities for failed HTTP requests. HTTP status-code classes and their intended semantics are defined by the HTTP specification. See the Requests response status-code documentation and RFC 9110 HTTP status codes.

QUESTION NO: 41

Which two statements describe NETCONF operations? (Choose two.)

A. NETCONF commonly uses SSH as its secure transport.

B. NETCONF configuration data is modeled with YANG.

C. NETCONF requires SNMPv2c community strings for authentication.

D. NETCONF requests use JSON-RPC as the required message format.

E. NETCONF can modify configuration only through device CLI commands.

ANSWER: A B

Explanation:

NETCONF commonly uses SSH as its secure transport and **NETCONF configuration data is modeled with YANG**. NETCONF is a standards-based protocol for installing, manipulating, and deleting the configuration of network devices. SSH is the mandatory-to-implement secure transport in the NETCONF over SSH specification, providing authentication, confidentiality, and integrity for management sessions.

NETCONF operations act on configuration datastores and exchange structured XML-encoded messages. YANG provides the data-modeling language used to define the configuration and operational data that NETCONF clients retrieve or modify. The protocol and its datastore operations are defined in RFC 6241, while the YANG language is specified in RFC 7950.

QUESTION NO: 42

Which two situations are flagged by software tools designed for dependency checking in continuous integration environments, such as OWASP? (Choose two.)

A. publicly disclosed vulnerabilities related to the included dependencies

B. mismatches in coding styles and conventions in the included dependencies

C. incompatible licenses in the included dependencies

D. test case failures introduced by bugs in the included dependencies

E. buffer overflows to occur as the result of a combination of the included dependencies

ANSWER: A C

Explanation:

Dependency-checking tools used in CI perform software-composition analysis on third-party libraries and packages included in an application. They identify publicly disclosed vulnerabilities related to the included dependencies by matching the dependency's package, version, and identifiers against vulnerability data sources such as the National Vulnerability Database. This gives development teams actionable information about known CVEs before software is released, enabling upgrades, removals, or mitigations to be incorporated into the delivery pipeline.

Incompatible licenses in the included dependencies are also an important dependency-governance finding. Open-source components carry license obligations that can affect redistribution, source-code disclosure, patent terms, or commercial use. License-policy checking allows an organization to identify dependencies whose licenses conflict with its legal or distribution requirements, then replace, approve, or otherwise manage those components before release. These vulnerability and license findings are both based on the inventory of externally sourced components and are well suited to automated CI checks. OWASP describes dependency analysis as identifying project dependencies with known, publicly disclosed vulnerabilities; OWASP's Software Component Verification Standard also addresses component inventory and license considerations. See OWASP Dependency-Check and OWASP Software Component Verification Standard.

QUESTION NO: 43

Which two types of storage are supported for app hosting on a Cisco Catalyst 9000 Series Switch? (Choose two.)

A. external USB storage

B. internal SSD

C. CD-ROM

D. SD-card

E. bootflash

ANSWER: A B

Explanation:

Cisco Catalyst 9000 Series switches support application hosting through Cisco IOS XE and the IOx application-hosting framework. For this feature, application images, package data, and persistent application storage can be placed on an internal SSD when the platform provides one. The SSD offers dedicated, higher-capacity local storage intended for hosted applications and their associated data.

External USB storage is also supported as an application-hosting storage location. A USB device can provide removable local capacity for application packages and application data, which is especially useful where an internal SSD is not installed or where additional portable storage is needed. Before deploying an application, the selected storage media must be available to IOS XE and the application-hosting configuration must identify the appropriate storage location. Cisco's Catalyst 9000 application-hosting documentation describes internal SSD and external USB flash storage as supported media for this purpose. See the Cisco IOS XE Catalyst 9000 Application Hosting Configuration Guide and Cisco's IOx Configuration Guide.

QUESTION NO: 44

A local data center deployment using Cisco NX-OS switches is scaling and requires automatic configuration at scale. Configuration management must be set up for a Cisco NX-OS switch by using Ansible. The Ansible control server is currently located on a different subnet than the switch. The solution has these requirements:

- The transport protocol used must be encrypted.
- The connections must originate from a server in the same local network. - Enable mode must be supported.

Which connectivity method must be used?

A. HTTP through a web proxy.

B. HTTPS through a repository and ansible_become method for privilege escalation.

C. SSH through a bastion host and ansible_become method for privilege escalation.

D. XML-RPC through a web proxy.

ANSWER: C

Explanation:

SSH through a bastion host and ansible_become method for privilege escalation is the appropriate connectivity method. Ansible uses its network_cli connection plugin to manage Cisco NX-OS devices over SSH, providing an encrypted management transport. A bastion host, also called a jump host, can be placed in the switch's local network. The Ansible control node establishes an SSH session to that host, and the bastion then initiates the device-facing SSH connection locally. This satisfies the requirement that the connection reaching the NX-OS switch originate from a server in the same local network, without requiring the control node itself to reside there.

For network devices, Ansible supports privilege escalation with `ansible_become`. With the network_cli connection type, the `enable` become method enters the device's privileged EXEC, or enable, mode when commands require that privilege level. The inventory or play variables can define the SSH connection, bastion/jump-host settings, and enable credentials while keeping the operational workflow centralized on the Ansible control server. See the Ansible network_cli connection documentation and Cisco's NX-OS programmability documentation for supported automation and remote-management approaches.

QUESTION NO: 45

An application requires SSL certificates signed by an intermediate CA certificate. The crt files must be available to the application:

• The root CA certificate is root_certrficate.crt.

• The intermediate CA certificate is intermediate_certificate.crt

• The application-specific SSL certificate is crt_certificate.crt.

Which Bash command outputs the certificate bundle as a .pern file?

A)

```
cat root certificate.crt intermediate certificate.crt >
certificate bundle.pem
```

B)

```
cat root certificate.crt intermediate certificate.crt
crl certificate.crt > certificate bundle.pem
```

C)

```
cat crl certificate.crt intermediate certificate.crt
root certificate.crt > certificate bundle.pem
```

D)

```
cat intermediate_certificate.crt root_certificate.crt >
certificate_bundle.pem
```

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: D

Explanation:

The correct command is the one that concatenates the application certificate first, followed by the intermediate CA certificate and then the root CA certificate, redirecting the resulting output into the bundle file. In practical form, that is `cat crt_certificate.crt intermediate_certificate.crt root_certrficate.crt > certificate_bundle.pem`. A PEM certificate bundle is a text file containing multiple PEM-encoded certificates, each with its own `BEGIN CERTIFICATE` and `END CERTIFICATE` boundaries. The server or application certificate must appear first because it identifies the endpoint that will use the bundle. The issuing intermediate certificate follows, allowing a client to establish the certificate path from that endpoint certificate to its issuer. Including the root certificate last provides the complete supplied chain when the application specifically requires all certificate files to be available together. The shell `cat` utility writes each input file to standard output in the order provided, while `>` redirects that combined output to a new PEM

bundle. OpenSSL documentation describes certificate-chain construction and the relationship between a leaf certificate, intermediates, and trust anchors: OpenSSL certificate verification documentation. PEM is also the standard textual encoding used by OpenSSL for certificate input and output: OpenSSL x509 documentation.

QUESTION NO: 46

Which action enhances end-user privacy when an application is built that collects and processes the location data from devices?

A. Pepper the MAC address for each device.

B. Salt the MAC address for each device.

C. Implement an algorithmic information theoretic loss to the MAC address for each device.

D. Use the network device serial number to encrypt the MAC address for each device.

ANSWER: B

Explanation:

Salt the MAC address for each device. A MAC address is a persistent device identifier and can become personal data when it is associated with location observations, timestamps, or a user account. Before storing or processing that identifier, the application should apply a unique, randomly generated salt as part of a one-way cryptographic hash process. The resulting value is a pseudonymous identifier that allows the application to use a device-specific reference without retaining the original MAC address in its operational data set.

Using a distinct salt prevents the same MAC address from producing a universally reusable hash value. This makes precomputed lookup tables and straightforward correlation with identifiers exposed by another system substantially more difficult. The salt must be generated with a cryptographically secure random source, protected alongside the associated data as appropriate, and combined with a modern one-way hash or keyed derivation function; salting alone is not encryption. This approach supports data minimization and pseudonymization principles for location-aware applications while reducing the privacy impact if collected records are disclosed. Cisco's privacy principles describe its commitment to protecting personal data, and NIST explains the security value of unique salts in resisting offline lookup attacks: Cisco Privacy Statement; NIST SP 800-63B.

QUESTION NO: 47

Refer to the exhibit.

```
import requests
url = "https://api.meraki.com/api/v0/networks/2/clients"
payload = None
headers = {
    "Content-Type": "application/json",
    "Accept": "application/json",
    "X-Cisco-Meraki-API-Key": "1b87b584fb5411eaadc10242ac120002"
}
response = requests.request('GET', url, headers=headers, data = payload)
print(response.text.encode('utf8'))
```

A Python script must list network clients in the Cisco Meraki API that have used a network with an ID of 2 The number of client entries per returned page is restricted to 1.000 according to the API specification Network 2 has 2.500 clients What must be added where the code is missing to print the content of each response?

A)

```
if response.links.get('next'):
    url = response.links['next']['url']
    response = requests.get(url)
    print(response.text)
```

B)

```
url = "https://api.meraki.com/api/v0/networks/2/
clients?startingAfter=1000"
response = requests.get(url)
print(response.text)
```

C)

```
url = "https://api.meraki.com/api/v0/networks/2/
clients?perPage=1600"
response = requests.get(url)
print(response.text)
```

D)

```
if response.links.get('next'):
    url = links['url']
    response = requests.get(url)
    print(response.text)
```

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: A

Explanation:

The correct code uses pagination by following the URL supplied for the next page in the HTTP response's `Link` header, while printing each page's response content before requesting the following page. Cisco Meraki list endpoints impose a maximum page size, so a request with `perPage=1000` can return only the first 1,000 client records. With 2,500 clients, the script must process three responses rather than assuming that one response contains every client.

The Python `requests` library exposes parsed link-header relationships through `response.links`. The pagination logic therefore checks whether a `next` relationship exists, retrieves its URL, makes a new authenticated GET request to that URL, and repeats until no next-page link is present. Printing the content during every loop iteration ensures that the initial page and every subsequent page are output. This approach also avoids manually constructing page numbers or cursor values, because the Meraki API provides the authoritative continuation URL.

Cisco documents the Network Clients endpoint and its pagination parameters at Cisco Meraki API: Get Network Clients. General Meraki pagination behavior and Link-header navigation are described in Cisco Meraki API Pagination.

QUESTION NO: 48

Refer to the exhibit.

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority: fake-ca-file
    server: https://1.2.3.4
  name: development
- cluster:
    insecure-skip-tls-verify: true
    server: https://5.6.7.8
  name: scratch
contexts:
- context:
    cluster: development
    namespace: frontend
    user: developer
  name: dev-frontend
- context:
    cluster: development
    namespace: storage
    user: developer
  name: dev-storage
- context:
    cluster: scratch
    namespace: default
    user: experimenter
  name: exp-scratch
current-context: ""
kind: Config
preferences: {}
users:
- name: developer
  user:
    client-certificate: fake-cert-file
    client-key: fake-key-file
- name: experimenter
  user:
```

```
- name: experimenter
  user:
    password: some-password
    username: exp
```

A kubeconfig file to manage access to Kubernetes clusters is shown. How many Kubernetes clusters are defined in the file, and which cluster FS accessed using username/password authentication rather than using a certificate?

A. three clusters; scratch

B. three clusters: development

C. two clusters; development

D. two clusters: scratch

ANSWER: D

Explanation:

two clusters: scratch is correct. In a kubeconfig file, the number of managed Kubernetes clusters is determined by the entries in the top-level `clusters` list, not by the number of users, contexts, or current-context declarations. The exhibit defines two cluster entries. Authentication is configured separately in the top-level `users` list, while each entry in `contexts` associates a cluster name with a user name. Following the context mapping for `scratch` identifies the user configuration used when that cluster is selected. That referenced user uses username/password credentials, represented by fields such as `username` and `password`, rather than client-certificate authentication fields such as `client-certificate` and `client-key`. Thus, the kubeconfig defines two clusters overall, and `scratch` is the cluster accessed through the username/password user entry. Kubernetes documents that kubeconfig contexts bind a cluster, a user, and optionally a namespace, and that user entries can contain credentials including basic-auth information or certificate material. See the Kubernetes kubeconfig documentation and the Kubeconfig API reference.

QUESTION NO: 49

An application has initiated an OAuth authorization code grant flow to get access to an API resource on behalf of an end user.

Which two parameters are specified in the HTTP request coming back to the application as the end user grants access? (Choose two.)

A. access token and a refresh token with respective expiration times to access the API resource

B. access token and expiration time to access the API resource

C. redirect URI a panel that shows the list of permissions to grant

D. code that can be exchanged for an access token

E. state can be used for correlation and security checks

ANSWER: D E

Explanation:

In an OAuth 2.0 authorization code grant, once the resource owner approves the requested authorization, the authorization server redirects the user agent back to the client application's registered redirect URI. The redirect request includes a `code` parameter: a short-lived authorization code that the application subsequently sends to the token endpoint, together with its client authentication and redirect URI where applicable, to obtain an access token. This separation keeps tokens out of the front-channel redirect and lets the authorization server authenticate the client before issuing tokens.

The redirect also returns `state` when the client included a state value in the original authorization request. The client must compare the returned value to the value associated with the user's initiating session. This correlation protects the flow against cross-site request forgery and helps the application identify the correct pending authorization transaction. Therefore, the parameters in the successful authorization response are the code that can be exchanged for an access token and the state value used for correlation and security checks. See RFC 6749, Authorization Response and RFC 6749, Cross-Site Request Forgery.

QUESTION NO: 50

What are two principles according to the build, release, run principle of the twelve-factor app methodology?

(Choose two.)

A. Code changes are able to be made at runtime.

B. Separation between the build, release, and run phases.

C. Releases should have a unique identifier.

D. Existing releases are able to be mutated after creation.

E. Release stage is responsible for compilation of assets and binaries.

ANSWER: B C

Explanation:

The build, release, run factor requires a strict separation of the stages used to deliver an application. A build stage converts source code and its declared dependencies into an executable build artifact. A release stage combines that immutable build with the deployment-specific configuration needed to operate it, such as the target environment's configuration values. Finally, the run stage executes the release in the selected environment. Maintaining this separation makes delivery repeatable and gives teams a clear, traceable progression from source code to a running application.

"Separation between the build, release, and run phases" is therefore a core principle because each phase has a distinct purpose and should not be blended. "Releases should have a unique identifier" is also correct because a release is an identifiable combination of a build and configuration. The Twelve-Factor App guidance states that every release should have a unique release ID, such as a timestamp or incrementing number, and that releases are intended to be immutable. Unique IDs make it possible to determine exactly what version and configuration are running, support reliable rollback to a prior release, and improve operational auditability. See the authoritative Twelve-Factor App: Build, release, run guidance and the related Twelve-Factor App methodology.

QUESTION NO: 51

A team of developers created their own CA and started signing certificates for all of their IoT devices.

Which action will make the browser accept these certificates?

A. Install a TLS instead of SSL certificate on the IoT devices.

B. Set the private keys 1024-bit RSA.

C. Preload the developer CA on the trusted CA list of the browser.

D. Enable HTTPS or port 443 on the browser.

ANSWER: C

Explanation:

Preload the developer CA on the trusted CA list of the browser. A browser accepts a server or device certificate only when it can validate the certificate chain from that certificate through any intermediate certificates to a trusted root certificate authority. Because the developers created a private CA, its root certificate is not included in the public root stores that browsers trust by default. Installing that CA's public root certificate in the relevant browser or operating-system trust store establishes it as a trust anchor. The browser can then validate certificates issued by that CA, provided the certificates are otherwise valid for the connection, including matching the requested hostname or IP address where applicable, being within their validity period, and containing appropriate key usage and extended key usage values. This approach is common for controlled enterprise, lab, and private IoT deployments, where administrators manage the devices and browser trust configuration. Mozilla documents how certificate authorities can be managed in Firefox's certificate settings, including importing a CA certificate for website identification: Firefox certificate authority settings. Chrome also relies on trusted root certificates provided through the operating system or managed enterprise policies, as described in Chrome Enterprise certificate management.

QUESTION NO: 52

Which transport protocol is used by gNMI?

A. HTTP/2

B. HTTP 1.1

C. SSH

D. MQTT

ANSWER: A

Explanation:

HTTP/2 is correct because gNMI defines its remote procedure call interface using gRPC, and gRPC uses HTTP/2 as its underlying transport protocol. HTTP/2 provides the capabilities gNMI needs for efficient network telemetry and configuration operations, including multiplexed bidirectional streams, flow control, header compression, and persistent connections. In practical gNMI deployments, a client establishes a gRPC connection to the network device and invokes gNMI service operations such as Capabilities, Get, Set, and Subscribe through that connection. The Subscribe operation particularly benefits from HTTP/2 streaming because telemetry updates can be delivered continuously over an established session rather than requiring a separate request for every update. gNMI commonly runs over TLS-secured gRPC, but TLS supplies transport security rather than replacing HTTP/2 as the protocol used for the gRPC transport. The OpenConfig gNMI specification explicitly states that gNMI uses gRPC for its service definition and describes its RPC methods and streaming behavior. The gRPC protocol documentation further identifies HTTP/2 as the transport foundation for gRPC communication. See the OpenConfig gNMI specification and the gRPC over HTTP/2 protocol documentation.

QUESTION NO: 53

Refer to the exhibit.

```
import requests
import time
import json

class Connection:
    def __init__(self, config):
        self._config = config
        self._session = None
        self._retries = 0
        self._MAX_RETRIES = 12

    def _setupSession(self):
        self._retries = 0
        if self._session is None:
            self._session = requests.Session()
        return

    def get(self, url, params=None):
        self._setupSession()
        resp = self._session.get(self._config.host + url, verify=False, params=params)
        if resp.status_code == 200:
            return json.loads(resp.content.decode('utf-8'))
            [                                              ]
            self._retries += 1
            exp_backoff = (2**(self._retries+3))/1000
            time.sleep(exp_backoff)
            self.get(url=url, params=params)
        return resp
```

A network engineer must integrate error handling tor time-outs on network devices using the REST interface. Which line of code needs to be placed on the snippet where the code is missing to accomplish this task?

A. elif resp.status_code == 429 or serf._retries < selff._MAX_RETRIES:

B. elif resp.status_code == 404 or self, retries < self._MAX_RETRIES:

C. elif resp.status_code == 429 and self .retries < self._MAX_RETRIES:

D. elif resp.status_code == 404 and self._relries < self._MAX_RETRIES:

ANSWER: C

Explanation:

`elif resp.status_code == 429 and self .retries < self._MAX_RETRIES:` is the appropriate conditional because it handles an HTTP 429 response only while the client remains within its configured retry limit. HTTP status 429, "Too Many Requests," tells a REST client that the service is temporarily refusing additional requests because a rate limit has been exceeded. A resilient network-automation client should treat this as a transient condition: it can wait according to the server's retry guidance, increase its retry counter, and attempt the request again. The retry-limit comparison is essential because it bounds the recovery behavior and prevents an endless retry loop if the device or API remains unavailable or continues throttling requests. This pattern supports dependable REST integrations by making transient service-pressure conditions recoverable while keeping the application's request behavior controlled and predictable. The definition and intended handling of HTTP 429 are specified in RFC 6585, section 4. Cisco's API guidance also emphasizes respecting rate limits and retry-related response information when consuming REST APIs; see the Cisco Intersight rate-limiting documentation.

QUESTION NO: 54

Which two controls protect sensitive configuration values used by a cloud-native application? (Choose two.)

A. Store secrets in a dedicated secret-management service.

B. Embed credentials in a Dockerfile to simplify deployment.

C. Grant workloads access using least-privilege identities.

D. Commit production API tokens to the source repository.

E. Send secrets as URL query parameters.

ANSWER: A C

Explanation:

Store secrets in a dedicated secret-management service and **grant workloads access using least-privilege identities** are appropriate controls. A dedicated secret-management service centralizes secure storage, access auditing, rotation, and distribution of values such as database credentials, API tokens, and private keys. Sensitive values should not be embedded in source code, container images, or publicly readable configuration files.

Workloads should authenticate with an identity that has permission to retrieve only the specific secrets required for their function. Least-privilege access limits the impact of a compromised workload and reduces unnecessary exposure of credentials. Secret values should also be protected in transit and rotated according to organizational policy. The Twelve-Factor App configuration guidance recommends separating configuration from code, and Kubernetes documents secure handling considerations at Kubernetes Secrets.

QUESTION NO: 55

Which two data encoding techniques are supported by gRPC? (Choose two.)

A. XML

B. JSON

C. ASCII

D. ProtoBuf

E. YAML

ANSWER: B D

Explanation:

JSON and ProtoBuf are supported data encoding techniques for gRPC. Protocol Buffers (ProtoBuf) are gRPC's primary and default interface-definition and message-serialization format. A `.proto` service definition specifies RPC methods and message schemas, and the Protocol Buffers compiler generates strongly typed client and server code for supported languages. ProtoBuf's compact binary representation and schema-based serialization are central to gRPC's efficient, cross-language communication model.

gRPC also supports JSON encoding through JSON transcoding or JSON-capable codecs, allowing HTTP/JSON clients to interact with gRPC APIs without requiring those clients to serialize Protocol Buffers directly. This is particularly useful when exposing an API to browser-based applications or external consumers that use conventional REST-style JSON payloads, while retaining gRPC service definitions and backend implementations. Cisco developers working with API integrations should therefore recognize ProtoBuf for native gRPC communication and JSON for interoperable HTTP/JSON access patterns. The gRPC documentation identifies Protocol Buffers as the default IDL and message format and documents JSON transcoding for HTTP APIs. See gRPC Introduction and gRPC documentation.

QUESTION NO: 56

Given an application that implements a basic search function as Well as a video upload function , which two load-balancing approaches

optimize the application's user experience? (Choose two.)

A. Video upload requests should be routed to the endpoint using an intermediate hop.

B. Video upload requests should be routed to the endpoint with highest data throughput.

C. Video upload requests should be routed to the endpoint with lowest round-trip latency.

D. Search requests should be routed to the endpoint with lowest round-trip latency.

E. Search requests should be routed to the endpoint with highest data throughput.

ANSWER: B D

Explanation:

Video upload requests should be routed to the endpoint with highest data throughput because uploads transfer comparatively large amounts of data and are therefore primarily constrained by available bandwidth and sustained transfer capacity. Selecting an endpoint that can provide greater throughput reduces the time required to complete an upload, makes progress more consistent, and improves the experience for users submitting large video files. Search requests should be routed to the endpoint with lowest round-trip latency because a search interaction is typically request-response oriented: the user waits for a query to reach the service and for results to return. Minimizing round-trip delay directly reduces perceived response time and makes the interface feel responsive, even when the response payload is relatively small. This distinction is an application-aware load-balancing principle: traffic should be directed according to the performance metric most important to that application flow. Cisco describes application-aware routing as selecting paths based on measured characteristics such as latency and available bandwidth, enabling traffic to use the path best suited to its requirements. See Cisco's Application-Aware Routing documentation and Cisco's SD-WAN overview.

QUESTION NO: 57

Which two types of organization are subject to GDPR? (Choose two.)

A. only organizations that operate outside the EU

B. any organization that offers goods or services to customers in the EU

C. only organizations that have offices in countries that are part of the EU

D. any organization that operates within the EU

E. only organizations that physically reside in the EU

ANSWER: B D

Explanation:

GDPR applies to any organization that operates within the EU when it processes personal data in the context of activities of an establishment in the Union. Its scope is not limited to companies headquartered in an EU member state; an establishment can include a stable and effective presence through which relevant processing activities occur. GDPR also applies extraterritorially to any organization that offers goods or services to customers in the EU, provided that the offering is directed at individuals in the Union. This means organizations located outside the EU can fall within GDPR's scope when they target EU-based individuals with products or services, whether those services are paid or provided free of charge. These jurisdictional rules are set out in Article 3, which establishes GDPR's territorial scope and is central to determining whether an organization must comply with the regulation. Cisco likewise notes that GDPR can apply both to organizations established in the EU and to organizations outside it that offer goods or services to, or monitor the behavior of, people in the EU. See Regulation (EU) 2016/679, Article 3 and Cisco's GDPR compliance overview.

QUESTION NO: 58

A bot has been created to respond to alarm messages. A developer is now creating a Webhook to allow the bot to respond to messages.

Which format allows the Webhook to respond to messages for the bot within Webex?

A. GET /messages?personId=me&roomId=NETWORK_STATUS Authorization: Bearer THE_BOTS_ACCESS_TOKEN

B. GET /messages?mentionedPeople=me&roomId=NETWORK_STATUS Authorization: Bearer THE_BOTS_ACCESS_TOKEN

C. GET /messages?mentionedBot=me&roomId=NETWORK_STATUS Authorization: Bearer THE_BOTS_ACCESS_TOKEN

D. GET /messages?botId=me&roomId=NETWORK_STATUS Authorization: Bearer THE_BOTS_ACCESS_TOKEN

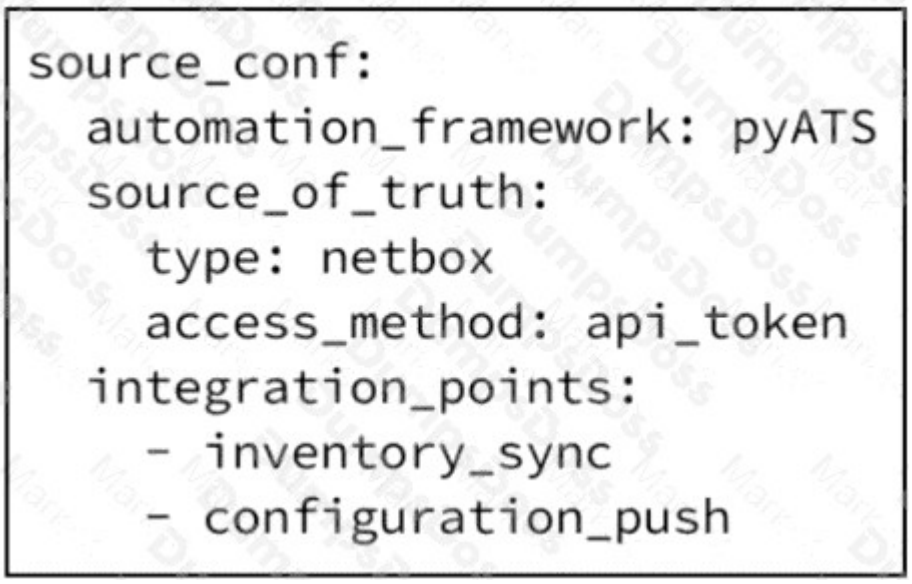
ANSWER: B

Explanation:

`GET /messages?mentionedPeople=me&roomId=NETWORK_STATUS Authorization: Bearer THE_BOTS_ACCESS_TOKEN` is the correct format because Webex supports the `mentionedPeople` query parameter when retrieving messages. For a bot access token, the special value `me` resolves to the authenticated bot's own identity. Combining it with `roomId=NETWORK_STATUS` limits the message query to the specified Webex space and identifies messages in which that bot was mentioned. This is useful for a bot that should react to direct requests, such as alarm-related commands, rather than processing every message posted in the space. In a production webhook workflow, Webex sends a notification when a message is created; the application then uses the message identifier supplied in that notification to retrieve message details through the Messages API. Filtering by the bot mention is an established Webex API pattern for bot interactions and ensures that the bot can determine whether a message was explicitly addressed to it. The request must include the bot access token in the HTTP Authorization bearer header so Webex can authenticate the bot and apply its access permissions. Cisco documents the Messages API query parameters, including `mentionedPeople` and use of `me`, at Webex List Messages API. Webhook event delivery is documented at Webex Webhooks API.

QUESTION NO: 59

Refer to the exhibit.



A network engineer is configuring automation using a real-time audit controller and plans to integrate information stored through an API-based resource catalog. The goal is to ensure that the chosen solution pulls device records directly into the automation workflow. The current framework provides flexible options for connecting with various data stores and external resources. Which integration step enables collection of accurate device attributes for orchestration?

A. Enable a resource catalog fetcher.

B. Attach a telemetry aggregation point.

C. Add a role-based access entry.

D. Extend a custom command launcher.

ANSWER: A

Explanation:

Enable a resource catalog fetcher. A fetcher is the integration component responsible for querying the external resource catalog and bringing the returned device records into the automation system's usable inventory or workflow context. By calling the catalog API, it can obtain the source-of-truth attributes needed for orchestration, such as device identity, platform information, management addressing, site or tenant association, and other metadata used to select and parameterize automation actions. This supports accurate, current orchestration because the workflow uses records retrieved from the catalog rather than relying on manually maintained or stale local values. In Cisco automation solutions, external systems are commonly integrated through APIs and adapters so that inventory and service information can be consumed programmatically by automation logic. The fetch operation is therefore the step that establishes the required data-collection path from the catalog to the orchestration workflow. Cisco NSO provides API-driven orchestration and integration capabilities for consuming external data and coordinating network services; see the Cisco NSO Developer Documentation. Cisco also describes API-based integration patterns and automation capabilities in the Cisco DevNet documentation portal.

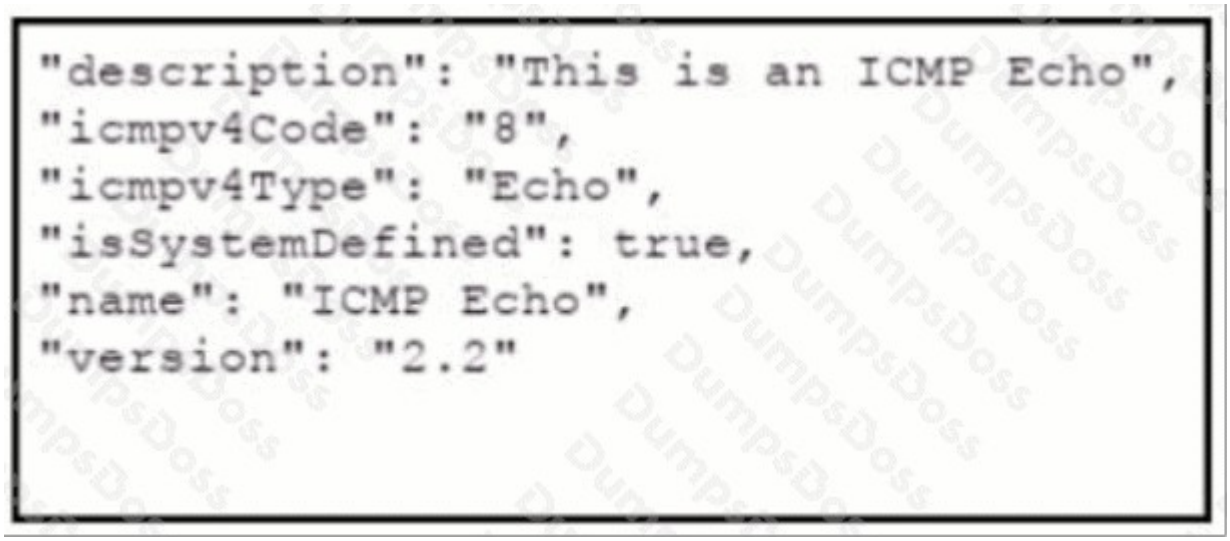
QUESTION NO: 60

Refer to the exhibits which show the documentation associated with the create port object API cafl in Cisco Firepower Threat Defense, and a cURL command. Which data payload completes the cURL command to run the APi call?

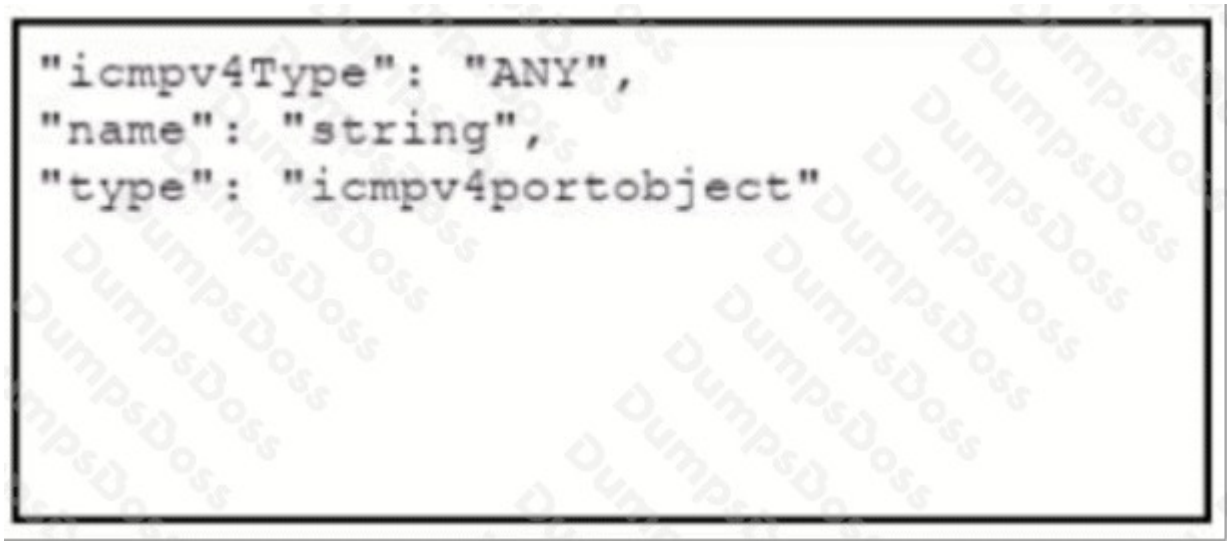
A)

"description": "string",
"icmpv4Code": "ANY_IPV4",
"icmpv4Type": "ANY",
"id": "string",
"isSystemDefined": "string",
"name": "string",
"type": "icmpv4portobject",
"version": "string"

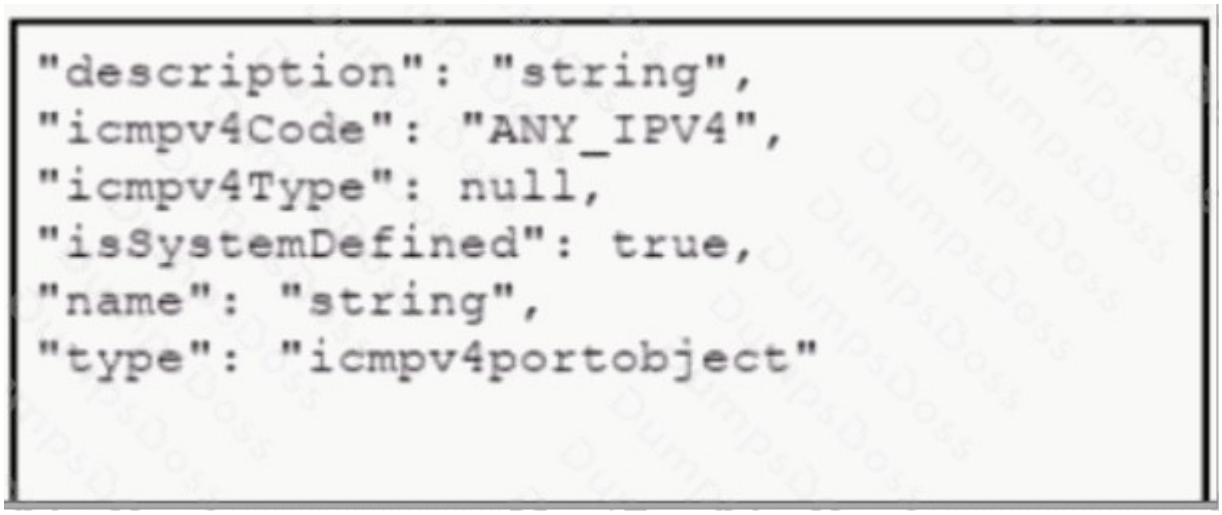
B)


"description": "This is an ICMP Echo",
"icmpv4Code": "8",
"icmpv4Type": "Echo",
"isSystemDefined": true,
"name": "ICMP Echo",
"version": "2.2"

C)


"icmpv4Type": "ANY",
"name": "string",
"type": "icmpv4portobject"

D)



A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: C

Explanation:

The payload shown in the selected response is correct because a Firepower Device Manager create-port-object request must send a JSON representation of a protocol-and-port object in the HTTP request body. The body identifies the object with the API object type and supplies the fields that define the port object, including its name, transport protocol, and port value or range. This structure enables Firepower Threat Defense to create a reusable network-object entry that can subsequently be referenced by access-control and other policy configuration APIs. The payload must also be valid JSON and match the schema expected by the create operation; the URL, authorization bearer token, and JSON content type in the cURL command provide the request context, while the payload supplies the resource to be created. Cisco's Firepower Threat Defense REST API reference documents the available object endpoints, resource schemas, and request-body requirements for configuration operations. Cisco also describes the REST API workflow, including authenticating and submitting JSON request bodies, in its API documentation. See the Cisco Firepower Threat Defense API Reference and the Cisco Secure Firewall Threat Defense REST API documentation.

QUESTION NO: 61

What are two advantages of using model-driven telemetry, such as gRPC. instead of traditional telemetry gathering methods? (Choose two.)

A. all data is ad-hoc

B. efficient use of bandwidth

C. no overhead

D. decentralized storage of telemetry

E. continuous information with incremental updates

ANSWER: B E

Explanation:

efficient use of bandwidth and **continuous information with incremental updates** are advantages of model-driven telemetry. Instead of repeatedly polling a device for large sets of operational data, a collector subscribes to specific modeled

data paths. The network device then streams the requested data using an efficient, structured encoding such as Protocol Buffers over a persistent transport session. This avoids the repetitive request-and-response exchanges associated with conventional polling and reduces the amount of duplicated data sent across the network.

Model-driven telemetry also supports continuous, near-real-time delivery. Subscriptions can be configured for periodic sampling or change-based updates, allowing the collector to receive new values as operational state changes. Incremental updates provide timely visibility into counters, interface state, routing information, and other modeled operational data without requiring a complete retrieval each time. This streaming approach helps monitoring systems detect changes faster while scaling more effectively across many devices and telemetry paths. Cisco describes telemetry subscriptions, encodings, and streaming transport behavior in its IOS XR Telemetry Configuration Guide; the underlying remote procedure call transport is documented by the gRPC introduction.

QUESTION NO: 62

```
import requests, json
from flask import Flask, request

app = Flask(__name__)

db_connection = psycopg2.connect()

@app.route('/message', methods=["GET"])
def message():

    messages = []
    with connection.cursor() as cursor:
        query = 'SELECT * FROM messages WHERE status = UNREAD'
        cursor.execute(query)
        messages = cursor.fetchall()
    return json.dumps(messages)

if __name__ == '__main__':
    app.run(debug=True)
```

Refer to the exhibit. A Python API server has been deployed. Based on metrics and logs, increased load has been noticed. Which two approaches must be taken to optimize API usage on the server? (Choose two.)

A. Include the ETag header in the API response

B. Include the Last-Modified header in the API response.

C. Change the query to be more efficient.

D. Include the Content-Type header in the API response.

E. Leverage middleware caching and respond with HTTP code 204 in the API response.

ANSWER: A B

Explanation:

Including the ETag header in the API response and including the Last-Modified header in the API response enable HTTP cache validation for resources returned by the Python API. An ETag is an opaque version identifier for a representation. After receiving it, a client or intermediary cache can send it back in an `If-None-Match` request header. Likewise, a Last-Modified value lets a client make a conditional request using `If-Modified-Since`. When the resource has not changed, the server can return `304 Not Modified` without retransmitting the full response body. This reduces response payload transfer, serialization work, and repeated retrieval or generation of identical representations, helping the API handle increased request volume more efficiently. These validators are particularly useful for frequently polled endpoints or resources requested repeatedly by multiple consumers, because clients can retain a valid cached representation while still checking whether it is current. HTTP defines ETags and conditional evaluation semantics in RFC 9110 and specifies conditional request behavior, including `304 Not Modified`, in RFC 9110 conditional requests.

QUESTION NO: 63

An organization manages a large cloud-deployed application that employs a microservices architecture across multiple data centers. Reports have received about application slowness. The container orchestration logs show that faults have been raised in a variety of containers that caused them to fail and then spin up brand new instances.

Which two actions can improve the design of the application to identify the faults? (Choose two.)

A. Automatically pull out the container that fails the most over a time period.

B. Implement a tagging methodology that follows the application execution from service to service.

C. Add logging on exception and provide immediate notification.

D. Do a write to the datastore every time there is an application failure.

E. Implement an SNMP logging system with alerts in case a network link is slow.

ANSWER: B C

Explanation:

Implement a tagging methodology that follows the application execution from service to service. provides the correlation required to troubleshoot a distributed microservices transaction. A request or trace identifier propagated between services lets operations teams reconstruct its complete path, associate latency and errors with a particular service invocation, and determine where the failure originated even when containers are short-lived and are recreated automatically. This is the fundamental purpose of distributed tracing in cloud-native applications. Tags also add useful searchable context, such as the service name, operation, deployment version, region, tenant, and correlation ID.

Add logging on exception and provide immediate notification. ensures that actionable diagnostic data is recorded at the point a fault occurs and that the responsible team can investigate while relevant runtime context is available. Structured exception logs should include the trace or correlation ID, timestamp, affected service and container, error details, and relevant request context. Alerts based on significant exception conditions or failure-rate thresholds enable prompt response to recurring container failures and user-visible slowness. Together, correlated tracing and exception logging connect symptoms across service boundaries to detailed failure evidence. Cisco describes observability as correlating metrics, events, logs, and traces for operational insight; see Cisco Observability. The OpenTelemetry tracing specification also describes propagation of context across service boundaries: OpenTelemetry Traces.

QUESTION NO: 64

What describes microservices?

A. self-contained artifact that includes the interfaces of all application layers

B. loosely coupled services that are independently deployable and maintainable

C. tightly coupled services that are built as a single unit that is deployable to the infrastructure

D. set of services that are provided through a communication call over the internet

ANSWER: B

Explanation:

Loosely coupled services that are independently deployable and maintainable accurately describes a microservices architecture. In this approach, an application is organized as a suite of small services, each focused on a specific business capability or function. A service owns its implementation and can generally be developed, tested, released, scaled, and updated independently of the other services. Services interact through well-defined APIs or lightweight communication mechanisms, allowing teams to evolve individual components without requiring a coordinated redeployment of the entire application. This separation supports faster delivery, clearer ownership, and targeted scaling when one capability has different performance requirements from another. The services are not simply separate network calls; their defining characteristics include autonomy, independently deployable units, and loose coupling through explicit contracts. Cisco's

application-development material discusses microservices as independently deployable components that communicate through APIs, while the AWS architecture guidance likewise identifies independent deployment and loose coupling as central microservices properties. See Cisco DevNet: Application Development and AWS: Microservices on AWS.