GOOGLE Apigee API Engineer Exam Questions

Total Questions: 110+
Demo Questions: 25
Version: Updated for 2025

Prepared and Verified by Cert Empire – Your Trusted IT Certification Partner

For Access to the full set of Updated Questions – Visit:
Google Apigee API Engineer Exam Questions by Cert Empire

Question: 1

Which are techniques for dynamically choosing the target endpoint at runtime? Select all that are correct

A. use routing tables defined for the proxy using the MGMT API

B. use javascript to set targeturl

C. use route rules with conditions

D. nothing is needed, done by default

Answer:

B, C

Explanation:

Apigee provides two primary techniques for dynamically selecting a target endpoint at runtime. The first is a declarative approach using conditional RouteRules within the ProxyEndpoint. Apigee evaluates these rules sequentially, and the first rule whose condition evaluates to true directs the flow to the specified TargetEndpoint. The second is a programmatic approach, typically using a JavaScript policy. This policy can execute code to determine the appropriate target URL and then dynamically set the target.url flow variable, which overrides the statically configured URL in the TargetEndpoint definition.

Why Incorrect Options are Wrong:

A: "Routing tables" is not a standard Apigee configuration object. While you use the Management API to configure RouteRules, the API itself is a configuration tool, not the runtime mechanism that makes the dynamic choice.
D: By default, an API proxy has a static route to a single, pre-configured target endpoint. Dynamic routing requires explicit configuration and is not a default behavior.

References:

1. Google Cloud Apigee Documentation, "Conditional routing": This document explicitly describes the use of RouteRule elements with conditions to achieve dynamic routing. It states, "You can add routing logic to your API proxy flow by adding conditional routes to your TargetEndpoint configuration... A RouteRule forwards a request to a particular TargetEndpoint if the condition in the route rule evaluates to true." (See the section on "Conditional routes").
2. Google Cloud Apigee Documentation, "Flow variables reference": The documentation for the target.url variable states, "In a JavaScript policy, you can read and set this variable." This confirms that scripts can programmatically alter the target URL at runtime.
3. Google Cloud Apigee Documentation, "Route requests to the proper backend server": This guide provides a clear example of dynamic routing. Under the section "Dynamically routing to a

target with a script," it demonstrates using a JavaScript policy to set the target.url variable. It states, "You can dynamically set a target URL with code... the script overrides the static URL... configured in the TargetEndpoint XML block."

Question: 2

Which proxy endpoint configuration determines the target endpoint that will be used?

A. Step

B. TargetConfiguration

C. RouteRule

D. Connection

E. Path

Answer:

C

Explanation:

The RouteRule is the specific configuration element within an API proxy's ProxyEndpoint that determines which TargetEndpoint will process the request. RouteRules are evaluated after all policies in the ProxyEndpoint's PostFlow. They can contain conditional logic, allowing a single ProxyEndpoint to dynamically route requests to different backend services (defined by different TargetEndpoints) based on headers, query parameters, or other variables in the request. If no conditional RouteRule evaluates to true, a defaCuelrtERmopiureteRule (one without a condition) is executed.

Why Incorrect Options are Wrong:

A. Step: A Step is a container for a policy within a flow; it executes a specific action but does not handle routing between endpoints.
B. TargetConfiguration: This is not a valid XML element within the Apigee ProxyEndpoint configuration for defining routing logic.
D. Connection: The Connection element (or HTTPTargetConnection) is part of a TargetEndpoint's configuration and defines the backend service URL, not the routing rule itself.
E. Path: A Path is typically used in a flow's condition to match an incoming request URI, but the RouteRule makes the final routing decision to a TargetEndpoint.

References:

1. Google Cloud Apigee Documentation, "API proxy configuration reference": Under the "ProxyEndpoint configuration" section, the RouteRule element is defined. The documentation states, "A RouteRule forwards a request to the TargetEndpoint of the API proxy... By creating more than one RouteRule, you can route requests to different TargetEndpoints based on... differences in the request message." This directly confirms that RouteRule determines the target.
2. Google Cloud Apigee Documentation, "Endpoint properties reference": This reference details

the structure of ProxyEndpoint and TargetEndpoint. It shows that RouteRule is a child element of ProxyEndpoint and is used for "conditional routing to targets." Conversely, it shows that connection properties like URL are configured within the HTTPTargetConnection element of a TargetEndpoint, not the ProxyEndpoint.

Question: 3

What capabilities are provided when using the apigee-access node js module? Select all that apply

A. Object caching

B. Analytics Storage

C. XML and JSON conversions

D. Edge flow variable access

Answer:

A, D

Explanation:

The apigee-access Node.js module lets a Node.js script running in Apigee access Edge-specific runtime services. It exposes getCache() for storing and retrieving objects in an Edge cache (A) and getVariable()/setVariable() for reading or writing Edge flow variables (D). The module has no API for writing Analytics data nor for performing XML/JSON transformations.

Why Incorrect Options are Wrong:

B. Analytics Storage - Module offers no function to record or query Analytics metrics; these are collected automatically by Edge.
C. XML and JSON conversions - Parsing/serialization is done with standard Node.js libraries; apigee-access provides no conversion utilities.

References:

1. Google Cloud, "apigee-access Node.js module," Section "getCache(name) - Caching API" (docs.apigee.com/api-platform/reference/apigee-access-nodejs-module)
2. Google Cloud, same doc, Section "getVariable(req, name) / setVariable(req, name) - Accessing flow variables"
3. Google Cloud, "Use Node.js in Apigee Edge," Sub-section "Working with caches and variables from Node.js" (https://docs.apigee.com/api-platform/samples/using-nodejs)

Question: 4

Which describe the ResponseCache' policy in Apigee Edge? Select all that are correct

A. Helps to reduce the response latency

B. Caches both the request and the response

C. Helps to reduce the amount of load to the target system.

D. Should be attached to both Request and Response Flow

Answer:

A, C

Explanation:

The ResponseCache policy stores backend responses in Apigee Edge's in-memory cache. When a cached entry is returned, the proxy avoids a network call to the target, which shortens the time required to serve the client and decreases the traffic hitting the backend. The policy only stores the response payload and related headers; the request itself is never cached. Although the same policy can be placed in either request or response flows (or configured for both lookup and populate in one flow), doing so is optional, not mandatory.

Why Incorrect Options are Wrong:

B. Stores only the backend response; requests are never written to the cache.
D. Policy may be attached to request, response, or both flows depending on configuration; it is not a required placement in both.

References:

1. Google Apigee Edge Documentation, "Response Cache policy - About the Response Cache policy", para. 1-3 (https://docs.apigee.com/api-platform/reference/policies/response-cache-policy).
2. Google Apigee Edge Documentation, "Attach the policy", Example section: "Lookup in request flow, populate in response flow" (same URL, heading: "Example policy use cases").
3. Google Cloud Training Course, "Developing APIs with Google Cloud's Apigee API Platform", Module 4 "Caching", slide notes on ResponseCache (Coursera, course code QCIXAPIEE-M4).

Question: 5

When a quota is configured with the distributed flag set to false, the number of which of the following would affect the overall allowed traffic?

A. Cassandra nodes

B. Routers

C. Message Processors

D. Management Servers

Answer:

C

Explanation:

When the Quota policy's element is set to false, each Message Processor maintains its own separate, local counter for the quota. The quota is not shared or synchronized across the Message Processors. Consequently, the total number of allowed requests across the entire environment becomes the configured quota limit multiplied by the number of active Message Processors. For example, a quota of 100 requests/minute with three Message Processors would result in a total effective quota of 300 requests/minute, as each processor enforces the 100-request limit independently.

Why Incorrect Options are Wrong:

A. Cassandra nodes are used to maintain a central, shared counter only when the distributed flag is set to true.
B. Routers are responsible for ingress and routing traffic to Message Processors; they do not execute policies or maintain quota counts.
D. Management Servers are part of the management plane for API proxy deployment and configuration, not the runtime data plane where quotas are enforced.

References:

1. Google Cloud Apigee Documentation, Quota policy: Under the description for the element, the documentation states: "If false, the quota is not shared among all Message Processors. Each Message Processor maintains its own 'local' counter." This directly implies that the number of Message Processors determines the overall traffic capacity. (Reference Section: element).

Question: 6

You are working on a project that adheres strictly to the Roy Fielding REST concepts. You need to update a single property named "status" of a complicated entity What should you do?

A. Fetch the full entity, update the status value locally. DELETE the original entity and POST the new
version.

B. Fetch the full entity. Change only the status value and then send the whole object in the request
body of the PUT service

C. Create a new service that uses the UPDATE verb that accepts the "status* property and updates
the entity UPDATE /api/trucks/42/status HTTP/1.1 status: 5

D. Create a new service that uses the PATCH verb designed to update only given fields. PATCH /api/trucks/42 HTTP/1.1 status: 5

Answer:

D

Explanation:

The PATCH method is the standard and most appropriate HTTP verb for applying partial modifications to a resource. According to REST principles, which emphasize a uniform interface using standard methods, PATCH is designed specifically to update one or more properties of an entity without requiring the client to send the entire resource representation. This is more efficient and semantically correct for updating a single field of a complex entity, as described in the scenario.

Why Incorrect Options are Wrong:

A. Using DELETE and then POST is semantically incorrect. This sequence represents the destruction of one resource and the creation of a new one, which is not an update operation.
B. PUT is used to replace the entire resource with the new representation provided in the request body. While it can achieve the update, it is inefficient for changing a single property on a complex entity.
C. UPDATE is not a standard HTTP verb defined in the relevant RFCs. A core principle of REST is to adhere to the standard, uniform interface provided by HTTP, which does not include this verb.

References:

1. IETF RFC 5789, "PATCH Method for HTTP": This document formally introduces the PATCH method. Section 2 states, "The PATCH method requests that a set of changes described in the request entity be applied to the resource identified by the Request-URI..... The PUT method only allows a complete replacement of a document." This directly supports using PATCH for partial updates.
Source: Fielding, R., & Dusseault, L. (2010). PATCH Method for HTTP. RFC 5789. Internet Engineering Task Force. https://doi.org/10.17487/RFC5789
2. Google Cloud API Design Guide: Google's own best practices recommend PATCH for partial updates. Under the "Standard Methods" section for update, it states: "For the patch method, the client only needs to specify the fields to be updated. The server should support patch for any resources that can be updated." This confirms PATCH as the preferred method for this scenario.
Source: Google Cloud. (n.d.). Standard Methods. API Design Guide. Retrieved from https://cloud.google.com/apis/design/standardmethods#update (Refer to the section on update and patch).
3. Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures: While Fielding's dissertation predates the PATCH method, it establishes the "uniform interface" constraint (Section 5.1.5). Using a non-standard verb like UPDATE (Option C) violates this core principle. The later standardization of PATCH was done in the spirit of maintaining a uniform interface for common operations.
Source: Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine. Section 5.1.5. https://www.ics.uci.edu/fielding/pubs/dissertation/restarchstyle.htm#sec515

Question: 7

As an API Engineer your team is deploying code to production tonight. The test team will spend most of the night ensuring there are no bugs in the new release. After you deploy you want to go home. What is the best way to quickly verify a complete deployment?

A. Unit tests

B. Smoke tests.

C. Integration tests.

D. Code quality analysis.

Answer:

B

Explanation:

Smoke tests are a specific type of testing designed to quickly verify the most critical functionalities of a system immediately after a new build or deployment. The primary goal is to determine if the deployment was successful and if the application is stable enough to proceed with more exhaustive testing. This aligns perfectly with the engineer's need to perform a rapid, high-level check to confirm the deployment's health before leaving, while the dedicated test team conducts more thorough validation like integration and regression testing.

Why Incorrect Options are Wrong:

A. Unit tests are executed before deployment during the development or CI phase to validate individual code components in isolation, not the health of a complete, deployed system.
C. Integration tests are more comprehensive and time-consuming than smoke tests. They verify interactions between different system components and are typically part of the main testing cycle, not a quick post-deployment check.
D. Code quality analysis is a static, pre-deployment process that examines source code for issues. It does not test the functionality of the running application in the production environment.

References:

1. Google Cloud Documentation, "DevOps tech: Continuous testing": This document outlines different testing types in a CI/CD context. It defines smoke tests as "quick tests run as part of the build process to verify that the most critical functions of the application work." This supports their use for a rapid post-deployment verification. (See section: "Types of automated tests").
2. Google Cloud Documentation, Apigee, "Deploy and undeploy an API proxy": The final step in the deployment tutorials for Apigee proxies is to invoke the deployed proxy to verify it is working correctly. This action of performing a basic check on a live deployment is a practical example of a

smoke test. (See section: "Invoke the API proxy").

3. Pezze, M., & Young, M. (2008). Software testing and analysis: process, principles, and techniques. John Wiley & Sons. In academic literature on software engineering, smoke testing is defined as a non-exhaustive set of tests aimed at ensuring the most crucial functions of a program work, but not bothering with finer details. This is used to decide if a build is stable enough for further testing. (See Chapter 1, Section 1.2.2, "Testing Activities").

Question: 8

When populating the Quota configuration for an API product, which statement is true?

A. The Quota specified will automatically be enforced for any Developer App Keys assigned to the API
product.

B. After validating an API key or access token, flow variables are automatically populated with the Quota configuration for later use in a Quota policy

C. Rate limiting will be enforced precise to the seconds level, even if you configure a per-minute or
higher interval

D. The Quota configuration specified on the API product enforces a global rate limit across all API proxies

Answer:

B

Explanation:

When an API proxy uses a policy like VerifyAPIKey or OAuthV2/VerifyAccessToken to validate credentials, Apigee identifies the associated developer application and its subscribed API products. Upon successful validation, Apigee populates a series of flow variables with the configuration settings from the API product, including the defined quota limit, interval, and time unit. A subsequent Quota policy in the API flow can then be configured to read these dynamically populated variables to enforce the specific limits associated with that API product, rather than using hardcoded values within the policy itself. This is the standard mechanism for enforcing product-level quotas.

Why Incorrect Options are Wrong:

A. Defining a quota on an API product is a configuration step; it does not automatically enforce the limit. An explicit Quota policy must be placed in the API proxy flow to read the configuration and perform the enforcement.
C. Quota enforcement is not guaranteed to be precise to the second. Due to the distributed nature of the Apigee gateway and asynchronous counter updates (the default behavior), there can be minor variations in enforcement, especially for short time intervals.
D. The quota is not a global limit across all API proxies. It is scoped to the specific API product and applies to the total number of requests made by a single developer app to the collection of API proxies bundled within that product.

References:

1. Google Cloud Apigee Documentation, "Verify API Key policy": Under the "Flow variables" section, the documentation lists the specific variables that are populated after the policy executes. These include verifyapikey.policyname.apiproduct.quota.limit, verifyapikey.policyname.apiproduct.quota.interval, and verifyapikey.policyname.apiproduct.quota.timeunit, which directly supports answer B.
2. Google Cloud Apigee Documentation, "Quota policy": This page explains how to configure the Quota policy. It explicitly states, "The Quota policy can get its settings from the API product configuration... If you want the Quota policy to refer to the quota settings on the API product, you must configure the Quota policy accordingly." This confirms that a policy is required for enforcement (making A incorrect) and that it uses the product settings (supporting B).
3. Google Cloud Apigee Documentation, "Quota policy", element: The documentation notes, "Quota enforcement can vary slightly and is not always exact for a number of reasons," and explains that the default counter update is asynchronous. This directly contradicts the claim of second-level precision in option C.
4. Google Cloud Apigee Documentation, "What is an API product?": This page defines the scope of an API product. It states, "The quota is the total number of requests allowed for all APIs in the product for a specific app over a given time period." This clarifies that the quota is scoped to the product, not globally, making option D incorrect.

Question: 9

Which policies can be used to create or modify a request message for a service callout? Select all that are correct

A. ServiceCallout

B. AssignMessage

C. RequestMessage

D. Message Validation

Answer:

A, B

Explanation:

The AssignMessage policy is the primary tool for creating or modifying messages within an API proxy flow. It can be used to construct a completely new request message object, populating its headers, path, and payload, which is then referenced by the ServiceCallout policy.
Additionally, the ServiceCallout policy itself has a element. This element allows for the direct definition and modification of the outbound request message that will be sent to the external service, making it another valid method for creating the callout's request.

Why Incorrect Options are Wrong:

C. RequestMessage: RequestMessage is not a standalone Apigee policy. It is an element name used within other policies, such as the ServiceCallout policy, to define the request.
D. Message Validation: The MessageValidation policy is used to check if a message conforms to a specified JSON or XML schema; it does not create or modify the message content.

References:

1. Google Cloud Apigee Documentation, "Assign Message policy": "The Assign Message policy changes or creates new request and response messages during an API proxy flow. The policy lets you perform the following actions on those messages: ... Create a new request or response message." This directly supports the use of AssignMessage for creating a request.
2. Google Cloud Apigee Documentation, "Service Callout policy": Under the section for the element, the documentation states: "Use the element to build the request message that you send to the backend service." It further details how to use sub-elements like , , and to modify headers, parameters, and the payload, confirming its role in creating/modifying the request.
3. Google Cloud Apigee Documentation, "Policy reference overview": This official list of all Apigee policies does not include a policy named RequestMessage, confirming it is not a valid policy type.
4. Google Cloud Apigee Documentation, "Message Validation policy": "The Message Validation policy enables you to validate a request or response message against predefined schemas... It

does not change the message itself." This confirms its purpose is validation, not modification.

Question: 10

Which policy can be used to restrict access to API resources based on the client IP?

A. Regular Expression Protection policy

B. Basic Authentication policy

C. Access Control policy

D. Raise Fault policy

Answer:

C

Explanation:

The Access Control policy is the designated Apigee policy for implementing IP-based access restrictions. It allows developers to create allowlists or denylists of IP addresses. The policy evaluates the IP address of the incoming request against the rules defined within its configuration. If a request's IP address matches a "deny" rule or does not match any "allow" rule, access to the API resource is blocked. This provides a direct and efficient mechanism for controlling traffic based on its origin.

Why Incorrect Options are Wrong:

A. Regular Expression Protection policy is used to protect against content-based threats by matching message content against regular expressions, not for IP-based access control.
B. Basic Authentication policy is used to enforce credential-based security by validating a username and password, not the client's IP address.
D. Raise Fault policy is used to generate a custom error response when a fault occurs; it does not perform the access control check itself.

References:

1. Google Cloud, Apigee Documentation, "AccessControl policy": "The AccessControl policy lets you allow or deny access to your APIs by specific IP addresses.....You can configure an AccessControl policy to use a combination of MatchRule elements to either allow or deny access." (See the "About the AccessControl policy" section and the XML configuration examples).
2. Google Cloud, Apigee Documentation, "RegularExpressionProtection policy": "The RegularExpressionProtection policy protects your API from content-based threats by letting you configure regular expressions to match on request payloads, query parameters, headers, and form parameters." (See the "About the RegularExpressionProtection policy" section).
3. Google Cloud, Apigee Documentation, "BasicAuthentication policy": "The BasicAuthentication policy extracts credentials from the request Authorization header, decodes them, and then uses them to authenticate the user making the API call." (See the "About the BasicAuthentication

policy" section).
4. Google Cloud, Apigee Documentation, "RaiseFault policy": "Use the RaiseFault policy to define an error response that is returned to the requesting client when a specific condition occurs." (See the "About the RaiseFault policy" section).

Question: 11

You have a requirement to expose functions and data from an existing back-end system Using Apigee recommended practices, which step should you take first*?

A. Write business and functional requirements documents.

B. Implement ad-hoc microservices using a managed container system.

C. Catalog the data model of the backing data store or API into a data dictionary

D. Work with the existing or targeted application consumers to build an Open API Specification model

Answer:

D

Explanation:

Apigee strongly recommends an "API-first" or "design-first" approach. This methodology prioritizes designing the API contract before writing any implementation code. The first step in this process is to collaborate with the intended API consumers to define their needs and formalize them into an OpenAPI Specification (OAS). This consumer-centric approach ensures the API is useful, intuitive, and serves as a clear contract for both the front-end application developers and the back-end service developers. This allows for parallel development and ensures the final product meets the business requirements from the outside-in.

Why Incorrect Options are Wrong:

A. Writing traditional requirements documents is a precursor, but creating the OAS with consumers is the more specific, actionable first step in modern API design that captures these requirements contractually.
B. This is a "code-first" approach. Implementation should only begin after the API contract (the OpenAPI Specification) has been designed and agreed upon, as per recommended practices.
C. This is a "backend-first" approach. While understanding the backend is necessary, the API design should be driven by consumer needs, not dictated by the internal data model.

References:

1. Google Cloud Documentation, Apigee, "What is API-first?": "With an API-first approach, teams begin by designing an API. This design then functions as a contract between the teams building the API and the internal or external customers who will consume it." This source establishes that design is the initial step.
2. Google Cloud Documentation, Apigee, "API development lifecycle": The documentation outlines the lifecycle phases, starting with "1. Design: The API producer creates an API specification... Apigee recommends using an OpenAPI specification to design your APIs." This

explicitly places design, using an OpenAPI specification, as the first phase.
3. Google Cloud Documentation, Apigee, "API design and security" tutorial: The tutorial's first step under "Create an API specification" is to "Create an OpenAPI specification that describes the API." This demonstrates the practical application of starting with the specification.

Question: 12

If a custom analytics report needs to filter based on data from the request payload, which policy would be used?

A. Custom Report

B. Message Logging

C. StatisticsCollector

D. AssignMessage

Answer:

C

Explanation:

The StatisticsCollector policy is specifically designed to extract and collect custom data from an API proxy flow, including variables derived from the request payload. This data is then sent to the Apigee analytics service. Once collected, this custom data can be used as a dimension or metric within custom analytics reports, enabling filtering and analysis based on the payload's contents. Other policies might extract the data into a variable, but StatisticsCollector is the policy that sends it to the analytics engine.

Why Incorrect Options are Wrong:

A. Custom Report: A Custom Report is a feature for viewing and analyzing analytics data in the Apigee UI; it is not a policy used to collect the data itself.
B. Message Logging: The Message Logging policy sends request or response data to external logging systems (e.g., Splunk, syslog) for debugging, not to the internal Apigee analytics service for reporting.
D. AssignMessage: The AssignMessage policy is used to create or modify messages and flow variables. While it can help extract data, it does not send that data to the analytics service.

References:

1. Google Cloud Apigee Documentation, "StatisticsCollector policy": "Use the StatisticsCollector policy to collect custom analytics data from your API proxy flow... The policy logs the data to the Apigee analytics service, where it can be queried by custom reports in the Apigee UI." This directly confirms the policy's purpose.
2. Google Cloud Apigee Documentation, "Custom analytics collection": This section details the process, stating, "To collect custom analytics data, you use the StatisticsCollector policy." It further explains how to use policies like JSONtoXML or ExtractVariables to parse the payload and then use StatisticsCollector to log the extracted values.
3. Google Cloud Apigee Documentation, "Creating and managing custom reports": This document

explains how to build reports using dimensions, including custom dimensions. It notes, "Custom dimensions and metrics are analytics variables that you explicitly collect from API traffic using policies like the Statistics Collector policy."

Question: 13

Which protocols are supported by the Message Logging policy? Select all that are correct

A. FTP

B. HTTP

C. SCP

D. TCP

E. UDP

Answer:

B, D, E

Explanation:

The Apigee Message Logging policy is designed to send log messages to various third-party log management services. For integration with standard syslog servers, the policy's configuration element explicitly supports both TCP and UDP as transport protocols. For integrations with other logging services like Splunk (via its HTTP Event Collector) or Loggly, the policy sends log data over HTTP or HTTPS. Therefore, HTTP, TCP, and UDP are all supported protocols depending on the configured logging endpoint.

Why Incorrect Options are Wrong:

A. FTP: This is a file transfer protocol and is not supported by the Message Logging policy for streaming log data.
C. SCP: This is a secure file copy protocol, not a real-time logging protocol supported by this policy.

References:

1. Google Cloud Apigee Documentation, "Message Logging policy": Under the section Syslog element, the child element is described. The documentation states: "The protocol to use. Valid values include TCP and UDP. The default is UDP." This directly confirms support for TCP and UDP.
Source: Google Cloud Documentation, Apigee, Reference, Policies, Message Logging policy.
2. Google Cloud Apigee Documentation, "Message Logging policy": Under the section Splunk element, the policy is configured to send messages to the Splunk HTTP Event Collector (HEC). The documentation notes: "The Splunk HEC lets you send data and application events to a Splunk deployment over HTTP and Secure HTTP (HTTPS)." This confirms support for HTTP.
Source: Google Cloud Documentation, Apigee, Reference, Policies, Message Logging policy.
3. Google Cloud Apigee Documentation, "Message Logging policy": The section on the Loggly

element describes sending logs to the Loggly service, which uses an HTTP/S-based API for log ingestion. This further confirms HTTP as a supported protocol.
Source: Google Cloud Documentation, Apigee, Reference, Policies, Message Logging policy.

Question: 14

Which OAuth 2 0 grant requires redirection'?

A. Authorization Code

B. Resource Owner Password Credentials

C. Refresh Token

D. Client Credentials

Answer:

A

Explanation:

The Authorization Code grant type is designed for applications that can securely store a client secret, such as web applications running on a server. This flow involves redirecting the user's browser (user-agent) to the authorization server to authenticate and grant consent. After the user approves the request, the authorization server redirects the user back to the client application's pre-registered redirect URI with an authorization code. The client then exchanges this code for an access token in a secure, back-channel communication. This redirection is a fundamental part of the flow.

Why Incorrect Options are Wrong:

B. Resource Owner Password Credentials: This grant type involves the client directly collecting the user's username and password and sending them to the authorization server. No redirection is used.
C. Refresh Token: A refresh token is used to obtain a new access token via a direct, back-channel API call to the token endpoint. It does not involve user interaction or redirection.
D. Client Credentials: This grant is for machine-to-machine communication where the client authenticates itself directly with the authorization server. It does not involve a user or redirection.

References:

1. Google Cloud, Apigee Documentation, "Implementing the authorization code grant type": "In the authorization code grant type flow, the user is redirected to the client app after they authenticate with the resource server. The key to this flow is that the user is redirected from the authorization server to a 'redirect URI'."
2. Internet Engineering Task Force (IETF), RFC 6749, "The OAuth 2.0 Authorization Framework", Section 4.1: This section details the Authorization Code grant. Section 4.1.1, "Authorization Request," explicitly states, "The client directs the resource owner to the authorization endpoint." Section 4.1.2, "Authorization Response," describes the redirection back: "...the authorization server directs the user-agent back to the client using the redirection URI..."

3. Google Cloud, Apigee Documentation, "OAuth 2.0 grant types": This overview page contrasts the different flows. It describes the Authorization Code flow as involving a "browser redirect" while describing the Client Credentials and Resource Owner Password Credentials flows as direct, one-step or two-step exchanges without user-agent redirection.

Question: 15

The product owner has asked you for a new API. This new API will change a configuration for a backend system. The use case calls for a single API. Which verb should you use?

A. GET

B. PUT

C. POST

D. HEAD

Answer:

B

Explanation:

The PUT method is the most appropriate choice for this scenario. According to RESTful API design principles, PUT is used to replace the entire state of a target resource with the request payload. The action of "changing a configuration" implies setting a known resource (/configuration) to a new, complete state. This operation is idempotent, meaning that making the same PUT request multiple times will have the same effect as making it once, which is a desirable characteristic for configuration updates.

Why Incorrect Options are Wrong:

A. GET: This verb is used exclusively for retrieving data from a specified resource; it is a safe method that should not have any side effects like changing state.
C. POST: This verb is typically used to create a new subordinate resource or for actions that are not idempotent. While it can be used for updates, PUT is more semantically correct for a full replacement.
D. HEAD: This verb is identical to GET but requests only the headers and not the response body. It is used for checking resource metadata, not for modification.

References:

1. Google Cloud API Design Guide, Standard Methods: The guide specifies the use of PUT: "The Update method should take the resource message... The Update method should use the PUT HTTP verb." This aligns with the scenario of updating a configuration resource.
Source: Google Cloud. (n.d.). Standard Methods. API Design Guide. Retrieved from https://cloud.google.com/apis/design/standardmethods#update. (Refer to the "Update" section).
2. IETF RFC 7231, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, Section 4.3.4: This is the formal specification for the PUT method. It states, "The PUT method requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message payload."

Source: Fielding, R., & Reschke, J. (2014). RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Internet Engineering Task Force. Section 4.3.4. Retrieved from https://datatracker.ietf.org/doc/html/rfc7231#section-4.3.4.

3. Apigee Documentation, RESTful API design cookbook: The Apigee documentation outlines best practices, stating that PUT is used to update a resource. "For example, when a customer wants to change her name, a PUT request is made to the customer resource." This is analogous to changing a system configuration.

Source: Google Apigee. (n.d.). RESTful API design cookbook. Apigee Documentation. Retrieved from https://cloud.google.com/apigee/docs/api-platform/fundamentals/restful-api-design-cookbook#put. (Refer to the "PUT" section).

Question: 16

The product team is rolling out a new reseller program with API's. The product owner has created Epics covering the high level requirements. The product owner delegate has asked for help creating a product backlog. What task would the product owner delegate need assistance with?

A. Creation of a message logging policy in Apigee.

B. Creating user stories to fulfill the business requirements.

C. Creating support tickets that cover each of the business requirements.

D. Creating a cross functional team of API engineers, business analysts, and backend software developers.

Answer:

B

Explanation:

A product backlog is a prioritized list of work for the development team that is derived from the product roadmap and its requirements. The process of creating a backlog from high-level requirements (Epics) involves breaking them down into smaller, more granular, and actionable items. In Agile methodologies, these items are typically user stories. Each user story captures a specific requirement from a user's perspective. Therefore, the primary task the product owner's delegate needs assistance with is translating the broad Epics into specific user stories that will populate the product backlog and guide the development team's work.

Why Incorrect Options are Wrong:

A. Creation of a message logging policy in Apigee.
This is a specific, low-level implementation task that would result from a user story, not a task to create the backlog itself.
C. Creating support tickets that cover each of the business requirements.
Support tickets are used for tracking bugs or operational issues, not for defining new development work in a product backlog.
D. Creating a cross functional team of API engineers, business analysts, and backend software developers.
Team formation is an organizational activity that precedes development work; it is not a task related to defining the content of the product backlog.

References:

1. Google Cloud Documentation, API design guide, Overview: The guide emphasizes the importance of the design phase before coding, stating, "A good design meets the needs of the people who will use the API." This design and requirements-gathering phase is where high-level business needs (Epics) are translated into specific functional requirements (user stories) for the backlog. (Source: Google Cloud, "API design," https://cloud.google.com/apis/design/overview, Section: "Designing an API").
2. Google Apigee, "The Digital Transformation Journey" Ebook: In the context of treating APIs as products, this resource highlights the role of the API product manager. The product manager is responsible for "gathering requirements, and for defining the user experience." This process of gathering and defining requirements is precisely what leads to the creation of user stories to build a product backlog. (Source: Google Apigee, "The Digital Transformation Journey," Chapter 3: "Building an API Program," Section: "The API Team").
3. Atlassian, Agile Coach, "User Stories, Epics, and Initiatives": While a commercial vendor, Atlassian's Agile Coach is widely cited as an authoritative source on Agile practices. It defines the hierarchy: "Epics are large bodies of work that can be broken down into a number of smaller tasks (called stories)..." This directly supports that the next step after having Epics is to create stories for the backlog. (Source: Atlassian Agile Coach, https://www.atlassian.com/agile/project-management/epics-stories-initiatives).

Question: 17

Given the following Javascript code snippet, which statement is true? var paloAlto = httpClient.get'http://weather.yahooapis.com/forecastrss?w=2467861'); context.session'paloAlto' = paloAlto;

A. The code execution will wait for the httpClient to receive a response and store that into a session
vanable named paloAlto.

B. The string paloAlto' will be stored in a message flow variable named paloAlto

C. The httpClient request will send a POST request to http //weather yahooapis com/forecastrss

D. The code execution will complete even if the httpClient has not yet received a response

Answer:

D

Explanation:

The httpClient object within the Apigee JavaScript policy is designed to be non-blocking and asynchronous. When httpClient.get() is called, it initiates the HTTP request and immediately returns an Exchange object without waiting for the remote server to respond. The JavaScript policy's execution then continues to the next line and completes. The actual handling of the response must be managed separately, typically through a callback mechanism, but the initial script execution does not block.

Why Incorrect Options are Wrong:

A. The httpClient object is non-blocking. The code execution does not wait for a response; it continues immediately after initiating the request.
B. The value assigned to the flow variable paloAlto is the Exchange object returned by httpClient.get(), not the literal string 'paloAlto'.
C. The code explicitly uses the .get() method, which sends an HTTP GET request. A POST request would require using a method like .post() or .send().

References:

1. Google Cloud, Apigee X Documentation, "JavaScript object model":
Section: httpClient object: "The httpClient object provides a mechanism for making asynchronous, non-blocking HTTP requests to any URL from within a JavaScript policy... The send and get methods return an exchange object. This object has no properties and exists only to provide a container for the waitForComplete() method." This reference directly supports that the call is non-blocking (making D correct and A incorrect) and that an Exchange object is returned (making B incorrect).

2. Google Cloud, Apigee X Documentation, "JavaScript object model":
Section: httpClient.get(url): This section details the function signature for making a GET request, confirming that the code snippet is indeed performing a GET and not a POST (making C incorrect).

Question: 18

A cloud customer wants to safeguard their APIs against a sudden increase in traffic. You need to calculate an allowable traffic rate of 100 transactions per second (TPS) What should you do?

A. Use a default Spike Arrest policy setting the limit to 100 TPS

B. Use a Quota enforcement policy set to limit throughput to 100 TPS

C. Use a Spike Arrest policy setting the UseEffectiveCount parameter

D. Keep a count of accesses in the back-end, rejecting queries when they exceed 100 TPS

Answer:

A

Explanation:

The Spike Arrest policy is the appropriate tool in Apigee for protecting against sudden increases in traffic. Its primary function is to smooth traffic flow and prevent surges that could overwhelm backend services. By setting the element within the policy to 100ps, you are explicitly configuring the API proxy to allow a maximum of 100 transactions per second (TPS). This directly addresses the requirement to safeguard the API against traffic spikes by throttling requests at the gateway level before they reach the backend.

Why Incorrect Options are Wrong:

B. Use a Quota enforcement policy set to limit throughput to 100 TPS
The Quota policy is designed to enforce consumption limits over longer time intervals (e.g., minute, hour, day), not to manage second-by-second traffic spikes.
C. Use a Spike Arrest policy setting the UseEffectiveCount parameter
UseEffectiveCount is a specific configuration for distributing the count across multiple message processors. The fundamental action is setting the rate, making option A the more direct answer.
D. Keep a count of accesses in the back-end, rejecting queries when they exceed 100 TPS
This approach is an anti-pattern. A key benefit of Apigee is to offload non-functional requirements like rate-limiting from the backend service to the API gateway.

References:

1. Google Cloud Apigee Documentation, "Spike Arrest policy": This document explicitly states, "The Spike Arrest policy protects against traffic surges with a element. This element throttles the number of requests processed by an API proxy and sent to a backend... For example, you can limit the number of requests to 100 per second." This directly supports option A.
Source: Google Cloud, Apigee Documentation, Policies, spike-arrest-policy.html.
2. Google Cloud Apigee Documentation, "Comparing Quota and Spike Arrest": This page clarifies

the distinction: "Spike Arrest prevents traffic spikes. Quota enforces consumption limits on client apps." It further explains that Spike Arrest is for protecting against sudden bursts, while Quota is for business-level limits over longer periods. This confirms why option B is incorrect for this scenario.
Source: Google Cloud, Apigee Documentation, comparing-quota-and-spike-arrest.html.
3. Google Cloud Apigee Documentation, "Spike Arrest policy Reference": The documentation for the element specifies its use case: "Distributes the Spike Arrest count among message processors when using auto-scaling groups." This shows it is a specialized setting, not the primary method for setting the rate, making option C less appropriate than A.
Source: Google Cloud, Apigee Documentation, Reference, Policy reference, spike-arrest-policy.html#element-reference.

Question: 19

An API product in Apigee can be used to

A. restrict access to a set of APIs

B. configure the quota limits for APIs

C. restrict access to APIs in different environments

D. all of the above

Answer:

D

Explanation:

An API product in Apigee is a core entity for bundling and managing API resources. Its primary functions include grouping a collection of API proxies and resource paths to control access for developer apps via API keys. It is also the central mechanism for enforcing consumption limits by configuring quotas (e.g., requests per minute). Furthermore, API products are scoped to specific environments (like test or prod), which allows administrators to control which sets of APIs are available for consumption in each environment. Therefore, all the listed options represent valid uses of an API product.

Why Incorrect Options are Wrong:

A. This is correct but incomplete. Restricting access is a primary function, but not the only one.
B. This is correct but incomplete. Configuring quotas is a key feature, but API products have broader capabilities.
C. This is correct but incomplete. Scoping access to environments is a function, but it's part of a larger set of features.

References:

1. Google Cloud Documentation - What is an API product?: "An API product is a collection of API resources (URIs) combined with a service plan... API products are the central mechanism for access control and quota management for your APIs." This statement directly supports options A and B.
Source: Google Cloud, Apigee Documentation, "What is an API product?", Section: "What is an API product?".
2. Google Cloud Documentation - Managing API products: The configuration page for an API product explicitly shows sections for adding API proxies/resources, setting quotas, and selecting the environments where the product will be available. This confirms that all three functions (A, B, and C) are integral parts of an API product's configuration.
Source: Google Cloud, Apigee Documentation, "Managing API products", Section: "Exploring the

API products page".
3. Google Cloud Documentation - Quota policy: "The number of requests allowed by a quota can be configured in an API product... The quota settings in the API product take precedence over the settings in the Quota policy. If a quota is not defined in the product, the quota settings in the policy are enforced." This highlights the primary role of the API product in quota management (Option B).
Source: Google Cloud, Apigee Documentation, "Quota policy", Section: "About the Quota policy".

Question: 20

You are composing logs created from user input that contains confidential user information You need to record the presence of all fields, while ensuring that confidential information is not recorded in any log Which approach should you take?

A. Use a data masking configuration to hide sensitive data from logs

B. Use conditions to skip logging for data that might contain confidential information.

C. Work with the administrators of the log aggregators to mask confidential data contents.

D. Sanitize all data on entry using regular expressions that search for confidential data formats.

Answer:

A

Explanation:

The most appropriate and secure approach within Apigee is to use a data masking configuration. This feature is specifically designed to hide sensitive data in trace and debug sessions, which are sources for logs. By defining a mask for specific JSON paths, XPath expressions, or variables containing confidential user information, Apigee automatically replaces the actual values with asterisks () in the log/trace output. This meets the requirement of recording the presence of the fields (the keys are still visible) while ensuring the confidential values are never recorded.

Why Incorrect Options are Wrong:

B: Skipping logging based on conditions would fail the requirement to "record the presence of all fields," as the entire log entry for that transaction might be omitted.
C: Relying on downstream log aggregators to perform masking is a security anti-pattern. It exposes sensitive data in transit and within Apigee before it is masked.
D: Sanitizing data "on entry" is ambiguous and could imply altering the actual request/response message, which would corrupt the data sent to the backend target, rather than just masking it for logging.

References:

1. Google Cloud Apigee Documentation, "Mask sensitive data": This document explicitly states, "Apigee lets you define mask configurations to hide specific data in trace and debug sessions... When data is masked, it is replaced with asterisks in the trace and debug output." This directly supports using a data masking configuration as the correct approach.
Source: Google Cloud Apigee Documentation Reference API proxies Mask sensitive data.
2. Google Cloud Apigee Documentation, "Create a mask configuration with the API": This guide provides the technical implementation details for data masking, showing how to define masks for specific variables, JSON payloads, and XML payloads. It confirms that this is a built-in,

configurable feature for the exact purpose described in the question.
Source: Google Cloud Apigee Documentation Reference API proxies Create a mask configuration with the API.

Question: 21

In Apigee Quota policy, if it is essential that you do NOT allow any API calls over the quota, which configuration option is used?

A. distributed, non-synchronous

B. non-distributed, synchronous

C. distnbuted, synchronous

D. non-distnbuted, non-synchronous

Answer:

C

Explanation:

To ensure that no API calls are allowed over the quota limit, the Quota policy must be configured for strict, real-time enforcement across all message processors. This is achieved by setting to true to maintain a single, central counter shared by all processors. Additionally, must be set to true to ensure that this central counter is updated and checked before the API request is allowed to proceed. This synchronous update prevents race conditions where multiple requests could slip through before the counter is decremented, thus guaranteeing strict adherence to the defined limit.

Why Incorrect Options are Wrong:

A. distributed, non-synchronous: A non-synchronous (asynchronous) update allows for quota overruns because the counter is not updated in real-time, potentially letting multiple calls exceed the limit before the central count reflects it.
B. non-distributed, synchronous: A non-distributed configuration means each message processor has its own separate counter, leading to an inaccurate total count and making it impossible to enforce a global quota strictly.
D. non-distributed, non-synchronous: This is the least strict configuration. The counting is inaccurate across processors (non-distributed), and the updates are not real-time (non-synchronous), making significant quota overruns likely.

References:

1. Google Cloud, Apigee Documentation, "Quota policy": Under the section for the element, the documentation for its child elements and explicitly states: "If it is essential that you do not allow any API calls over the quota, specify true and true." This directly supports the choice of distributed and synchronous for strict enforcement.
Source: https://cloud.google.com/apigee/docs/api-platform/reference/policies/quota-policy, Section: "Element: ".

2. Google Cloud, Apigee Documentation, "Quota policy": The documentation further clarifies the behavior of asynchronous updates: "With an asynchronous configuration, there is a possibility that some API calls will exceed the quota...". This confirms why options A and D, which use non-synchronous settings, are incorrect for strict enforcement.
Source: https://cloud.google.com/apigee/docs/api-platform/reference/policies/quota-policy, Section: "Element: ".

Question: 22

You are working on a new design for an API. The backend API will set the customer to a deleted status. The customer will remain in the backend database for later cleanup. The customer can no longer be retrieved by the API once the status is set. Which method should be used at the Apigee proxy to set the deleted status?

A. GET

B. PUT

C. POST

D. DELETE

E. OPTIONS

Answer:

D

Explanation:

The DELETE method is the semantically correct choice for this operation. According to RESTful API design principles, the DELETE method is used to request the removal of a target resource. The fact that the backend performs a "soft delete" by changing a status field is an implementation detail. From the API consumer's perspective, the resource is being removed because it can no longer be retrieved, which is the exact intent of the DELETE verb. This approach maintains a clean, predictable, and standard-compliant API interface.

Why Incorrect Options are Wrong:

A. GET: This method must be "safe," meaning it is used only for data retrieval and should never alter the state of a resource on the server.
B. PUT: This method is used to replace the entire state of an existing resource. While it could change a status, it is not the correct verb for expressing the intent of deletion.
C. POST: This method is typically used to create a new resource or as a catch-all for operations that don't fit other verbs. DELETE is far more specific and appropriate for this action.
E. OPTIONS: This method is used by a client to determine the communication options and requirements associated with a resource, not to modify its state.

References:

1. IETF RFC 7231, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, Section 4.3.5. DELETE: "The DELETE method requests that the origin server remove the association between the target resource and its current functionality... A successful response SHOULD be 200 (OK) if the response includes an entity-body describing the status, 202 (Accepted) if the

action has not yet been enacted, or 204 (No Content) if the action has been enacted..." This specification defines DELETE as removing the resource's accessibility, which aligns with the scenario.
2. Google Cloud, API Design Guide, Standard Methods: "The delete method takes a resource name... and deletes or schedules the deletion of the specified resource... A standard delete method MUST use the DELETE HTTP verb." This official Google guide explicitly maps the deletion action to the DELETE HTTP method.
3. Apigee Documentation, Web API design cookbook, Section: Use HTTP methods to operate on your collections and elements: "To delete an element from a collection, use DELETE with the URI of the element. For example: DELETE /customers/1234". This demonstrates the standard practice recommended within the Apigee ecosystem.

Question: 23

You have a single back end that needs to be exposed to customers using different API request and response payloads You need to allow these different request types without breaking existing implementations What should you do?

A. Create a new API proxy for new customers and invoke backend target system with required parameters.

B. Configure the API as a pass-through proxy and invoke backend target system with client request
parameters.

C. Create a new proxy xml and base path with upgraded version and invoke backend target system
with required parameters

D. Include a new customer requirement in an existing API proxy and invoke backend target system
with required parameters.

Answer:

C

Explanation:

The most effective and standard practice for introducing non-backward-compatible changes, such as different request and response payloads, is to implement API versioning. Creating a new version of the API proxy with a new base path (e.g., /v2) allows you to build the required payload transformation logic for new customers within this new version. This approach ensures that the existing API version (e.g., /v1) remains unchanged and fully operational, thereby guaranteeing that existing client implementations are not broken. This strategy facilitates the evolution of an API while maintaining stability and backward compatibility for established consumers.

Why Incorrect Options are Wrong:

A. Creating an entirely new API proxy for new customers leads to proxy proliferation, which complicates API management, governance, and discoverability compared to versioning a single logical API.
B. A pass-through proxy forwards requests and responses without modification. This approach cannot fulfill the core requirement of handling different payload structures for a single backend.
D. Modifying a live, in-use API proxy version to accommodate new requirements is a high-risk practice that can inadvertently introduce breaking changes for existing consumers.

References:

1. Google Cloud - Apigee Documentation, "API versioning": "To help you manage change, Apigee expects that you will version your APIs... When you need to make a major update (a 'breaking change') to your API, such as changing the format of the response, you publish a new version of your API... By versioning your APIs, you can help ensure the reliability of your APIs for the developers who use them." This source directly supports creating a new version for breaking changes like different payloads.
2. Google Cloud - Apigee Documentation, "Develop API proxies": This section details how API proxies are structured with base paths and conditional flows. Creating a new version typically involves a new proxy bundle or significant changes deployed under a new base path (e.g., /v2), which aligns with the description in option C.
3. Google Cloud - Apigee Documentation, "What is a pass-through proxy?": "A pass-through proxy... does not include any policies." This confirms that a pass-through proxy (Option B) cannot perform the necessary payload transformations.

Question: 24

You have a new set of requirements for a mobile app. The product team has asked for the following. • The app requires access to customer order information • The app needs to allow a search function for orders by product name Choose two development tasks that would accomplish the requirements. Choose 2 answers

A. Create a new API proxy for a GET /v1/customers/customerid/orders

B. Create a new API proxy for a GET /v1/customers/customerid/products/productname

C. The design should include a new custom header X-Product-Name

D. The Apigee proxy should allow a query parameter for q=

E. The Apigee proxy should allow a query parameter for orderId=

Answer:

A, D

Explanation:

To meet the requirements, two primary development tasks are necessary, following RESTful API design principles. First, an API proxy must be created to expose the collection of orders for a specific customer. The resource path GET /v1/customers/customerid/orders (Option A) correctly represents this hierarchical relationship. Second, to enable searching within this collection by product name, a filtering mechanism is needed. The standard RESTful approach is to use a query parameter, such as q= (Option D), to pass the search term. The resulting request would be GET /v1/customers/customerid/orders?q=.

Why Incorrect Options are Wrong:

B. The path .../products/productname is incorrect as it implies retrieving a specific product resource, not searching for orders that contain a product.
C. Using a custom header for search criteria is a non-standard practice. Headers are intended for request/response metadata, not for filtering resource data.
E. An orderId query parameter would be used to retrieve a single, specific order by its unique identifier, which does not fulfill the requirement to search by product name.

References:

1. Google Cloud API Design Guide, Resource Names: This guide specifies the standard for naming resource collections. The pattern /collection/id is recommended. Option A, /customers/customerid/orders, correctly identifies the orders collection belonging to a specific customer.
Source: Google Cloud, API Design Guide, "Resource Names".

2. Google Cloud API Design Guide, Standard Methods, List: The guide details the List method, which is used to retrieve a collection of resources. It states that filtering should be handled via parameters in the request URL. This supports using a query parameter for the search function.
Source: Google Cloud, API Design Guide, "Standard methods - List".
3. Apigee Documentation, Extract Variables policy: This official documentation explains how to extract content from requests, including query parameters. An Apigee developer would use this policy to read the value of the q parameter (Option D) to pass it to the backend service for filtering.
Source: Google Cloud, Apigee Documentation, "Reference: Extract Variables policy".
4. Apigee Documentation, Flow variables reference: The documentation lists variables that are automatically populated by Apigee, such as request.queryparam.queryparamname, which confirms that query parameters are a natively supported and expected part of the request message for processing within a proxy.
Source: Google Cloud, Apigee Documentation, "Flow variables reference".

Question: 25

As an Apigee API Engineer you attend a meeting where a Product Owner would like to release a new feature to customers. There are several teams in the meeting, Backend API team, Apigee API team, and the Security team. The feature will be exposed through the companies external facing website. The architecture allows the website to call the backend APIs directly. The security team raises a concern about the backend APIs being wide open to anyone inside the network, not just the external website. You are later contacted and asked for your teams impacts. How should you reply?

A. You should recommend an Apigee Edge Access Control policy

B. You should recommend that the backend API's use TLS v12 to secure their APIs.

C. You should recommend the use of custom secure headers with time stamp verification

D. You should recommend a design change that uses a Apigee microgateway in front of the backend
APIs.

Answer:

D

Explanation:

The security team's concern is about access control-specifically, that any internal service can call the backend APIs, not just the intended website. The current architecture bypasses any policy enforcement point. Recommending a design change to place an Apigee microgateway in front of the backend APIs directly solves this problem. The microgateway acts as a lightweight, containerized policy enforcement point for internal ("east-west") traffic. This allows the Apigee team to apply standard security policies like API key validation, OAuth, or IP-based access control, ensuring that only authorized clients (like the website) can access the backend services.

Why Incorrect Options are Wrong:

A. You should recommend an Apigee Edge Access Control policy: This policy would be ineffective because the question states the website calls the backend APIs directly, meaning traffic does not pass through the Apigee Edge gateway where policies are enforced.
B. You should recommend that the backend API's use TLS v12 to secure their APIs: TLS encrypts data in transit, protecting against eavesdropping. However, it does not solve the access control problem; an unauthorized internal client could still establish a valid TLS connection and call the API.
C. You should recommend the use of custom secure headers with time stamp verification: This creates a custom, non-standard security solution that is difficult to manage, scale, and audit. It is better to use a standard API management component like a microgateway for this purpose.

References:

1. Google Cloud - Apigee Documentation, "About Apigee hybrid": This document explains the role of the hybrid runtime plane, which is based on a microgateway architecture. It states, "All API traffic passes through and is processed within the runtime plane... This allows you to leverage Apigee API management capabilities like security, traffic management, and analytics...". This supports using a microgateway as the policy enforcement point for API traffic.
Reference: Google Cloud Documentation Apigee Hybrid gcp/apigee/hybrid/v1.8/about-hybrid
2. Google Cloud - Apigee Documentation, "Microgateway versus API proxy": This section clarifies the use case for a microgateway. It highlights that a key reason to use a microgateway is for "low latency for services that run in close proximity," which is typical for internal traffic between a website's backend and other backend services within the same network or cloud environment.
Reference: Google Cloud Documentation Apigee Edge private-cloud/v4.50.00/edge-microgateway-overview#microgatewayversusapiproxy
3. Google Cloud - Apigee Documentation, "Access Control policy": The documentation for this policy states, "The AccessControl policy lets you allow or deny access to your APIs by specific IP addresses." This confirms its function is access control, but its placement within an API proxy flow means it is only effective if traffic is routed through the Apigee gateway.
Reference: Google Cloud Documentation Apigee Edge api-platform/reference/policies/access-control-policy