DUMPSARENA

Automating Cisco Data Center Solutions (DCAUTO)

Cisco 300-635

Version Demo

Total Demo Questions: 13

Total Premium Questions: 136

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Data Center Automation and Orchestration	29
Topic 2, Data Center Device-Centric Automation	59
Topic 3, Data Center Infrastructure as Code	6
Topic 4, Data Center Automation with Cisco ACI	42
Total	136

QUESTION NO: 1

Which two characteristics apply to RESTCONF? (Choose two.)

A. It uses HTTP methods to access YANG-defined data.

B. It supports JSON and XML data encodings.

C. It uses SNMP OIDs as its resource paths.

D. It requires XML request and response bodies.

E. It does not support configuration changes.

ANSWER: A B

Explanation:

RESTCONF provides a programmatic interface for accessing YANG-defined data by using HTTP methods. A RESTCONF client can use GET to retrieve modeled configuration or state data and can use methods such as PATCH, PUT, POST, and DELETE to create, update, or remove resources as supported by the server.

RESTCONF resource paths identify data resources in a YANG datastore hierarchy. RESTCONF is not a replacement for YANG; it uses YANG models to define the data that clients access. It also does not require XML exclusively, because RESTCONF supports JSON and XML encodings. See RFC 8040: RESTCONF Protocol.

QUESTION NO: 2

Which two capabilities apply to the DCNM API? (Choose two.)

A. DCNM provides an XML-based SOAP API.

B. DCNM requires a license to use the API.

C. Some features of DCNM must be configured through the GUI.

D. All API operations can be performed using the DCNM GUI.

E. DCNM provides a REST-based API.

ANSWER: A E

Explanation:

DCNM supports programmatic integration through both an XML-based SOAP interface and a REST-based interface. The XML/SOAP API is intended for structured web-service interactions in environments that use SOAP messaging and XML payloads. It enables external applications and orchestration systems to invoke DCNM functions without requiring an operator to perform each task manually in the user interface.

The REST-based API provides an HTTP-oriented method for automation clients to interact with DCNM resources and operational functions. REST calls are well suited to scripts and automation frameworks because clients can use standard HTTP methods, authenticate to the controller, submit structured request bodies, and consume returned resource data. This API model is commonly used to automate fabric operations, retrieve inventory or status information, and integrate DCNM with higher-level tools.

Providing both interfaces supports different integration requirements: existing enterprise applications can use SOAP-based web services, while newer automation workflows can use REST. Cisco's DCNM documentation and software resources are available through the Cisco DCNM support page, and Cisco's current successor platform API documentation is available from the Cisco Developer Nexus Dashboard Fabric Controller documentation.

QUESTION NO: 3

Which Kubernetes container network interface provides intent-based networking from the same pane of glass that VMs and bare-metal servers are managed?

A. ACI CNI plug-in

B. Contiv CNI plug-in

C. Ingress CNI plug-in

D. Calico CNI plug-in

ANSWER: A

Explanation:

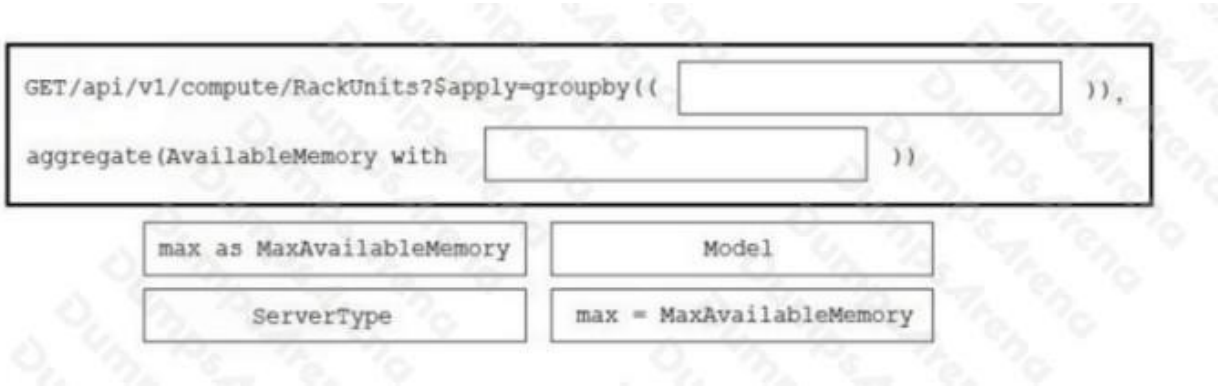
ACI CNI plug-in is correct because it integrates Kubernetes container networking with Cisco Application Centric Infrastructure, allowing container workloads to be governed through the same policy-driven fabric and operational interface used for virtual-machine and bare-metal workloads. Cisco ACI uses an application-centric, intent-based policy model: administrators define connectivity and security requirements, and the fabric programs the required network behavior. The ACI CNI plug-in extends this model into Kubernetes by connecting pods and Kubernetes services to the ACI fabric policy domain. This gives network and platform teams consistent visibility, segmentation, policy enforcement, and operational control across traditional data-center endpoints and containerized applications rather than creating a separate networking-management domain for the cluster. The integration is designed to let Kubernetes workloads participate in ACI constructs such as endpoint groups and contracts, while retaining Kubernetes-native application deployment workflows. Cisco documents that the ACI CNI deployment provides connectivity for Kubernetes workloads using the Cisco ACI fabric and is managed through the APIC controller, which is the centralized management and policy platform for ACI. See the Cisco ACI Containers Installation Guide and Cisco Application Centric Infrastructure overview.

QUESTION NO: 4 - (DRAG DROP)

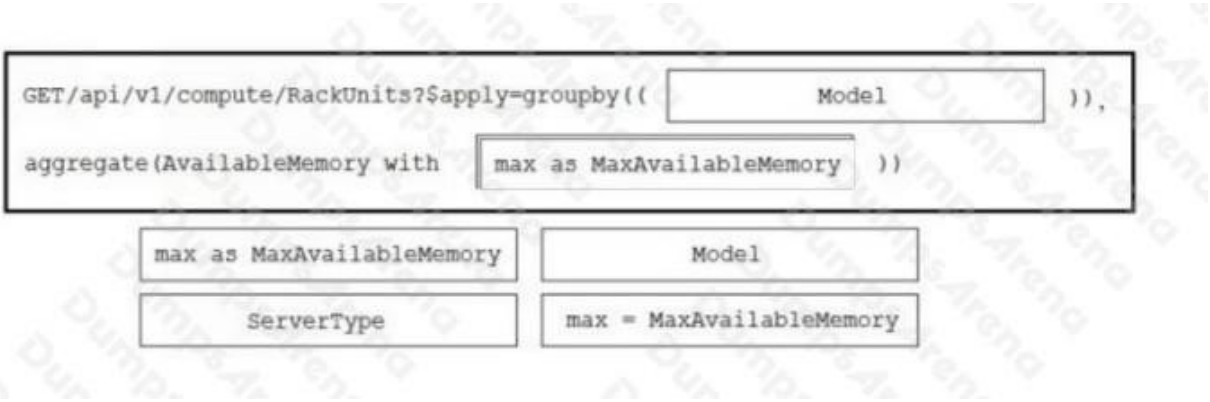
DRAG DROP

A co-worker is using Cisco Intersight to determine the maximum available memory per server for their company's data center. Drag and drop the code to complete the Cisco Intersight API call that provides the desired results. Not all options are used.

Select and Place:



ANSWER:



Explanation:

The completed request correctly uses Cisco Intersight's OData-style `$apply` query processing to summarize rack-unit inventory data rather than returning every individual rack-unit record. The `groupby` transformation uses `Model`, so the API organizes the returned RackUnits into groups based on their server model. This makes the output useful for capacity planning because each returned result represents a model grouping instead of an unaggregated list of servers.

Within each model group, `aggregate(AvailableMemory with max as MaxAvailableMemory)` evaluates the `AvailableMemory` property and selects its greatest value. The `max` aggregate is the appropriate operation when the requirement is to determine the maximum available memory. Naming the computed output `MaxAvailableMemory` provides a clear property name in the response, distinguishing the calculated result from the original per-device `AvailableMemory` field.

Therefore, the complete expression returns a compact result set containing a model value and the highest available-memory value found among rack units in that model group. This follows the standard OData aggregation structure: group records first, then apply an aggregate method to a numeric property, and assign an alias to the resulting calculated property. Cisco Intersight exposes its REST resources through an API designed for query parameters and filtering, while OData documents the `groupby` and `aggregate` transformations used in this kind of request. See the Cisco Intersight API documentation and the OData Data Aggregation Extension for the grouping and aggregation model.

QUESTION NO: 5

Which procedure accesses the REST API browser within Cisco UCS Director?

A. Send an HTTP GET request to https://[UCS Director IP]/api/get_resources/.

B. Log in as the user REST/user to access the REST API interface.

C. Enable the Developer menu. Select Orchestration in the UI, then select the REST API browser tab.

D. Select the API browser from the Cisco UCS Director End User Portal catalog of services.

ANSWER: C

Explanation:

Enable the Developer menu. Select Orchestration in the UI, then select the REST API browser tab. Cisco UCS Director exposes its interactive REST API Browser through the administrative user interface rather than through an End User Portal catalog item or a specific resource endpoint. The Developer menu must first be enabled so that the Orchestration area presents development-oriented tools. From there, opening the REST API Browser tab provides an interface for browsing available REST operations, supplying parameters, executing requests, and viewing returned data. This is useful when validating an API call before incorporating it into an automation workflow, script, or orchestration task. The browser is therefore a UI feature reached through the Developer and Orchestration navigation path, with access governed by the authenticated user's UCS Director role and privileges. Cisco's UCS Director programming documentation describes the REST API facilities and their use for programmatic interaction with UCS Director resources: Cisco UCS Director Programming Reference Guides. Additional UCS Director documentation is available through the Cisco UCS Director documentation portal.

QUESTION NO: 6

A set of automation scripts work with no issue from a local machine, but an experiment needs to take place with a new package found online.

How is this new package isolated from the main code base?

A. Add the new package to your requirements.txt file.

B. Create a new virtual machine and perform a pip install of the new package.

C. Perform a pip install of the new package when logged into your local machine as root.

D. Create a new virtual environment and perform a pip install of the new package.

ANSWER: D

Explanation:

Create a new virtual environment and perform a pip install of the new package. A Python virtual environment is an isolated directory containing its own Python interpreter context and installed-package set. Activating that environment before using `pip install` places the experimental package and any of its dependencies in the environment rather than in the system-wide Python installation or the dependency set used by the working automation scripts. This makes it possible to test the package safely, including its required versions, without introducing dependency conflicts or unintended changes to the main code base's established runtime. If the experiment succeeds, its dependencies can subsequently be deliberately documented and incorporated into the project's dependency-management process. If it does not, deleting the virtual-environment directory cleanly removes the experimental installation. Python's standard `venv` module is specifically intended for creating lightweight, disposable, and isolated environments for projects and experiments. Use `python -m venv <environment-name>`, activate the environment for the current shell, and then run `python -m pip install <package-name>` to ensure pip targets that environment. See the Python venv documentation and the Python Packaging User Guide for pip and virtual environments.

QUESTION NO: 7

Refer to the exhibit.

Which two statements are true about this API GET request to the ACI APIC? (Choose two.)

A. The API call creates a new 10G interface in the APIC

B. The API call reads information from a managed object.

C. The API response is encoded in JSON

D. The API call reads information from an object class

E. The API response is encoded in XML

ANSWER: B D

Explanation:

The statements "The API call reads information from a managed object." and "The API call reads information from an object class" are correct because Cisco ACI exposes its management information through a hierarchical object model. A REST request using the HTTP GET method retrieves existing APIC state rather than changing the fabric configuration. In this request, the managed-object portion identifies the object, or distinguished-name context, from which data is retrieved. The class portion identifies the managed-object type that APIC is to return or select within that context. This is a common ACI API pattern: a client can retrieve a specific managed object and use a class-based target to obtain instances of a particular object class associated with that object or its subtree. The returned data represents APIC's managed-object model and can then be consumed by automation tooling for inventory, operational-state checks, or configuration validation. Cisco documents managed-object and class queries as core REST API mechanisms in its ACI API documentation. See the Cisco ACI developer documentation and the Cisco APIC REST API Configuration Guide.

QUESTION NO: 8

What is a characteristic of synchronous and asynchronous API calls?

A. Synchronous API calls must always use a proxy server.

B. ASynchronous communication uses more overhead for client authentication

C. Synchronous communication is harder to follow and troubleshoot

D. Synchronous API calls wait to return until a response has been received

ANSWER: D

Explanation:

Synchronous API calls wait to return until a response has been received. In this request-response pattern, the client sends a request and blocks execution of the calling flow until the server completes processing and returns a response, error, or timeout. This behavior makes the result immediately available to the caller after the operation returns, which is useful when subsequent logic depends directly on the returned data or confirmation that an action completed. For example, automation might synchronously retrieve a device configuration or submit a request whose response contains an identifier needed for the next API operation. The client must therefore account for network latency and server processing time by setting appropriate connection and request timeouts. In contrast, an asynchronous design lets the client continue working after submission and obtains the eventual result through mechanisms such as polling, callbacks, webhooks, or a job-status endpoint. Cisco automation commonly uses REST APIs, whose HTTP request-response exchange is synchronous at the transport interaction level, although an API can accept work asynchronously and return a task resource for later status retrieval. See the HTTP Semantics specification (RFC 9110) and Cisco's Cisco Developer documentation for API and automation guidance.

QUESTION NO: 9

Which two statements about gRPC are true? (Choose two.)

A. It is an IETF draft.

B. It is an IETF standard.

C. It runs over SSH.

D. It is an open source initiative.

E. It runs over HTTPS.

ANSWER: D E

Explanation:

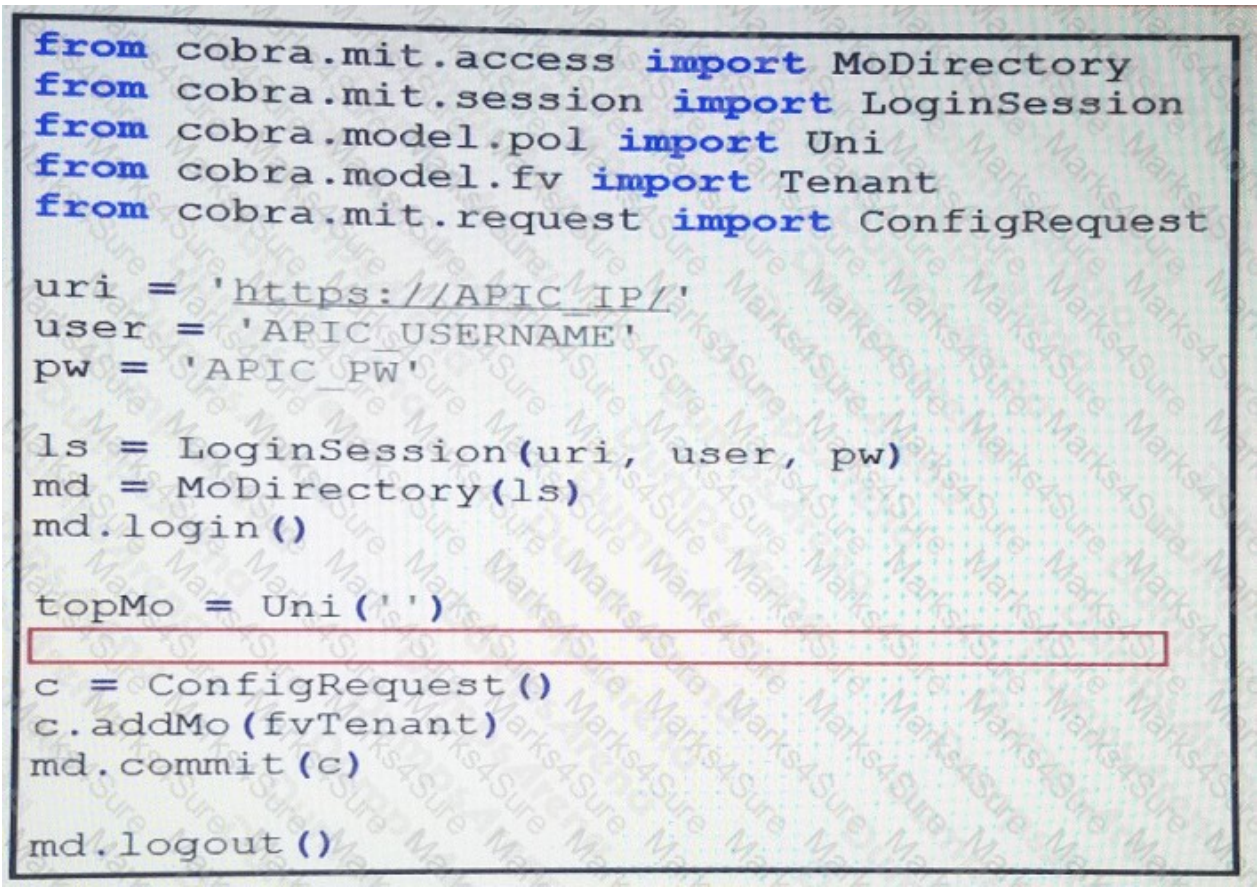
gRPC is an open-source, high-performance remote procedure call framework originally created at Google and now developed openly by the gRPC community. Its open development model provides public source code, documentation, and implementations for numerous languages, enabling clients and services written in different languages to communicate through a common RPC framework. The project's source and governance materials are publicly available through the gRPC organization and repository.

gRPC uses HTTP/2 as its underlying transport protocol. HTTP/2 supports features that are central to gRPC's design, including multiplexed streams, bidirectional streaming, header compression, and flow control. In production deployments, gRPC endpoints are commonly protected with TLS; HTTP/2 used with TLS is generally accessed using the `https` scheme. Thus, "It runs over HTTPS" accurately reflects the common secure deployment model, while the more precise protocol description is that gRPC runs over HTTP/2 and can use TLS for transport security.

See the official gRPC introduction for its HTTP/2-based architecture and the gRPC GitHub repository for the open-source project.

QUESTION NO: 10

Refer to the exhibit:



```
from cobra.mit.access import MoDirectory
from cobra.mit.session import LoginSession
from cobra.model.pol import Uni
from cobra.model.fv import Tenant
from cobra.mit.request import ConfigRequest

uri = 'https://APIC_IP/'
user = 'APIC_USERNAME'
pw = 'APIC_PW'

ls = LoginSession(uri, user, pw)
md = MoDirectory(ls)
md.login()

topMo = Uni('')

c = ConfigRequest()
c.addMo(fvTenant)
md.commit(c)

md.logout()
```

Refer to the exhibit: The code should create a new tenant named Cisco via the Cobra SDK, which shows up after the execution of this script in the APIC dashboard.

Which code must be inserted into the red box to create this tenant?

A. Tenant = NewTenant(name= ' Cisco ')

B. tenant — Tenant(topMo, name-Cisco ')

C. Tenant = Tenant(topMo, name= ' Cisco ')

D. Tenant = Tenant(' Cisco ')

ANSWER: C

Explanation:

`Tenant = Tenant(topMo, name= ' Cisco ' )` correctly instantiates the ACI tenant managed object in the Cobra object model. In Cobra, a managed object is created by calling its model class with its parent managed object as the first argument and supplying naming properties as keyword arguments. Here, `topMo` represents the existing top-level policy universe object, commonly the `polUni` managed object obtained from APIC. Passing `topMo` establishes the required parent-child relationship, while the `name` keyword supplies the tenant's naming value, Cisco. The resulting Python object represents the tenant in the locally built configuration tree. When the script subsequently commits that tree through a Cobra `ConfigRequest` using the APIC MoDirectory session, APIC persists the new tenant and it becomes visible in the dashboard. This reflects Cobra's declarative managed-object workflow: construct the object beneath the appropriate distinguished-name parent, then submit the configuration request to APIC. Cisco's Cobra SDK source and examples are available in the Cisco Datacenter Cobra repository; APIC automation concepts and API configuration guidance are also documented in Cisco's APIC developer documentation.

QUESTION NO: 11

Which two statements about YANG data models are true? (Choose two.)

A. It can model configuration and operational state data.

B. It can define RPC operations and notifications.

C. It is an HTTP transport protocol.

D. It can be used only with NETCONF.

E. It is limited to vendor-neutral models.

ANSWER: A B

Explanation:

YANG is a data modeling language used to model configuration data, operational state data, RPC operations, and notifications. Its hierarchical structure enables an API client to address structured data consistently through model-defined paths. Cisco platforms use YANG models with programmable interfaces such as NETCONF, RESTCONF, and model-driven telemetry.

YANG models can be defined by standards organizations, vendors, or users. IETF YANG modules provide standardized definitions, while Cisco native YANG models expose platform-specific features and data. YANG is not itself a transport protocol and does not replace the API protocols used to exchange modeled data. See RFC 7950: The YANG 1.1 Data Modeling Language and the Cisco IOS XE YANG models documentation.

QUESTION NO: 12

How is an ACI class name composed?

A. :Method

B. ClassName:Method

C. Package:ClassName

D. MitName:Method

ANSWER: C

Explanation:

In the Cisco ACI Management Information Tree (MIT) object model, a class name is composed as **Package:ClassName**. The package identifies the functional area or managed-object namespace, and the class name identifies the particular managed object within that package. For example, tenant-related managed objects are in the `fv` package, so a tenant is represented as `fv:Tenant` in the model notation. This package-and-class format lets automation clients and developers identify object types consistently across the APIC data model.

This naming is important when working with the APIC REST API, SDKs, model documentation, and metadata. Although REST API URLs commonly use the concatenated API class form, such as `fvTenant`, ACI documentation expresses the underlying MIT class identity using the package-qualified `package:ClassName` convention. Understanding that relationship helps when translating between model references, API endpoint examples, and SDK object definitions. Cisco's APIC API reference documents managed-object classes and their package context, while the ACI programming documentation describes the MIT-based object model used by APIC. See the Cisco APIC REST API documentation and Cisco ACI developer documentation.

QUESTION NO: 13 - (DRAG DROP)

DRAG DROP

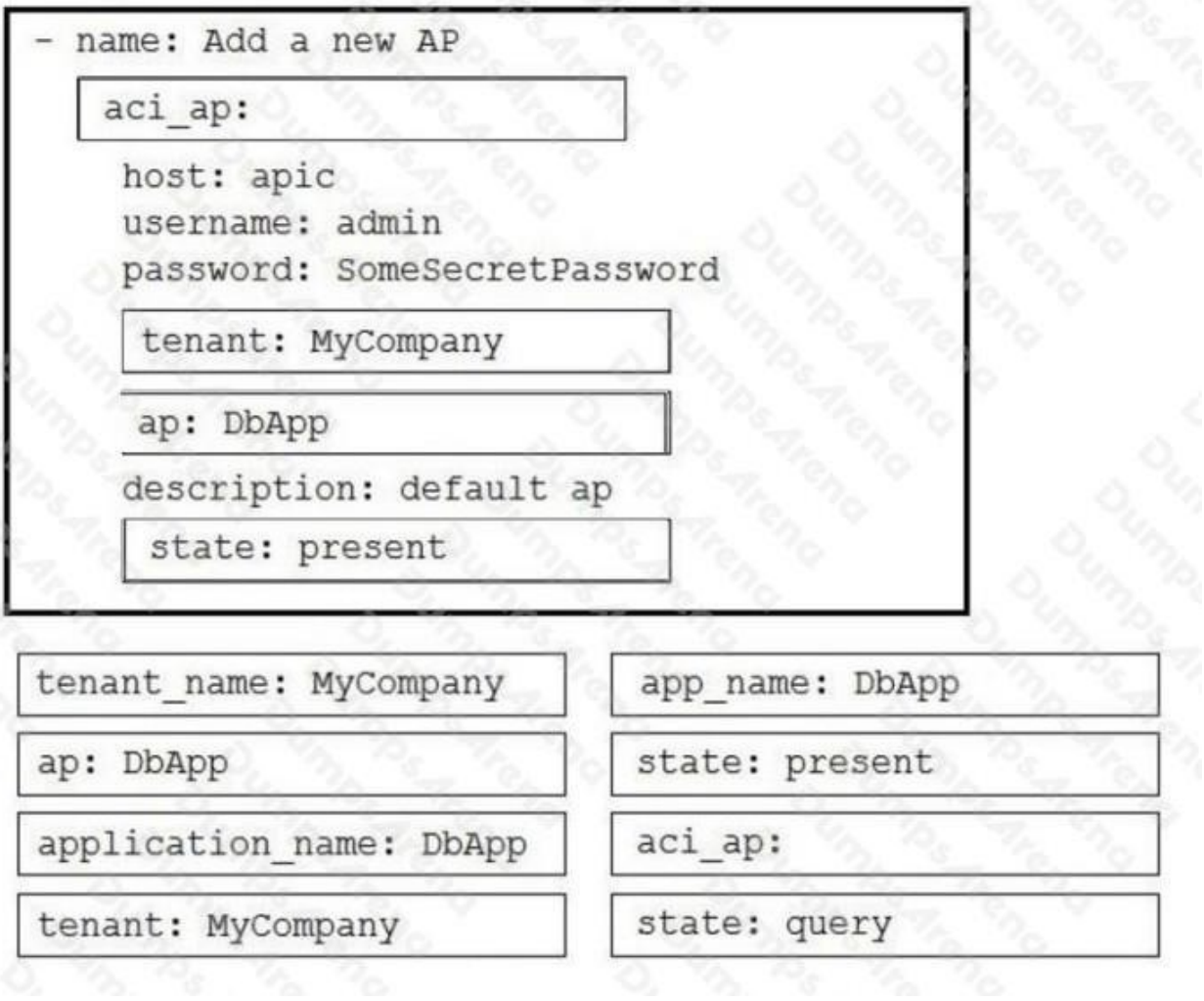
Drag and drop the correct YAML components from the bottom onto the correct blanks within the Ansible playbook to create a new application profile called "DbApp" using the Ansible ACI module. Not all options are used.

Select and Place:



tenant_name: MyCompany	app_name: DbApp
ap: DbApp	state: present
application_name: DbApp	aci_ap:
tenant: MyCompany	state: query

ANSWER:



Explanation:

The completed task correctly uses `aci_ap:` as the Ansible ACI application-profile module. This module manages an application profile within a Cisco ACI tenant, so it is the appropriate module declaration directly beneath the task name. The connection values shown for the APIC host, username, and password provide the controller access details used by the module.

The `tenant: MyCompany` setting identifies the tenant that will contain the application profile. In Cisco ACI, application profiles are tenant-scoped objects, so the tenant must be supplied to establish the proper configuration hierarchy. The `ap: DbApp` setting provides the application-profile name, thereby defining the requested new profile named DbApp. The existing description line is included as an optional descriptive attribute for that same application-profile object.

Finally, `state: present` expresses the desired declarative result: the DbApp application profile must exist in the MyCompany tenant. When Ansible runs the playbook, it creates the profile if it does not already exist and otherwise maintains it with the specified attributes. This is the standard idempotent Ansible approach for configuring ACI managed objects. Cisco's ACI collection documentation for the application-profile module describes the tenant, application-profile, description, and state parameters used in this task: Cisco ACI aci_ap module documentation. General information on using Cisco ACI modules through Ansible is also available in the Cisco ACI Ansible collection documentation.