**Cisco ACI & APIC**

ACI – Application Centric Infrastructure

SDN Based Solution

The solution is based on two components:

1. Cisco Nexus family of switches
2. Cisco Application Policy Infrastructure Controller (APIC)

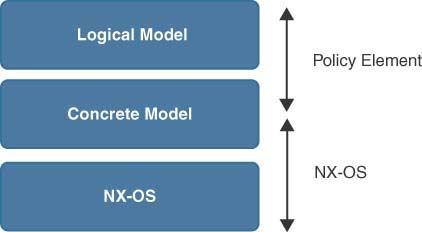
The Cisco Nexus 9000 family of switches can run in two separate modes of operation:

1. standalone (or NX-OS) mode
2. ACI mode

Some of the features and capabilities of the Cisco APIC are as follows:

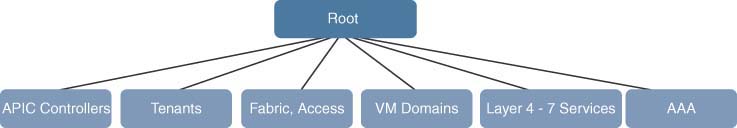
* Application-centric network policy for physical, virtual, and cloud infrastructure
* Data model–based declarative provisioning
* Designed around open standards and open APIs
* Cisco ACI fabric inventory and configuration
* Software image management
* Fault, event, and performance monitoring and management
* Integration with third-party management systems such as VMware, Microsoft, and OpenStack
* Cloud APIC appliance for Cisco cloud ACI deployments in public cloud environments

A minimum of three APICs in a cluster are needed for high availability.



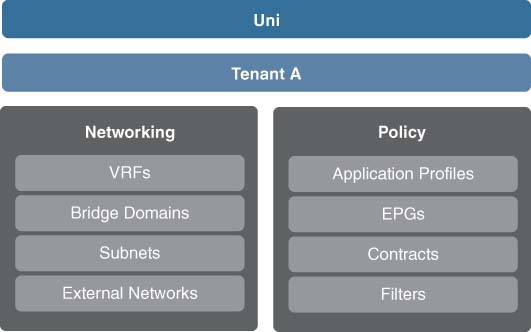
The ACI policy model

Overview of the MIT and its elements.



**Building Blocks of Cisco ACI Fabric Policies**

Tenants are top-level MOs that identify and separate administrative control, application policies, and failure domains. A tenant can represent a customer in a managed service provider environment or an organization in an enterprise environment, or a tenant can be a convenient grouping of objects and policies. A tenant’s sublevel objects can be grouped into two categories: tenant networking and tenant policy



**APIC REST API**

As mentioned previously, the APIC REST API is a programmatic interface that uses the REST architecture. The API accepts and returns HTTP or HTTPS messages that contain JSON or XML documents.

The generic APIC REST API URI looks as follows:

**https://{APIC\_Host}:{port}/api/{mo|class}/{dn|classname}.{xml|json}?[options]**

The complete Cisco ACI REST API documentation with information on how to use the API can be found at <https://developer.cisco.com/docs/aci/> .

the Cisco DevNet always-on APIC instance is available at [https://sandboxapicdc.cisco.com](https://sandboxapicdc.cisco.com/)

username : **admin**

password : **ciscopsdt**

ACI Python SDK called Cobra can be used for advanced development. For basic day-to-day configuration, there is a Python library called **acitoolkit**.

**Visore:** the Application Policy Infrastructure Controller (APIC) Object Store Browser.

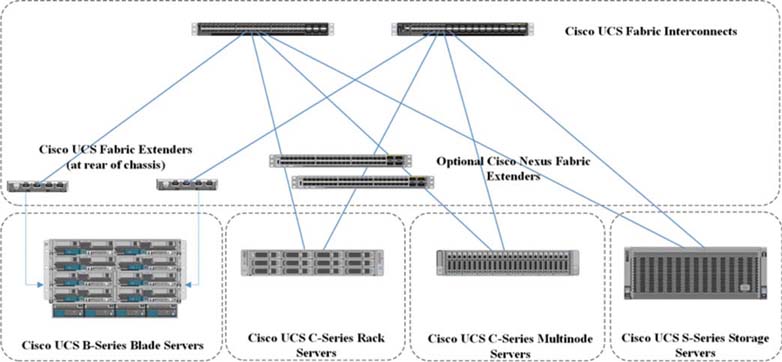
https://sandboxapicdc.cisco.com/visore.html#

**UCS Manager**

Cisco UCS B-series blade servers, C-series rack servers, S-series storage servers, UCS Mini, and Cisco HyperFlex hyperconverged servers can all be managed through one interface: UCS Manager.

UCS Manager provides unified, embedded management of all software and hardware components of Cisco UCS.

Cisco UCS Manager software runs on a pair of hardware appliances called fabric interconnects. The two fabric interconnects form an active/standby cluster that provides high availability.



With Cisco UCS Manager, the data center servers can be managed using an infrastructure-as-code framework. This is possible through another innovation that is included with the Cisco UCS solution: the service profile. The service profile is a logical construct in UCS Manager that contains the complete configuration of a physical server.

At the top of the hierarchical structure is the **sys** object, which contains all the parent and child nodes in the tree. Each object in Cisco UCS has a unique distinguished name that describes the object and its place in the tree. The information model is centrally stored and managed by a process running on the fabric interconnects that is called the Data Management Engine (DME).

A specific managed object in the MIT can be identified by its distinguished name (DN) or by its relative name (RN). The DN specifies the exact managed object on which the API call is operating and consists of a series of relative names:

DN = {RN}/{RN}/{RN}/{RN}...

A relative name identifies an object in the context of its parent object.

The Cisco UCS Manager XML API model includes the following programmatic entities:

* **Classes:** Classes define the properties and states of objects in the MIT.
* **Methods:** Methods define the actions that the API performs on one or more objects.
* **Types:** Types are object properties that map values to the object state.

Several types of methods are available with the XML API:

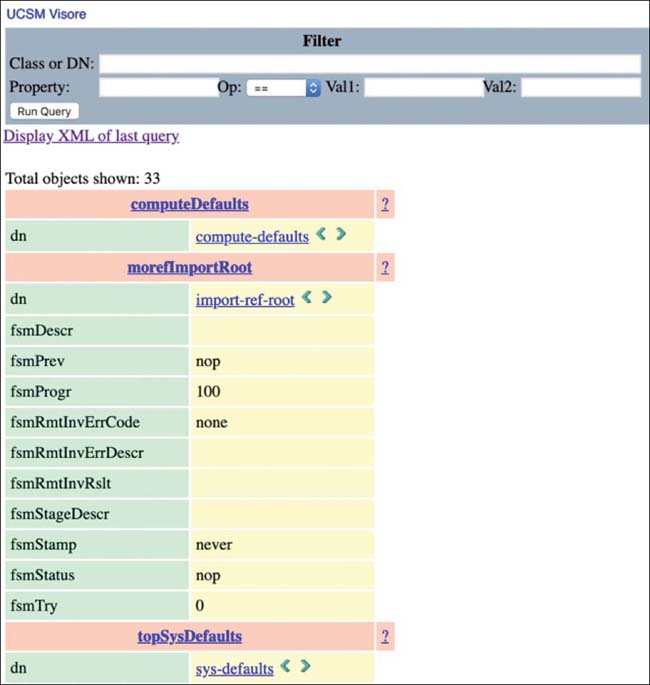
* **Authentication methods:** These methods, which include the following, are used to authenticate and maintain a session
* **Query methods:** These methods, which include the following, are used to obtain information on the current configuration state of an object
* **Configuration methods:** These methods, which include the following, are used to make configuration changes to managed objects

Since the query methods available with the XML API can return large sets of data, filters are supported to limit this output to subsets of information. Four types of filters are available:

* **Simple filters –** Boolean type
* **Property filters –** Property value is compared
* **Composite filters –** Logical AND OR
* **Modifier filter –** Change based

External applications can get Cisco UCS Manager state change information either by regular polling or by event subscription. Full event subscription is supported with the API and is the preferred method of notification. Polling usually consumes a lot of resources and should be used only in limited situations.

Cisco UCS Manager provides a managed object browser called Visore. Visore can be accessed by navigating to https://<*UCS-Manager-IP*>/visore.html.



<https://developer.cisco.com/site/ucs-dev-center/>.

<https://developer.cisco.com/site/ucs-mim-ref-api-picker/>.

The Cisco UCS Platform Emulator can be used to create and test a supported Cisco UCS configuration or to duplicate an existing Cisco UCS environment for troubleshooting and development purposes. The Cisco UCS Platform Emulator is delivered as an .ova file and can run in nearly any virtual environment. The complete Cisco UCS Manager information model documentation is also bundled within the UCS Platform Emulator.

We will require a reservable sandbox to try out the UCSM API.

<https://developer.cisco.com/docs/sandbox/#!data-center/data-center-sandbox-highlights>

curl -k -X POST https://10.10.20.110/nuova \

-H 'Content-Type: application/xml' \

-d '<aaaLogin inName="ucspe" inPassword="ucspe"></aaaLogin>'

curl -k -X POST https://10.10.20.110/nuova \

-H 'Content-Type: application/xml' \

-d '<configFindDnsByClassId

classId="computeItem"

cookie="1573019916/7c901636-c461-487e-bbd0-c74cd68c27be" />'

curl -k -X POST https://10.10.20.110/nuova \

-H 'Content-Type: application/xml' \

-d '<configResolveDn

cookie="1573019916/7c901636-c461-487e-bbd0-c74cd68c27be"

dn="sys/chassis-4/blade-8" />'

Automation Tools for UCSM

The Cisco UCS PowerTool suite is a PowerShell module that helps automate all aspects of Cisco UCS Manager. The PowerTool cmdlets work on the Cisco UCS MIT. The cmdlets can be used to execute read, create, modify, and delete operations on all the managed objects in the MIT.

The Cisco UCS PowerTool suite enables easy integration with existing IT management processes and tools. The PowerTool suite can be downloaded for Windows via PS Gallery and for Linux from <https://community.cisco.com/t5/cisco-developed-ucs-integrations/cisco-ucs-powertool-core-suite-for-powershell-core-modules-for/ta-p/3985798>.

The Cisco UCS Python SDK works on the Cisco UCS Manager MIT and CRUD actions on the managed objects in the tree.

Python versions 2.7 and higher and version 3.5 and higher are supported.

C:\Users\sucvenka\Desktop\Week 2\svs-devnet-handson> pip install ucsmsdk

Collecting ucsmsdk

Downloading ucsmsdk-0.9.10.tar.gz (4.2 MB)

|████████████████████████████████| 4.2 MB 329 kB/s

Requirement already satisfied: setuptools in c:\users\sucvenka\appdata\local\programs\python\python37\lib\site-packages (from ucsmsdk) (53.0.0)

Collecting pyparsing

Using cached pyparsing-2.4.7-py2.py3-none-any.whl (67 kB)

Requirement already satisfied: six in c:\users\sucvenka\appdata\local\programs\python\python37\lib\site-packages (from ucsmsdk) (1.13.0)Using legacy setup.py install for ucsmsdk, since package 'wheel' is not installed.

Installing collected packages: pyparsing, ucsmsdk

Running setup.py install for ucsmsdk ... done

Successfully installed pyparsing-2.4.7 ucsmsdk-0.9.10

PS C:\Users\sucvenka\Desktop\Week 2\svs-devnet-handson>

*Code Output:*

PS C:\Users\sucvenka\Desktop\Week 2\svs-devnet-handson> & C:/Users/sucvenka/AppData/Local/Programs/Python/Python37/python.exe "c:/Users/sucvenka/Desktop/Week 2/svs-devnet-handson/Week4/UCSMDemo.py"

DN SERIAL ADMIN STATE MODEL TOTAL MEMORY

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

sys/chassis-3/blade-1 SRV126 in-service UCSB-EX-M4-1 49152

sys/chassis-3/blade-3 SRV127 in-service UCSB-EX-M4-1 49152

sys/chassis-3/blade-5 SRV128 in-service UCSB-EX-M4-1 49152

sys/chassis-3/blade-7 SRV129 in-service UCSB-EX-M4-1 49152

sys/chassis-4/blade-1 SRV130 in-service UCSB-B200-M4 49152

sys/chassis-4/blade-2 SRV131 in-service UCSB-B200-M4 49152

sys/chassis-4/blade-3 SRV132 in-service UCSB-B200-M4 49152

sys/chassis-4/blade-4 SRV133 in-service UCSB-B200-M4 49152

sys/chassis-4/blade-5 SRV134 in-service UCSB-B200-M4 49152

sys/chassis-4/blade-6 SRV135 in-service UCSB-B200-M4 49152

sys/chassis-4/blade-7 SRV136 in-service UCSB-B200-M4 49152

sys/chassis-4/blade-8 SRV137 in-service UCSB-B200-M4 49152

sys/chassis-5/blade-1 SRV138 in-service UCSB-B200-M5 49152

sys/chassis-5/blade-2 SRV139 in-service UCSB-B200-M5 49152

sys/chassis-5/blade-3 SRV140 in-service UCSB-B200-M5 49152

sys/chassis-5/blade-4 SRV141 in-service UCSB-B200-M5 49152

sys/chassis-5/blade-5 SRV142 in-service UCSB-B200-M5 49152

sys/chassis-5/blade-6 SRV143 in-service UCSB-B200-M5 49152

sys/chassis-5/blade-7 SRV144 in-service UCSB-B200-M5 49152

sys/chassis-5/blade-8 SRV145 in-service UCSB-B200-M5 49152

sys/chassis-6/blade-1 SRV146 in-service UCSB-B200-M4 49152

sys/chassis-6/blade-2 SRV147 in-service UCSB-B200-M4 49152

sys/chassis-6/blade-3 SRV148 in-service UCSB-B200-M4 49152

sys/chassis-6/blade-4 SRV149 in-service UCSB-B200-M4 49152

sys/chassis-6/blade-5 SRV150 in-service UCSB-B200-M4 49152

sys/chassis-6/blade-6 SRV151 in-service UCSB-B200-M4 49152

sys/chassis-6/blade-7 SRV152 in-service UCSB-B200-M4 49152

sys/chassis-6/blade-8 SRV153 in-service UCSB-B200-M4 49152

PS C:\Users\sucvenka\Desktop\Week 2\svs-devnet-handson>

**UCS Director**

Cisco UCS Director replaces manual configuration and provisioning processes with orchestration in order to optimize and simplify delivery of data center resources.

Cisco UCS Director provides comprehensive visibility and management of data center infrastructure components. From a data center management perspective, the following are some of the tasks that can be performed using Cisco UCS Director:

* Create, clone, and deploy service profiles and templates for all Cisco UCS servers and compute applications.
* Manage, monitor, and report on data center components such as Cisco UCS domains or Cisco Nexus devices.
* Monitor usage, trends, and capacity across a converged infrastructure on a continuous basis.
* Deploy and add capacity to converged infrastructures in a consistent, repeatable manner.

Cisco UCS Director is supported by a broad ecosystem. Third-party hardware and solution vendors can use the southbound APIs and the SDKs provided with them to develop integrations into the Cisco UCS Director management model. Northbound APIs can be used by DevOps and IT operations management tools to interact with Cisco UCS Director and perform all the functions provided by the solution in a programmable and automated fashion.

**UCSD Essential Concepts:**

***Task :*** Atomic unit of work which represents a single action with inputs and outputs.

Task Library examples:

* SSH command task (executing a command in a Secure Shell session)
* an inventory collection task (gathering information about available devices)
* new VM provisioning task (creating a new virtual machine),

For tasks which are not predefined in the task library, there is scope to create custom tasks.

***Workflow :*** A *workflow* is a series of tasks arranged to automate a complex operation. The simplest workflow contains a single task, but workflows can contain any number of tasks. Workflows are at the heart of Cisco UCS Director orchestration. They automate processes of any level of complexity.

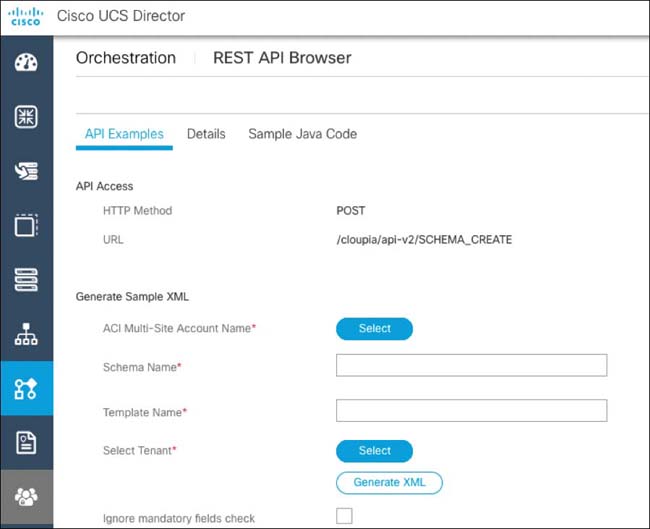
Workflows are built using the Workflow Designer, which is a drag-and-drop interface. In Workflow Designer, the tasks are arranged in sequence and define inputs and outputs to those tasks. Loops and conditionals can be implemented using flow of control tasks. Every time a workflow is executed, a service request is generated. Workflows can be scheduled for later execution, and Cisco UCS Director stores details of completed service requests. A service request can have one of several states, depending on its execution status: scheduled, running, blocked, completed, or failed.

***Libraries and catalogs :*** They are collections of predefined tasks and workflows that can be used as building blocks for more advanced workflows.

A module is the topmost logical entry point into Cisco UCS Director. In order to add or extend the functionality of the system, a new module must be developed and deployed on Cisco UCS Director. A module developed using Open Automation behaves the same way as any Cisco UCS Director built-in feature or module.

Cisco UCS Director offers a REST API that enables applications to consume or manipulate the data stored in Cisco UCS Director. Applications use HTTP or HTTPS requests from the REST API to perform Create/Read/Update/Delete (CRUD) operations on Cisco UCS Director resources. With an API call, a developer can execute Cisco UCS Director workflows and change the configuration of switches, adapters, policies, and any other hardware and software components. The API accepts and returns HTTP messages that contain JavaScript Object Notation (JSON) or Extensible Markup Language (XML) documents.

Within the user advanced settings is an option to enable the developer menu. By enabling the developer menu, access to the REST API browser and the Report Metadata features is turned on. The REST API browser becomes visible under the Orchestration tab of Cisco UCS Director and provides API information and API code generation capabilities for all available APIs. The Report Metadata option becomes available on all the pages of the Cisco UCS Director GUI; when selected, it returns the API code that the GUI is using to retrieve the information that is displayed to the user in that specific page. This code includes a complete URL that is ready to paste into a browser to send the request to Cisco UCS Director. Both the REST API browser and the Report Metadata features are extremely valuable to developers as they provide ready-to-use sample code and API calls to all the resources available in Cisco UCS Director.



Each REST API request must be associated with an HTTP header called **X-Cloupia-Request-Key**, with its value set to the REST API access key retrieved previously. The REST API request must contain a valid URL of the following format:

**https://Cisco\_UCS\_Director/app/api/rest?formatType=json&opName=operationName&opData=operationData**

where

* **Cisco\_UCS\_Director:** This is the IP address or hostname of the Cisco UCS Director VM.
* **formatType:** This can be either JSON or XML; it is JSON in this case. (Only JSON is discussed throughout the rest of this chapter.)
* **opName:** This is the API operation name that is associated with the request (for example, userAPIGetMyLoginProfile), as explored later in this chapter.
* **opData:** This contains the parameters or the arguments associated with the operation. Cisco UCS Director uses JSON encoding for the parameters. If an operation doesn’t require any parameters, the empty set {} should be used. When building the URL, escape characters should be encoded as appropriate.

Programming guides and complete documentation of the Cisco UCS REST API can be found at the following link: <https://www.cisco.com/c/en/us/support/servers-unified-computing/ucs-director/products-programming-reference-guides-list.html>. The Cisco DevNet team makes available a reservable sandbox called “UCS Management” for learning purposes. This sandbox contains an installation of Cisco UCS Director and is available at <https://developer.cisco.com/sandbox>.

curl -k -L -X GET \

-g 'http://10.10.10.66/app/api/rest?formatType=json&opName=userAPIGetWorkflowInput

s&opData={param0:%22VMware%20OVF%20Deployment%22}' \

-H 'X-Cloupia-Request-Key: 8187C34017C3479089C66678F32775FE'

**Intersight**

Some of the benefits of using Cisco Intersight are the following:

* It simplifies Cisco UCS and Cisco HyperFlex management with a single management platform.
* It makes it possible to scale across data center and remote locations without additional complexity.
* It automates the generation and forwarding of technical support files to the Cisco Technical Assistance Center to accelerate the troubleshooting process.
* Full programmability and automation capabilities are available through a REST API interface.
* A streamlined upgrade process is available for standalone Cisco UCS servers.

The Intersight API is a programmatic interface to the Management Information Model that is similar to Cisco ACI and Cisco UCS Manager. Just like Cisco ACI and Cisco UCS Manager MIMs, the Cisco Intersight MIM is composed of managed objects. Managed objects or REST API resources are uniquely identified by URI (uniform resource identifier) or, as seen earlier in this chapter, distinguished name (DN).

Example of managed objects include Cisco UCS servers; Cisco UCS fabric interconnects; Cisco HyperFlex nodes and clusters; server, network, and storage policies; alarms; statistics; users; and roles. Cisco Intersight managed objects are represented using a class hierarchy specified in the OpenAPI specification. All the API resources are descendants of the **mo.Mo** class.

| **Property Name** | **Description** |
| --- | --- |
| **Moid** | A unique identifier of the managed object instance. |
| **ObjectType** | The fully qualified class name of the managed object. |
| **AccountMoid** | The Intersight account ID for the managed object. |
| **CreateTime** | The time when the managed object was created. |
| **ModTime** | The time when the managed object was last modified. **ModTime** is automatically updated whenever at least one property of the managed object is modified. |
| **Owners** | An array of owners, which represents effective ownership of the object. |
| **Tags** | An array of tags that allow the addition of key/value metadata to managed objects. |
| **Ancestors** | An array containing the MO references of the ancestors in the object containment hierarchy. |
| **Parent** | The direct ancestor of the managed object in the containment hierarchy. |

Every managed object has a unique **Moid** identifier assigned when the resource is created. The **Moid** is used to uniquely distinguish a Cisco Intersight resource from all other resources. The **Moid** is a 12-byte string set when a resource is created.

Each managed object can be addressed using a unique uniform resource identifier (URI) that includes the **Moid**. The URI can be used in any HTTPS request to address the managed object. A generic Cisco Intersight URI is of the following form:

**https://intersight.com/**path**[?**query**]**

The URI of a managed object includes the following:

* **https:** The HTTPS protocol
* **intersight.com:** The Cisco Intersight hostname
* path: The path, organized in hierarchical form
* query: An optional query after the question mark and typically used to limit the output of the response to only specific parameters

For example, the URI of an object with **Moid 48601f85ae74b80001aee589** could be:

[**https://intersight.com/api/v1/asset/DeviceRegistrations/48601f85ae74b80001aee589**](https://intersight.com/api/v1/asset/DeviceRegistrations/48601f85ae74b80001aee589)

Managed objects may include object relationships, which are dynamic links to REST resources. Cisco Intersight uses Hypermedia as the Engine of Application State (HATEOAS) conventions to represent object relationships. Object relationships can be links to self or links to other managed objects, which, taken as a whole, form a graph of objects. By using relationships as a first-class attribute in the object model, many classes of graphs can be represented, including trees and cyclic or bipartite graphs.

Intersight provides a rich query language based on the OData standard. The query language is represented using URL query parameters for **GET** results. Several types of data are supported with the Intersight queries, including string, number, duration, data and time, and time of day.

When a client sends an API request, the Intersight web service must identify and authenticate the client. The Intersight web service supports two authentication methods:

* API keys
* Session cookies
* An Intersight API key is composed of a keyId and a keySecret. The API client uses the API key to cryptographically sign each HTTP request sent to the Intersight web service. The **“signature”** parameter is a base 64–encoded digital signature of the message HTTP headers and message content. API keys are generated in the Settings > API section of the Intersight web interface. As a best practice, it is recommended to generate separate API keys for each client application that needs access to the API.
* Cookies are used primarily by the Intersight GUI client running in a browser. When accessing the Intersight web service, end users must first authenticate to [https://sso.cisco.com](https://sso.cisco.com/). When authentication is successful, sso.cisco.com sends a signed SAML assertion to the Intersight web service, and Intersight generates a session cookie with a limited time span validity. The client must send the session cookie in each API request.

<https://intersight.com/apidocs/downloads/>