White Paper

The Cisco Application Policy Infrastructure Controller

Introduction: What Is the Cisco Application Policy Infrastructure Controller?

The Cisco® Application Policy Infrastructure Controller (referred to as the APIC) is a distributed system implemented as a cluster of controllers. The APIC provides a single point of control, a central API, a central repository of global data, and a repository of policy data for the Cisco Application Centric Infrastructure (ACI). The Cisco ACI is conceptualized as a distributed overlay system with external endpoint connections controlled and grouped via policies. Physically, ACI is a high-speed, multipath leaf and spine (bipartite graph) fabric.

The APIC is a unified point of policy-driven configuration. The primary function of the APIC is to provide policy authority and policy resolution mechanisms for the Cisco ACI and ACI-attached devices. Automation is provided as a direct result of policy resolution and of rendering its effects onto the Cisco ACI fabric.

The APIC communicates in the infrastructure VLAN (in-band) with the Cisco ACI spine and leaf nodes to distribute policies to the points of attachment (Cisco leaf) and provide a number of key administrative functions to the Cisco ACI. The APIC is not directly involved in data plane forwarding, so a complete failure or disconnection of all APIC elements in a cluster will not result in any loss of existing datacenter functionality.

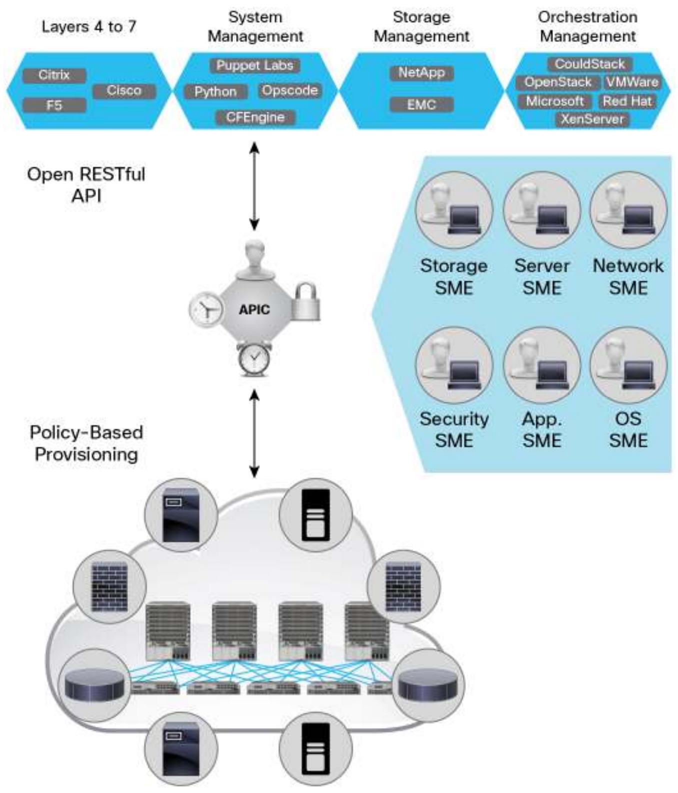
In general, policies are distributed to nodes as needed upon endpoint attachment or by an administrative static binding. You can, however, specify “resolutional immediacy,” which regulates when policies are delivered into Cisco nodes. “Prefetch” or early resolution is one of the modes. The most scalable mode is the “just-in-time mode,” in which policies are delivered to nodes just in time upon detection of the attachment. Attachment detection is based on analysis of various triggers available to the APIC.

A central APIC concept is to express application networking needs as an extension of application-level metadata through a set of policies and requirements that are automatically applied to the network infrastructure. The APIC policy model allows specification of network policy in an application- and workload-centric way. It describes sets of endpoints with identical network and semantic behaviors as endpoint groups. Policies are specified per interaction among such endpoint groups.

Key features of the Cisco APIC:

- Application centric network policies
- Data model-based declarative provisioning
- Application, topology monitoring, and troubleshooting
- Third-party integration (Layer 4 through 7 services, storage, computing, WAN)
- Image management (spine and leaf)
- ACI inventory and configuration
- Implementation on a distributed framework across a cluster of appliances

Figure 1. Cisco APIC Policy Model



Scalable and Flexible

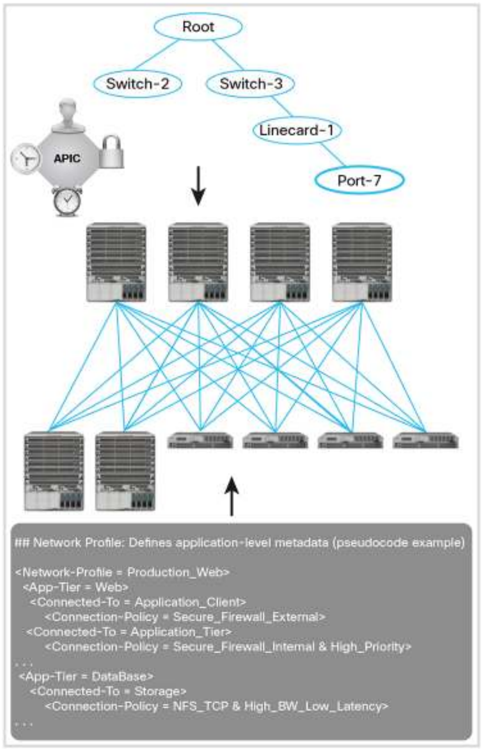
A single APIC cluster supports over 1 million ACI endpoints, more than 200,000 ports, and more than 64,000 tenants and provides centralized access to ACI information through a number of interfaces, including an object-oriented RESTful API with XML and JSON bindings, a modernized user-extensible command-line interface (CLI), and a GUI. All methods are based on a model of equal access to internal information. Furthermore, APIC clusters are fully extensible to computing and storage management.

APIC Is Not Another NMS

The APIC is a network policy control system. However, it is not a network element management system and should not be mistaken for a manager of managers. Instead, it is designed to extend the manageability innovations of the ACI Fabric OS platform by augmenting it with a policy-based configuration model and providing end-to-end ACI global visibility. Cisco has cultivated a broad ecosystem of partners to work with the APIC and provide important functions, including:

- Fault and event management
- System and configuration tools
- Performance management
- Automation tools
- Orchestration frameworks
- Statistical collection and analysis
- Hypervisor, storage, and computing management
- Layer 4 through 7 services integration
- Cloud management
- IP address management (IPAM)

Figure 2. Policy-driven Fabric



```
## Network Profile: Defines application-level metadata (pseudocode example)

<Network-Profile = Production_Web>
 <App-Tier = Web>
  <Connected-To = Application_Client>
     <Connection-Policy = Secure_Firewall_External>
  <Connected-To = Application_Tier>
     <Connection-Policy = Secure_Firewall_Internal & High_Priority>
. . .
 <App-Tier = DataBase>
  <Connected-To = Storage>
     <Connection-Policy = NFS_TCP & High_BW_Low_Latency>
. . .
```

Virtual ACI Context: Securing Tenants

A tenant is a logical container or a folder for application policies. It can represent an actual tenant, an organization, or a domain or can just be used for the convenience of organizing information. A normal tenant represents a unit of isolation from a policy perspective, but it does not represent a private network. A special tenant named "common" has sharable policies that can be used by all tenants. A context is a representation of a private Layer 3 namespace or Layer 3 network. It is a unit of isolation in our ACI framework. A tenant can rely on several contexts. Contexts can be declared within a tenant (contained by the tenant) or can be in the "common" tenant. This approach enables us to provide both multiple private Layer 3 networks per tenant and shared Layer 3 networks used by multiple tenants. This way, we do not dictate a specific rigidly constrained tenancy model. The endpoint policy specifies a common ACI behavior for all endpoints defined within a given virtual ACI context.

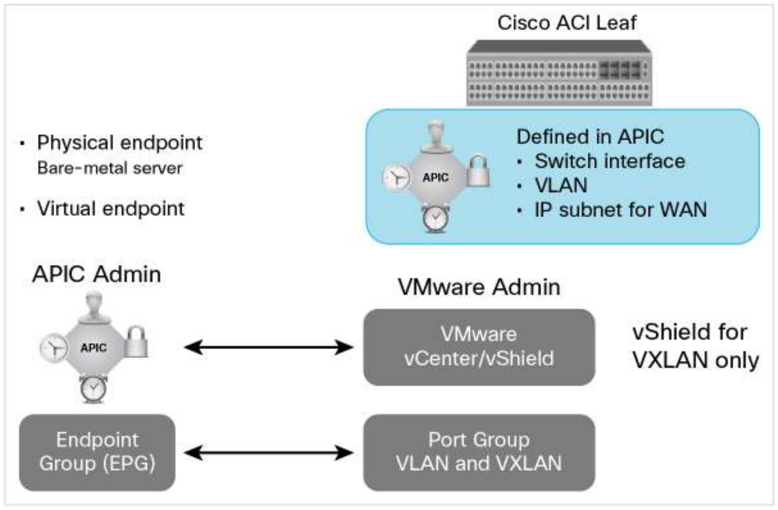
Endpoints and Policy Control

The Cisco ACI is conceptualized as a distributed overlay system with external endpoint connections controlled and grouped via policies. The central concept here is to group endpoints (EPs) with identical semantics into endpoint groups (EPGs) and then write policies that regulate how such groups can interact with each other. These policies provide rules for connectivity, visibility (access control), and isolation of the endpoints. The APIC's primary responsibility is distributing, tracking, and updating such policies to corresponding Cisco ACI nodes as client endpoint connectivity to the Cisco ACI is established, changed, or removed. Endpoint policy control consists of two logically coupled elements:

Policy repository: This is a collection of policies and rules applied to existing or hypothetical (either deleted or not yet created) endpoints.

Endpoint registry: This is a registry of endpoints currently known to the Cisco ACI. External client endpoints are any external computing, network, or storage element that is not a component of the Cisco ACI. Client endpoints are directly connected to the Cisco leaf, or indirectly via a fabric extender (FEX) or intermediate switches (such as blade switches in the blade systems) or a virtual switch. For example, a computing endpoint can be a single virtual machine's VM network interface card (vmNIC) or virtual NIC (vNIC), a physical server connection via a physical NIC, or a virtualized vNIC (SR-IOV, Palo, etc.)

Figure 3. Endpoint Identification



Endpoints and their ACI attachment location may or may not be known when the EPG is initially defined on the APIC. Therefore, endpoints either can be prespecified into corresponding EPGs (statically at any time) or can be added dynamically as they are attached to the ACI. Endpoints are tracked by a special endpoint registry mechanism of the policy repository. This tracking serves two purposes: It gives the APIC visibility into the attached endpoints and dictates policy consumption and distribution on the Cisco leaf switches.

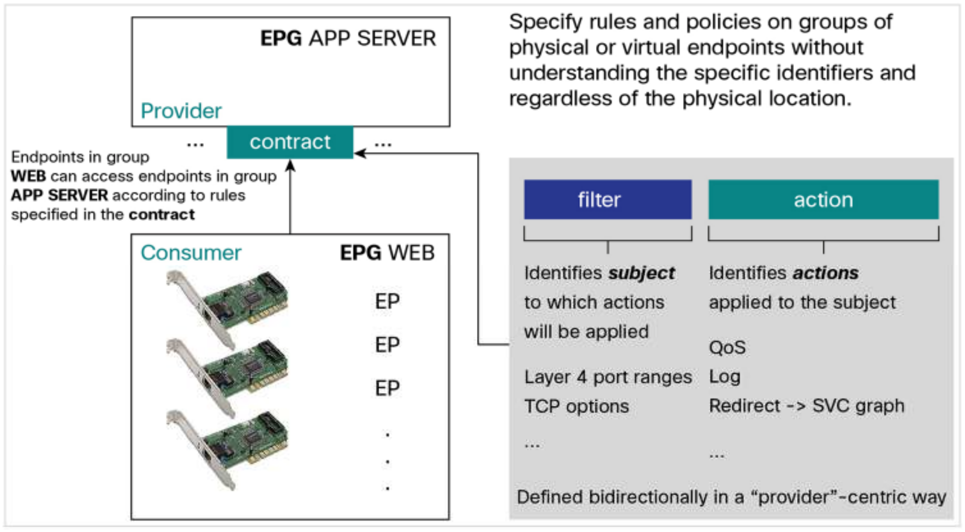
Endpoint Groups: Building Blocks of Policy and Automation

An endpoint group (EPG) is a collection of endpoints with identical ACI-level behaviors and requirements. The practical implication of the EPG in relation to the VM management layer is that it can be thought of as a vSphere port group, or as a network as defined in the OpenStack Neutron API. Groups have an “application-tier-like” semantic at the application metadata level, where a web server connects to the network as a member of the EPG “web-servers” and all rules pertaining to web servers in a given application are applied. However, in more traditional environments, it is possible to think of an EPG as a collection of physical ports, VLANs, a subnet, or some other unit of isolation or networked workload containment. This group behavior represents the most basic building block of network configuration automation on the APIC.

Endpoint Group Contracts

The APIC policy model defines EPG “contracts” between EPGs that control the various parameters between application tiers such as connectivity, visibility, service insertion, packet quality of service (QoS), etc. A contract allows a user to specify rules and policies for groups of physical or virtual endpoints without understanding any specific identifiers or even who is providing the service or who is consuming it, regardless of the physical location of the devices (Figure 4). This abstraction of specificity makes the Cisco policy model truly object oriented and highly flexible. Each contract consists of a filtering construct, which is a list of one or more classifiers (IP address, TCP/UDP ports, etc.), and an action construct that dictates how the matching traffic is handled (allow, apply QoS, log traffic, etc.).

Figure 4. Endpoint Group Contracts


EPG APP SERVER
Provider
contract
...
...
Endpoints in group WEB can access endpoints in group APP SERVER according to rules specified in the contract
Consumer
EPG WEB
EP
EP
EP
Specify rules and policies on groups of physical or virtual endpoints without understanding the specific identifiers and regardless of the physical location.
filter
action
Identifies subject to which actions will be applied
Layer 4 port ranges
TCP options
...
Identifies actions applied to the subject
QoS
Log
Redirect -> SVC graph
...
Defined bidirectionally in a “provider”-centric way

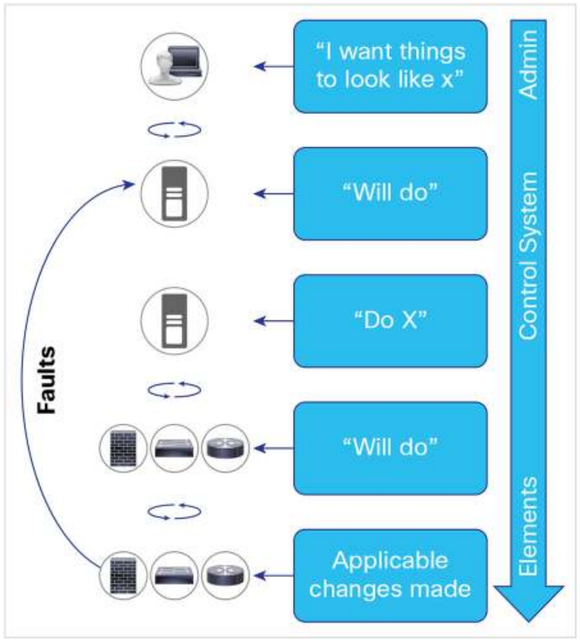
Another implication of the contract filters is their effect on the distribution of policy and endpoint visibility information. For a given endpoint attaching to a given leaf, only the information about related endpoints is communicated, along with corresponding policies.

A Model-Based Controller Implemented with Promise Theory

In model-driven architectures, the software maintains a complete representation of the administrative and operational state of the system (the model).

Subject-matter experts (SMEs) do not configure the physical system resources directly but rather define logical configuration policy abstractions of policy state (hardware independent) that control different aspects of the system behavior.

Figure 5. Promise Theory Model



The APIC uses a variant of promise theory with full formal separation of the logical and concrete models, in which no configuration is carried out against concrete entities (Figure 5). Concrete entities are configured implicitly as a side effect of the logical model implementation. Concrete entities can be physical, but they don't have to be (such as VMs and VLANs).

Logical configurations are rendered into concrete configurations by applying the policies in relation to the available physical resources, taking into account their state and capabilities.

Concrete configurations are deployed to the managed endpoints in an asynchronous, nonblocking fashion.

This involves profiles and policies in which logical entities are expressed as policy requirements. The management model applies uniformly to the ACI, services, and system behaviors.

The enforcement of polices is achieved via a hive of policy elements (PEs), a concept inspired by sensor networks. PEs enforce the desired state expressed in declared policies on each node of the ACI, in accordance with promise theory. This is distinctly different from traditional top-down management, in which every resource is configured directly.

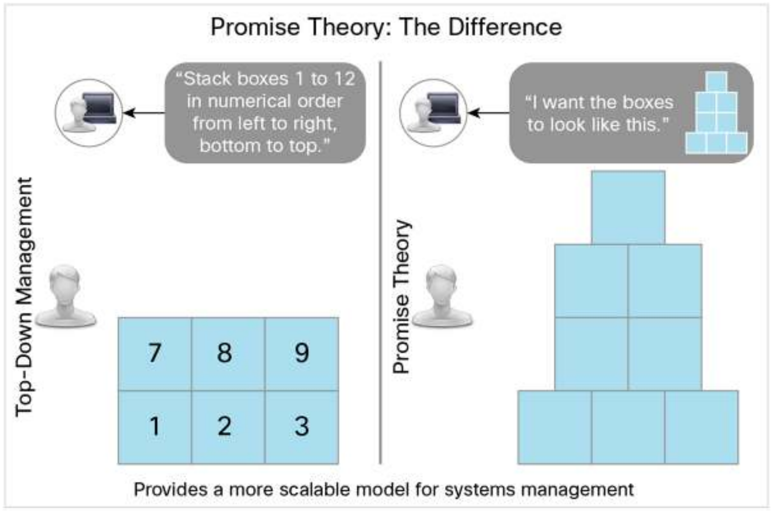
Given an excitation trigger, it is the responsibility of a PE, directly or indirectly via a proxy, to trigger policy resolution. Once policy is resolved, its artifacts are rendered to the concrete model, and the backend (ACI Fabric OS processes) reacts to such changes.

Why Promise Theory?

Promise theory has many advantages over traditional elemental management (Figure 6):

- Provides declarative automation.
- State convergence: The PE continuously conducts checks to ensure that the configuration complies with the desired state of the infrastructure.
- Tight policy loop: The PE corrects drift of the operational state from the desired state.
- Provides continual real-time auditing of the operational state as compared to the defined state.
- Removes the side effects of change.
- Model remains intact at scale.
- Scales linearly with the object-driven model.
- Objects are responsible for the requested configuration.
- No assumptions are made concerning the current object state. Relies on trust relationships and end-device ownership of configuration change.

Figure 6. Promise Theory vs. Top-Down Management

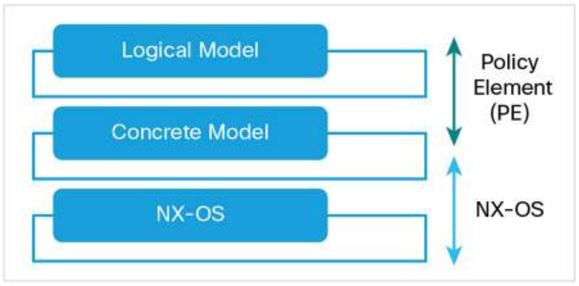

Promise Theory: The Difference
"Stack boxes 1 to 12 in numerical order from left to right, bottom to top."
"I want the boxes to look like this."
Top-Down Management
Promise Theory
7
8
9
1
2
3
Provides a more scalable model for systems management

Cisco ACI Operating System (ACI Fabric OS)

Cisco has taken the traditional Cisco Nexus® OS (NX-OS) developed for the datacenter and pared it down to the essential features required for a modern datacenter deploying the Cisco ACI. There have also been deeper structural changes so that the ACI Fabric OS can easily render policy from the APIC into the physical infrastructure. A Data Management Engine (DME) in ACI Fabric OS provides the framework that serves read and write requests from a shared lockless datastore. The datastore is object oriented, with each object stored as chunks of data.

A chunk is owned by one ACI Fabric OS process, and only the owner of this process can write to the data chunk. However, any ACI Fabric OS process can read any of the data simultaneously through the CLI, Simple Network Management Protocol (SNMP) or an API call. A local policy element (PE) enables the APIC to implement the policy model directly in ACI Fabric OS (Figure 7).

Figure 7. ACI Fabric OS

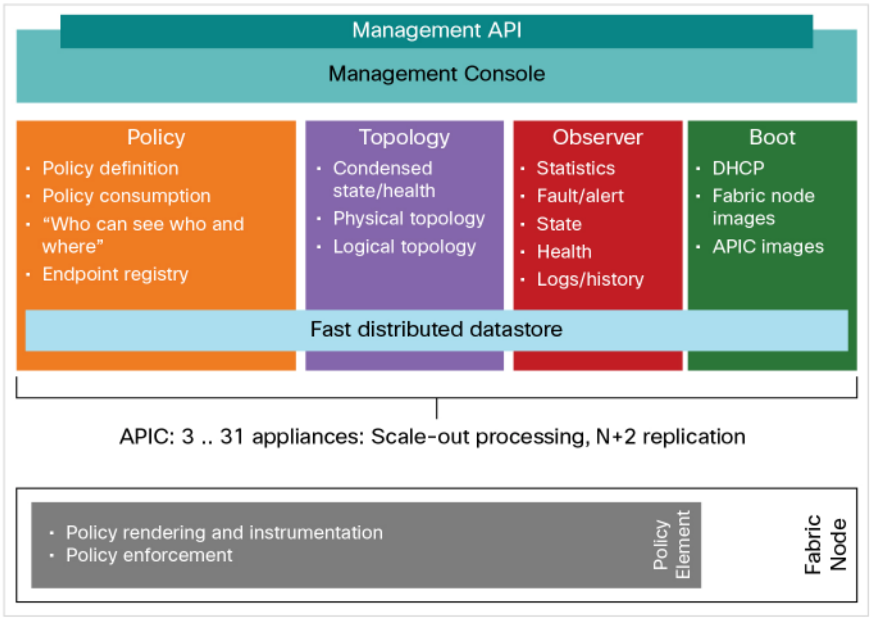


Architecture: Components and Functions of the APIC

The APIC consists of a set of basic control functions, including:

- Policy Manager (policy repository)
- Topology Manager
- Observer
- Boot Manager
- Appliance Director (cluster controller)
- VMM Manager
- Event Manager
- Appliance Element

Figure 8. APIC Component Architecture



Policy Manager

The Policy Manager is a distributed policy repository responsible for the definition and deployment of the policy-based configuration of the Cisco ACI. This is a collection of policies and rules applied to existing or hypothetical (not yet created) endpoints. The endpoint registry is a subset of the Policy Manager that tracks endpoints connecting to the Cisco ACI and their assignment to endpoint groups as defined by the policy repository.

Topology Manager

The Topology Manager maintains up-to-date ACI topology and inventory information. Topology information is reported to the APIC by the leaf and spine switches. The physical topology is based on the information discovered by the Link Layer Discovery Protocol (LLDP) and the routing topology of the fabric as reported by protocols (modified intermediate system to intermediate system [IS-IS]) running within the fabric infrastructure space.

A global view of time-accurate topology information is available in the Topology Manager, including:

- Physical topology (Layer 1; physical links and nodes)
- Logical path topology (reflection of Layer 2 + Layer 3)

Topology information, along with associated aggregated operational state, is asynchronously updated in the Topology Manager upon detection of topology changes, and is available for queries via the APIC API, CLI, and UI.

A subfunction of Topology Manager performs inventory management for the APIC and maintains a complete inventory of the entire Cisco ACI. The APIC inventory management subfunction provides full identification, including model and serial number, as well as user-defined asset tags (for ease of correlation with asset and inventory management systems) for all ports, line cards, switches, chassis, etc.

Inventory is automatically pushed by the DME-based policy element/agent embedded in the switches as soon as new inventory items are discovered or removed or transition in state in the local repository of the ACI node.

Observer

The Observer is the monitoring subsystem of the APIC, and it serves as a data repository of the Cisco ACI's operational state, health, and performance, including:

- Hardware and software state and health of ACI components
- Operational state of protocols
- Performance data (statistics)
- Outstanding and past fault and alarm data
- Record of events

Monitoring data is available for queries via the APIC API, CLI, and UI.

Boot Director

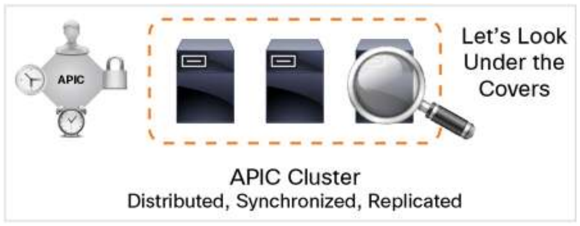
The Boot Director controls the booting and firmware updates of the Cisco spine and leaf and the APIC controller elements. It also functions as the address allocation authority for the infrastructure network, which allows the APIC and the spine and leaf nodes to communicate. The following process describes bringing up the APIC and cluster discovery.

- Each APIC in the Cisco ACI uses an internal private IP address to communicate with the ACI nodes and other APICs in the cluster. APICs discover the IP address of other APICs in the cluster using an LLDP-based discovery process.
- APICs maintain an appliance vector (AV), which provides a mapping from an APIC ID to an APIC IP address and a universally unique identifier (UUID) of the APIC. Initially, each APIC starts with an AV filled with its local IP address, and all other APIC slots are marked unknown.
- Upon switch reboot, the PE on the leaf gets its AV from the APIC. The switch then advertises this AV to all of its neighbors and reports any discrepancies between its local AV and neighbors' AVs to all the APICs in its local AV.
- Using this process, APICs learn about the other APICs in the ACI via switches. After validating these newly discovered APICs in the cluster, APICs update their local AV and program the switches with the new AV. Switches then start advertising this new AV. This process continues until all the switches have the identical AV and all APICs know the IP address of all the other APICs.

Appliance Director

The Appliance Director is responsible for formation and control of the APIC appliance cluster. The APIC is delivered as a software appliance that is installed on customer-owned server hardware ("bare-metal").A minimum of three controllers are initially installed for control of the scale-out ACI (Figure 9). The ultimate size of the APIC cluster is directly proportionate to the ACI size and is driven by the transaction rate requirements. Any controller in the cluster is able to service any user for any operation, and a controller can be seamlessly added to or removed from the APIC cluster. It is important to understand that unlike an OpenFlow controller, none of the APIC controllers are ever in the data path.

Figure 9. Appliance Director


APIC
Let's Look Under the Covers
APIC Cluster
Distributed, Synchronized, Replicated

VMM Manager

The VMM Manager acts as an agent between the policy repository and a hypervisor and is responsible for interacting with hypervisor management systems such as VMware's vCenter and cloud software platforms such as OpenStack and CloudStack. VMM Manager inventories all of the hypervisor elements (pNICs, vNICs, VM names, etc.) and pushes policy into the hypervisor, creating port groups, etc. It also listens to hypervisor events such as VM mobility.

Event Manager

The Event Manager is a repository for all the events and faults initiated from the APIC or the fabric nodes.

Appliance Element

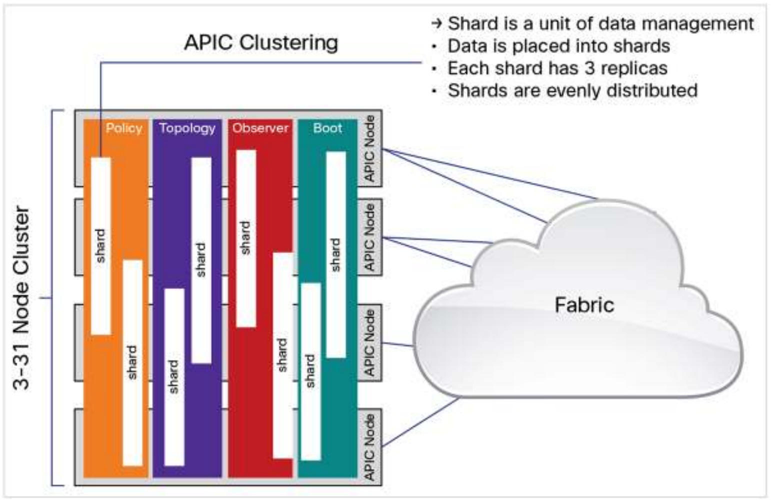
The Appliance Element is a monitor for the local appliance. It manages the inventory and state of the local APIC appliance.

Architecture: Data Management

The APIC cluster uses a technology from large databases called **sharding** (Figure 10). This technology provides scalability and reliability to the data sets generated and processed by the **Distributed Policy Repository**, the **endpoint registry**, the **Observer**, and the **Topology Manager**. The data for these APIC functions is partitioned into logically bounded subsets called **shards** (analogous to database shards). A shard is a unit of data management, and all of the above data sets are placed into shards:

- Each shard has three replicas
- Shards are evenly distributed
- They enable horizontal (scale-out) scaling
- They simplify the scope of replications

Figure 10. Sharding


APIC Clustering
→ Shard is a unit of data management
• Data is placed into shards
• Each shard has 3 replicas
• Shards are evenly distributed
3-31 Node Cluster
Policy
Topology
Observer
Boot
shard
APIC Node
Fabric

One or more shards are located on each APIC appliance and processed by a controller instance located on that appliance. The shard data assignments are based on a predetermined hash function, and a static shard layout determines the assignment of shards to appliances.

Each replica in the shard has a use preference, and writes occur on the replica that is elected leader. Other replicas are followers and do not allow writes. In the case of a split-brain condition, automatic reconciliation is performed based on timestamps. Each APIC controller has all APIC functions; however, processing is evenly distributed throughout the APIC controller cluster.

User Interface: Graphical User Interface (GUI)

The GUI is an HTML5-based web UI that works with most modern web browsers. The GUI provides seamless access to both the APIC and the individual nodes.

User Interface: Command-Line Interface (CLI)

A full stylistic and semantic (where it applies) compatibility with Cisco NX-OS CLI is provided. The CLI for the entire Cisco ACI is accessed through the APIC and supports a transactional mode. There is also the ability to access specific Cisco ACI nodes with a read-only CLI for troubleshooting. An integrated Python-based scripting interface is supported that allows user-defined commands to attach to the command tree as if they were native platform-supported commands. Additionally, the APIC provides a library for any custom scripts.

User Interface: RESTful API

The APIC supports a comprehensive RESTful API over HTTP(S) with XML and JSON encoding bindings. Both class-level and tree-oriented data access is provided by the API.

Representational state transfer (REST) is a style of software architecture for distributed systems such as the World Wide Web. REST has emerged over the past few years as a predominant web services design model. REST has increasingly displaced other design models such as SOAP and Web Services Description Language (WSDL) due to its simpler style. The uniform interface that any REST interface must provide is considered fundamental to the design of any REST service, and thus the interface has these guiding principles:

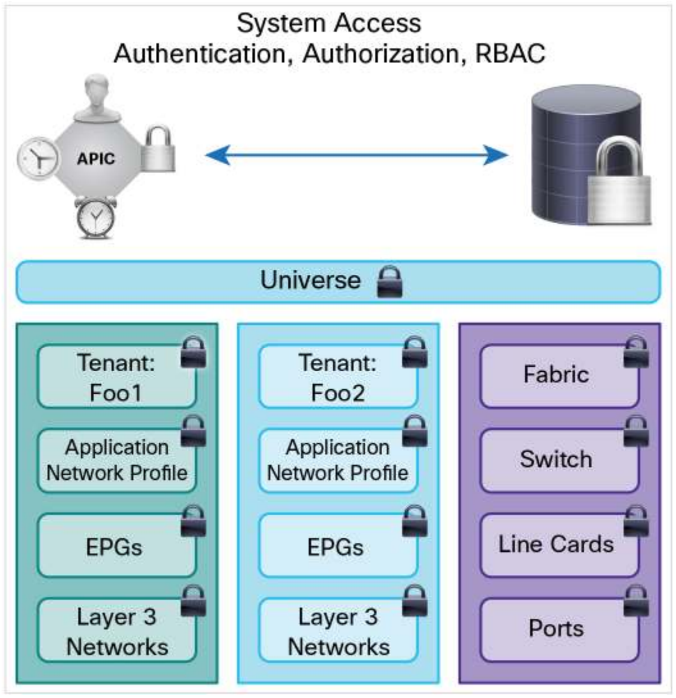
- Identification of resources: Individual resources are identified in requests, for example, using URIs in web-based REST systems. The resources themselves are conceptually separate from the representations that are returned to the client.
- Manipulation of resources through these representations: When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource on the server, provided it has permission to do so.
- Self-descriptive messages: Each message includes enough information to describe how to process the message. Responses also explicitly indicate their cache ability.
- An important concept in REST is the existence of resources (sources of specific information), each of which is referenced with a global identifier (such as a URI in HTTP). In order to manipulate these resources, components of the network (user agents and origin servers) communicate via a standardized interface (such as HTTP) and exchange representations of these resources (the actual documents conveying the information).

Any number of connectors (clients, servers, caches, tunnels, etc.) can mediate the request, but each does so without “seeing past” its own request (referred to as **layering**, another constraint of REST and a common principle in many other parts of information and networking architecture). Thus, an application can interact with a resource by knowing two things: the identifier of the resource and the action required - it does not need to know whether there are caches, proxies, gateways, firewalls, tunnels, or anything else between it and the server actually holding the information. The application does, however, need to understand the format of the information (representation) returned, which is typically an HTML, XML, or JSON document of some kind, although it may be an image, plain text, or any other content.

System Access: Authentication, Authorization, and RBAC

The APIC supports both local and external authentication and authorization (TACACS+, RADIUS, Lightweight Directory Access Protocol [LDAP]) as well as role-based administrative control (RBAC) to control read and write access for **all** managed objects and to enforce ACI administrative and per-tenant administrative separation (Figure 11). The APIC also supports domain-based access control, which enforces where (under which subtrees) a user has access permissions.

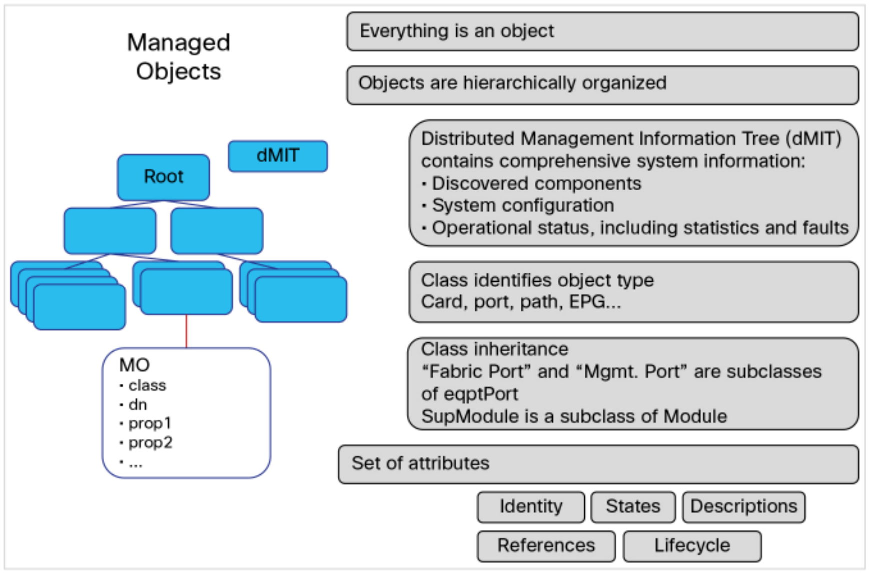
Figure 11. Authentication, Authorization, and RBAC



API Requests and URL/URI

The Cisco ACI Fabric OS Data Management Engine (DME) hierarchical object model approach is a very good fit for a RESTful interface, as URLs and URIs map directly into distinguished names identifying managed objects (MO) on the tree, and any data on the distributed Management Information Tree (dMIT) can be described as a self-contained structured text document encoded in XML or JSON (Figure 12). This structure is similar to that of Common Management Information Protocol (CMIP) and other X.500 invariants. The DME was designed to allow the control of managed resources by presenting their manageable characteristics as object properties. The objects have parent-child relationships that are identified using distinguished names and properties, which are read and modified by a set of create, read, update, and delete (CRUD) operations. The object model features a full unified description of entities and no artificial separation of configuration, state, or runtime data (Figure 13).

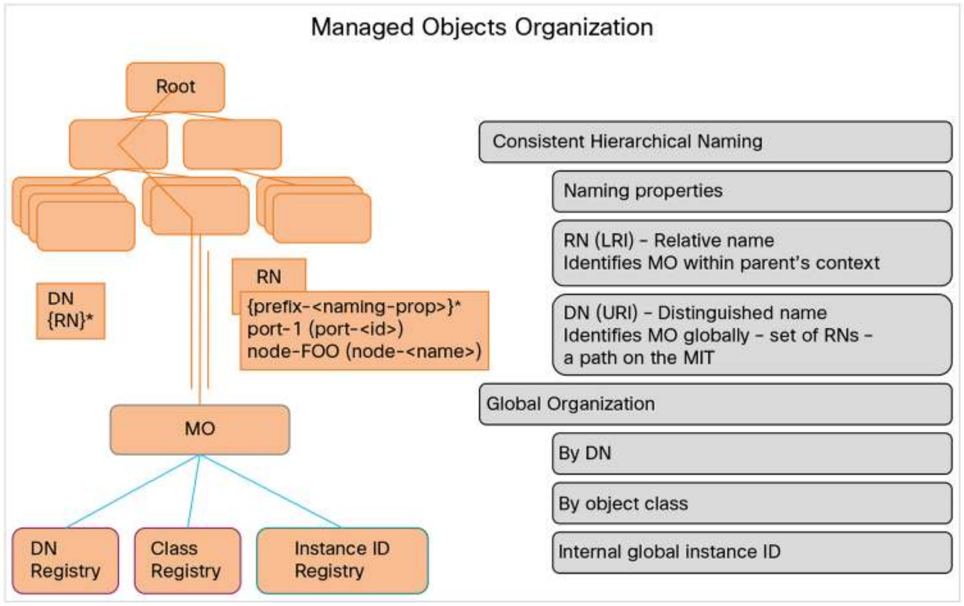
Figure 12. Unified Data Model


Managed Objects
Everything is an object
Objects are hierarchically organized
dMIT
Root
Distributed Management Information Tree (dMIT) contains comprehensive system information:
• Discovered components
• System configuration
• Operational status, including statistics and faults
Class identifies object type
Card, port, path, EPG...
MO
• class
• dn
• prop1
• prop2
• ...
Class inheritance
"Fabric Port" and "Mgmt. Port" are subclasses of eqptPort
SupModule is a subclass of Module
Set of attributes
Identity
States
Descriptions
References
Lifecycle

Accessing the dMIT:

- Queries return a set of objects or subtrees.
- There is an option to return an entire or partial subtree for each object in resolution scope.
- RBAC privileges define what types of objects can be accessed.
- Domain identifies what subtrees are accessed.

Figure 13. Organization of Managed Objects


Managed Objects Organization
Root
Consistent Hierarchical Naming
Naming properties
RN (LRI) – Relative name
Identifies MO within parent's context
DN
{RN}*
RN
{prefix-<naming-prop>}*
port-1 (port-<id>)
node-FOO (node-<name>)
DN (URI) – Distinguished name
Identifies MO globally – set of RNs – a path on the MIT
Global Organization
MO
By DN
By object class
Internal global instance ID
DN Registry
Class Registry
Instance ID Registry

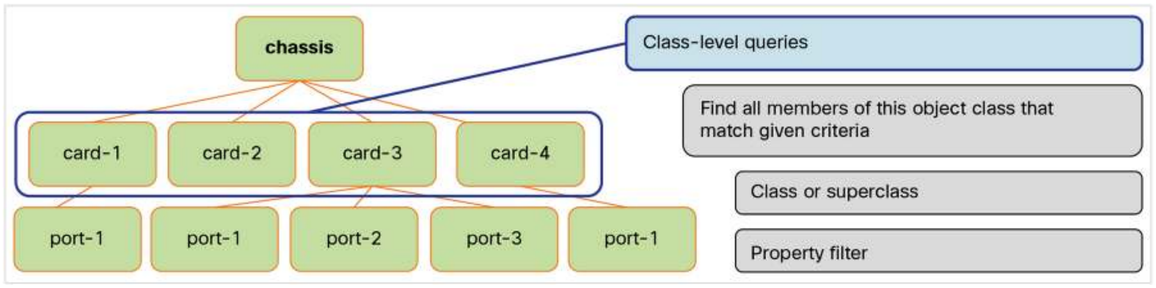
The REST API uses standard HTTP commands for retrieval and manipulation of APIC data. The URL format used in the API is represented as follows:

<system>/api/[mo|class]/[dn|class][:method].[xml|json] {options}

- system: System identifier, an IP address or DNS-resolvable host name.
- mo | class: Indicates whether this is an mo/tree (MIT) or class-level query.
- class: Managed object class (as specified in the information model) of the objects queried. Class name is represented as <pkgName><ManagedObjectClassName>.
- dn: Distinguished name (unique hierarchical name of the object on the MIT) of the object queried.
- method: Optional indication of what method is being invoked on the object; applies only to HTTP POST requests.
- xml|json: Encoding format.
- Options: Query options, filter, arguments.

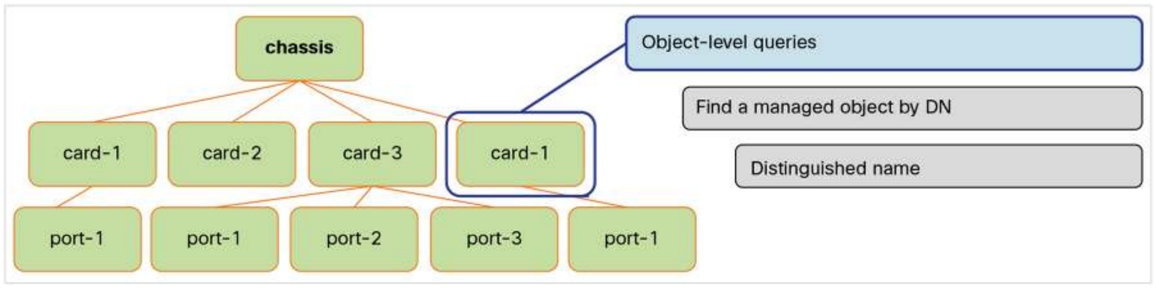
For example, ***ifc-1.foo.com:7580/api/node-20/mo/sys/ch/lcslot-1/lc.xml*** globally identifies linecard 1 of system 20 (Figure 14).

Figure 14. Class-Level Queries



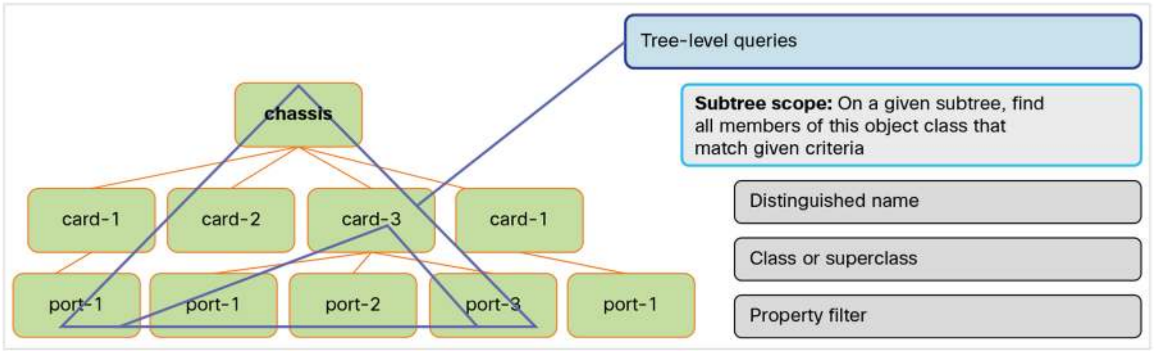
The API also supports using a specific class-level URL format, providing access to all of a certain class of objects from the dMIT (Figure 15).

Figure 15. Object-Level Queries



The API further supports using a tree- or subtree-level URL format, providing access to a specific tree or subtree of objects from the dMIT (Figure 16).

Figure 16. Tree-Level Queries



Conclusion

The Cisco Application Policy Infrastructure Controller (APIC) is a modern, highly scalable distributed control system that manages the Cisco ACI switch elements and provides policy-driven network provisioning that is implicitly automated. Additionally, the APIC provides the technology to implement a new paradigm of application policy and a robust automation platform for the network and all the attached elements. The APIC is designed to do all of this while remaining out of the data path, thus allowing extremely high performance of the Cisco ACI.

Printed in USA C11-730021-00 11/13