ACI-SE M04

Application Policy Infrastructure Controller (APIC)

Application Policy Infrastructure Controller

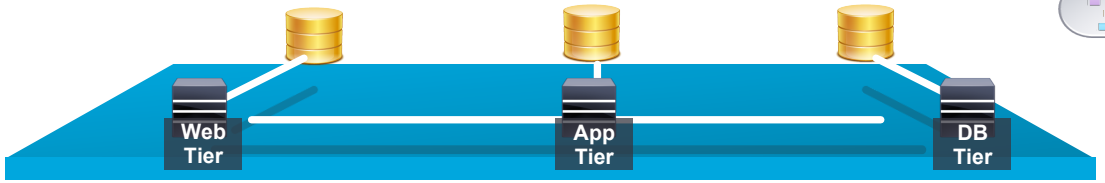
APIC Operations

ACI Goal: Common Policy and Operations Framework

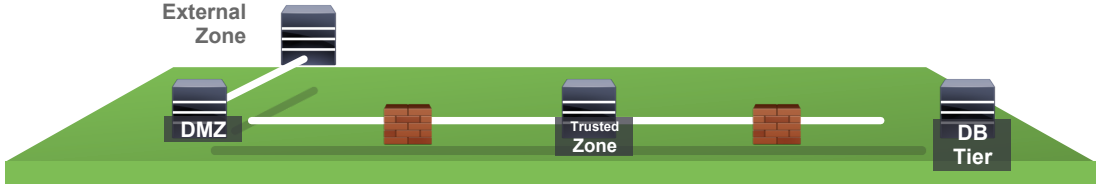
Application Admin

Security Admin

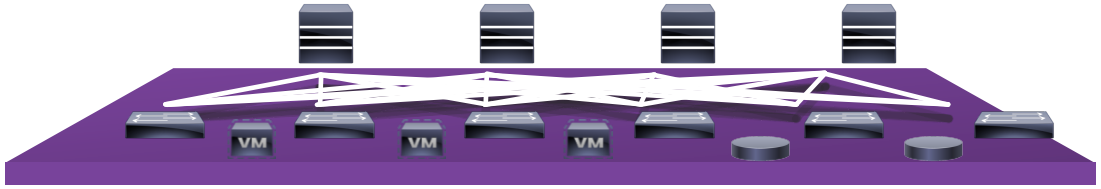
Network Admin

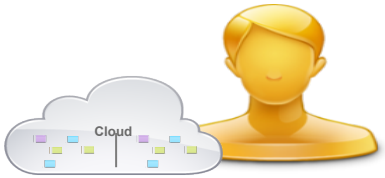


APPLICATION



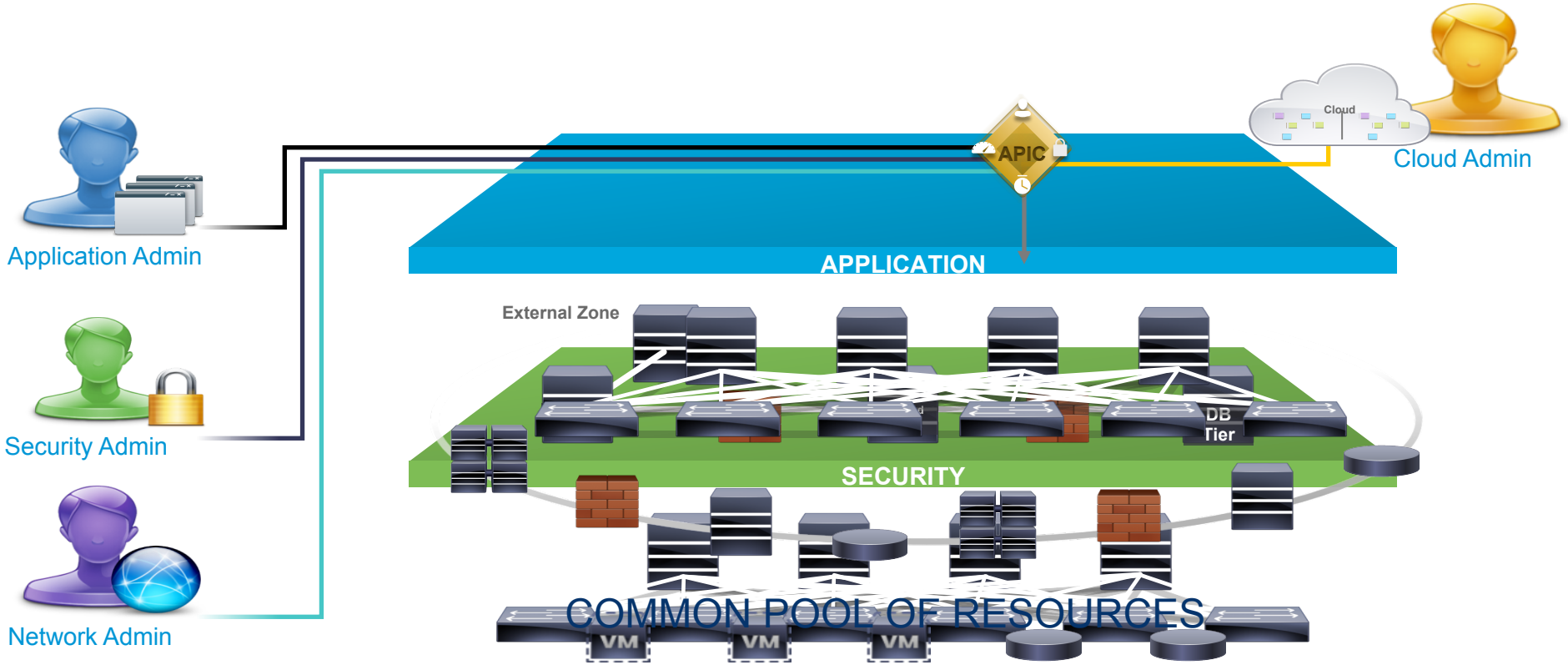
SECURITY



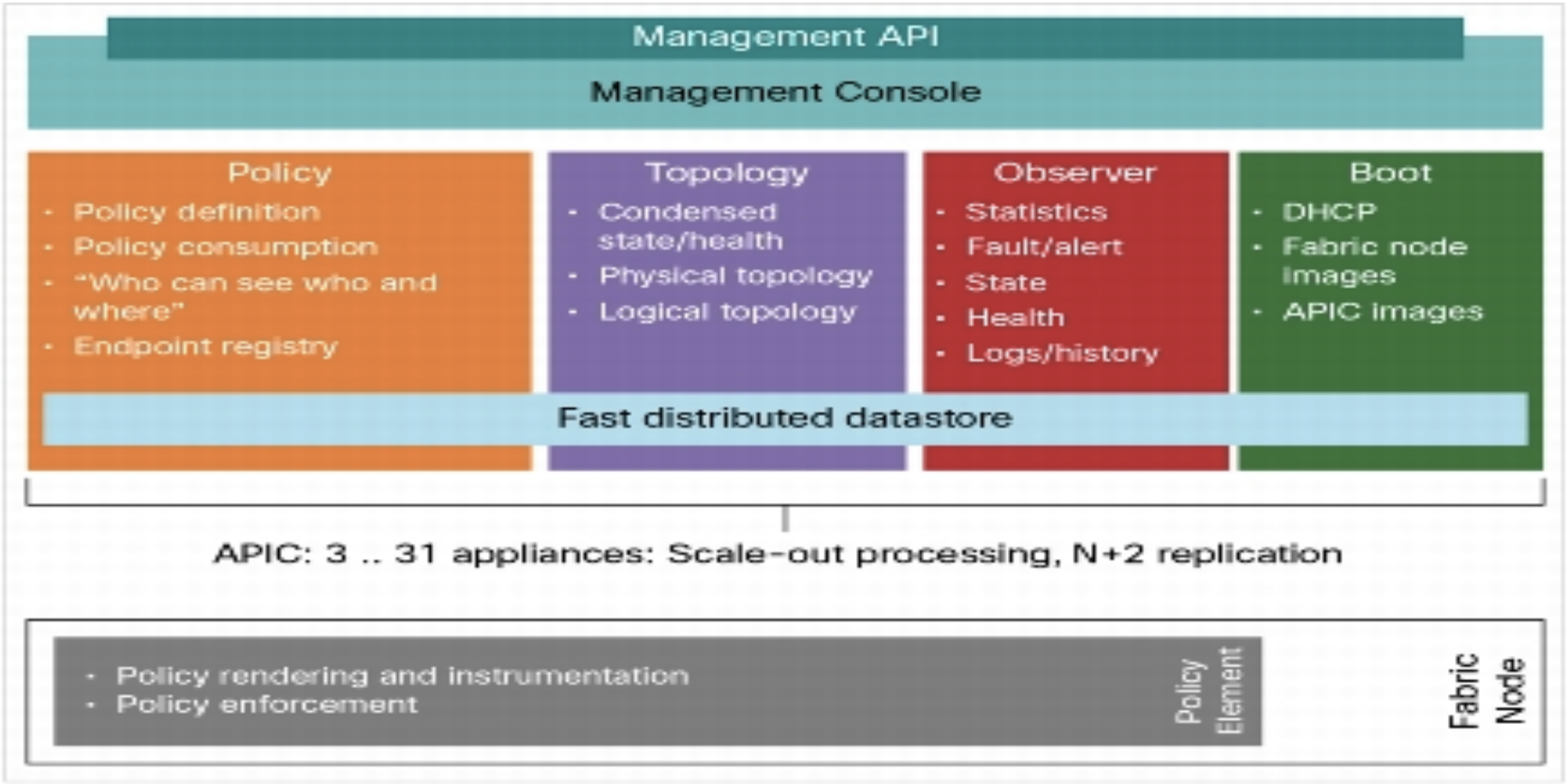


Cloud Admin

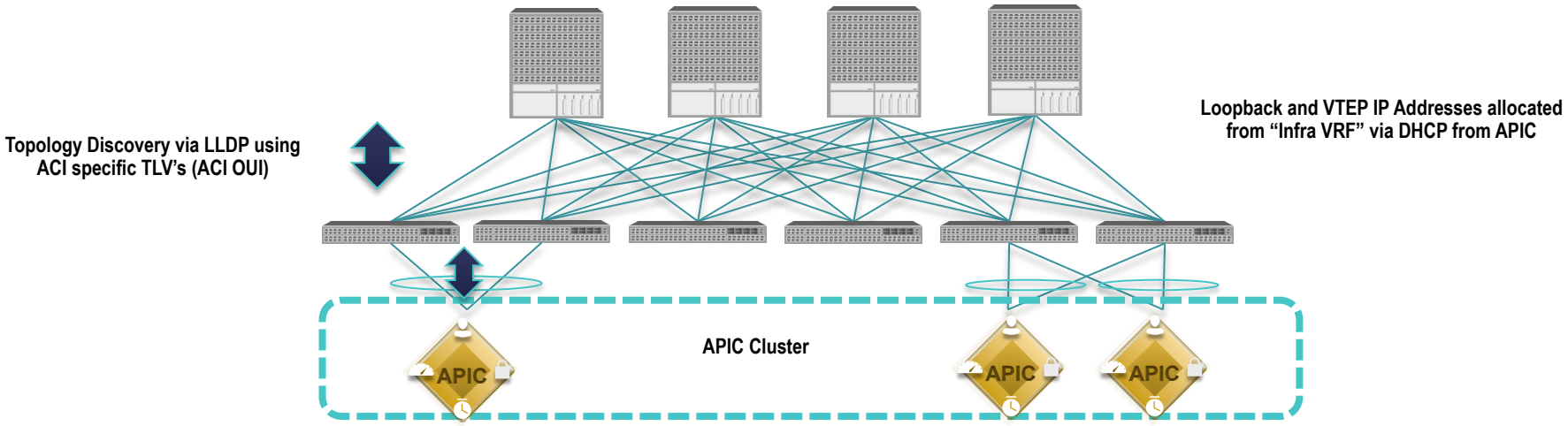
ACI Goal: Common Policy and Operations Framework



APIC

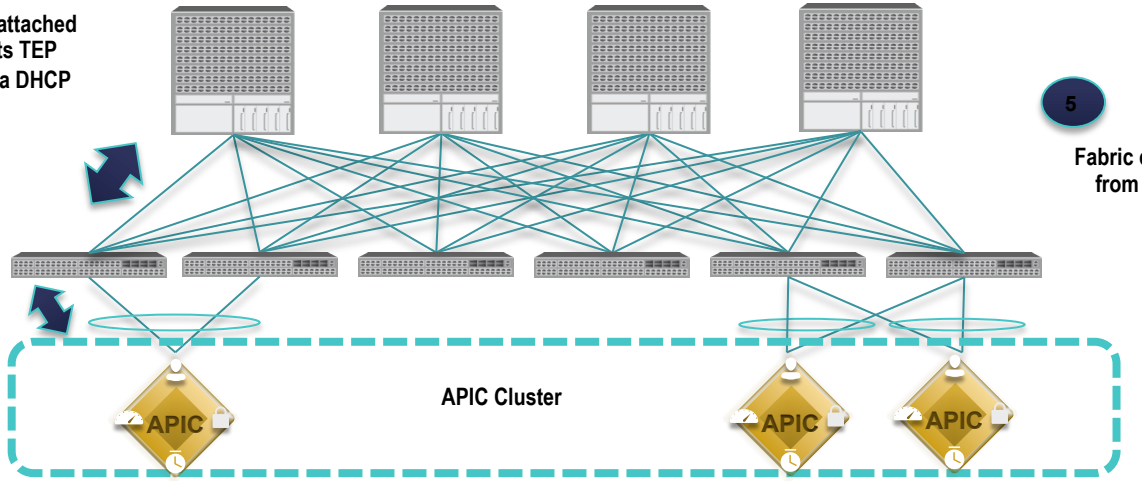

Management API
Management Console
Policy
Policy definition
Policy consumption
"Who can see who and where"
Endpoint registry
Topology
Condensed state/health
Physical topology
Logical topology
Observer
Statistics
Fault/alert
State
Health
Logs/history
Boot
DHCP
Fabric node images
APIC images
Fast distributed datastore
APIC: 3 .. 31 appliances: Scale-out processing, N+2 replication
Policy rendering and instrumentation
Policy enforcement
Policy Element
Fabric Node

Fabric Initialization & Maintenance



- ACI Fabric supports discovery, boot, inventory and systems maintenance processes via the APIC
  - Fabric Discovery and Addressing
  - Image Management
  - Topology validation through wiring diagram and systems checks

Fabric Initialization & Maintenance



6 Fabric will self assemble starting from multiple APIC sources

3 Spine switch discovers attached Leaf via LLDP, requests TEP address and boot file via DHCP

5 Fabric can be discovered and initialized from multiple sources concurrently

2 Leaf switch discovers attached APIC via LLDP, requests TEP address and boot file via DHCP

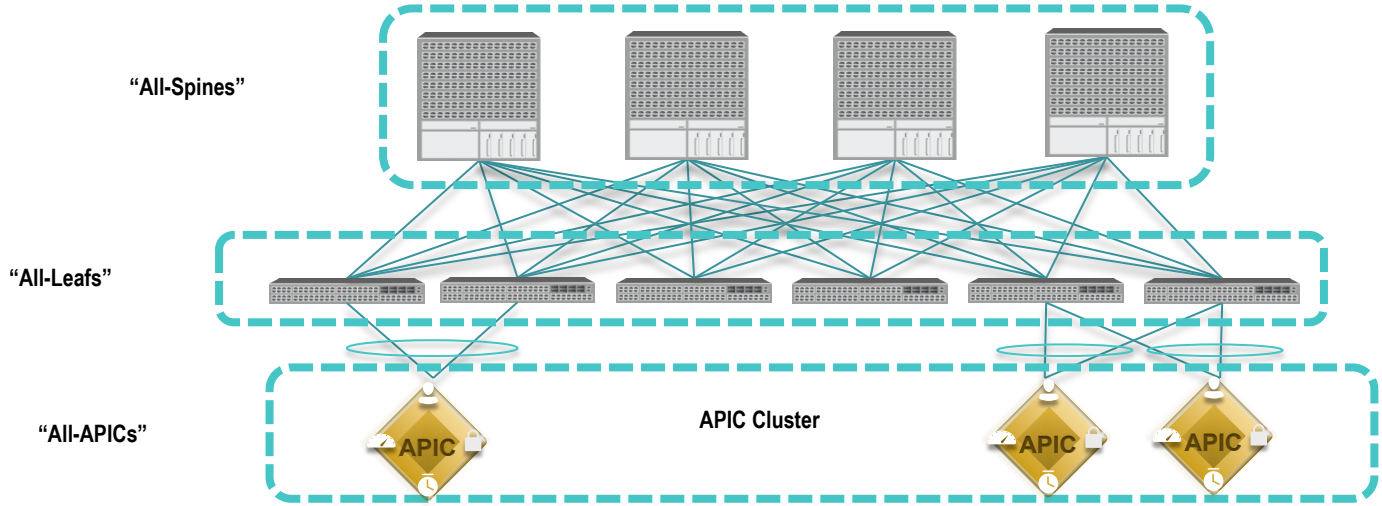
7 APIC Cluster will form when members discovery each other via Appliance Vector (AV)

1 APIC bootstrap configuration

1) APIC Cluster Configuration
2) Fabric Name
3) TEP Address space (Infra-VRF)
4) …

4 All nodes in the same APIC cluster should contain same bootstrap information if they are intended to form a cluster

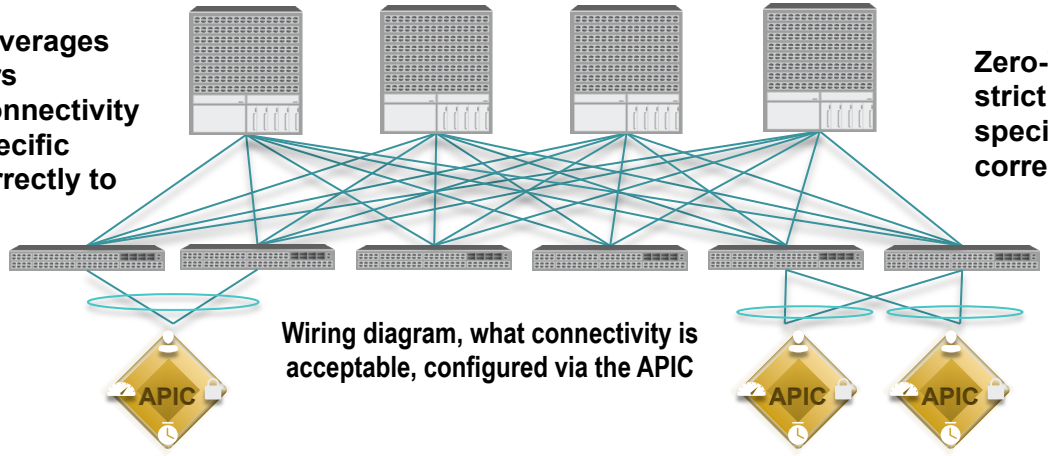
Fabric Initialization & Maintenance



- ACI Fabric leverages the same Global Catalogue methodology as UCS, the supported HW/SW matrix, image versioning, …
- APIC and switch node image management controlled via APIC policies
  - Policies control which images should be on which groupings of devices, when the images should be upgraded/downgraded
  - Also control the upgrade process, automatic, manual step by step, …

Fabric Initialization & Maintenance

Minimal Touch install leverages specific boot parameters combined with loose connectivity policies to ensure a specific device is connected correctly to the fabric

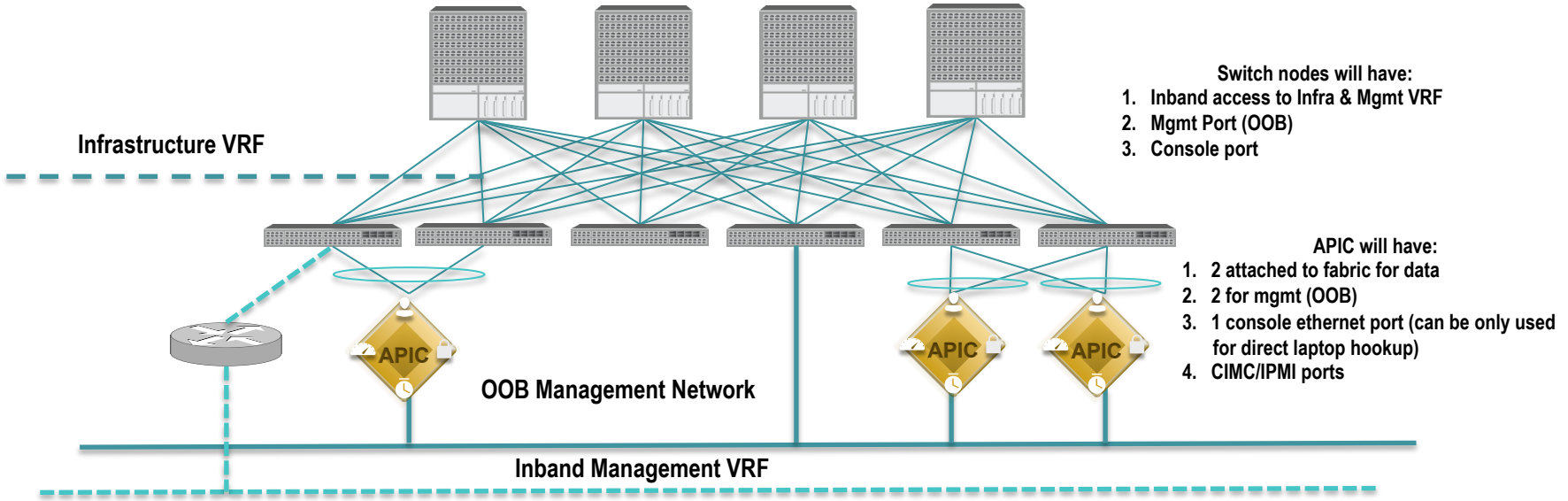


Wiring diagram, what connectivity is acceptable, configured via the APIC

Zero-Touch install leverages strict wiring policy to ensure a specific device is connected correctly to the fabric

- All nodes in the fabric are fundamentally ‘stateless’ in that they require no local state to be configured and added to the fabric
- To provide more explicit control at FCS it is assumed one of two mechanisms will be used
  - Loose wiring plan rules + local device config (switch-role, fabric-ID, node-ID)
  - Strict wiring plan rules with zero touch node config

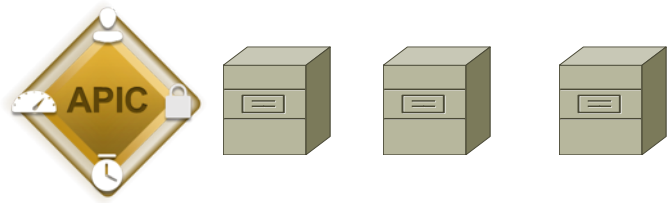
Management Networks



- Infra VRF – Used for inband APIC to switch node communication, non routable outside the fabric
- Inband Management Network – ‘tenant’ VRF created for inband access to switch nodes
- OOB Management Network – APIC and switch node dedicated mgmt ports

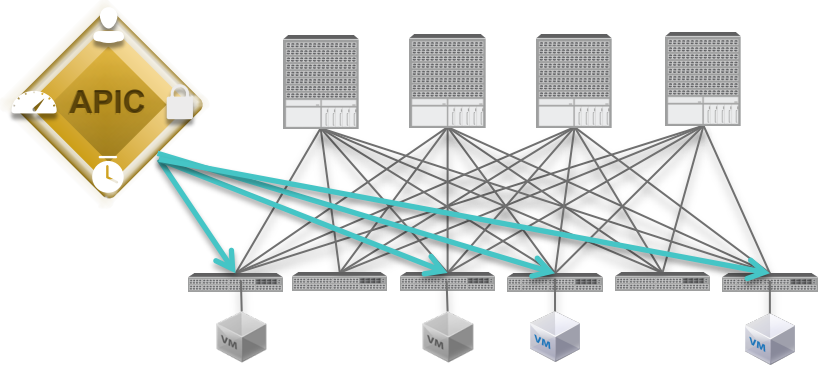
What is APIC?

- APIC is the policy controller
- It's not the control plane
- It's not in the data path



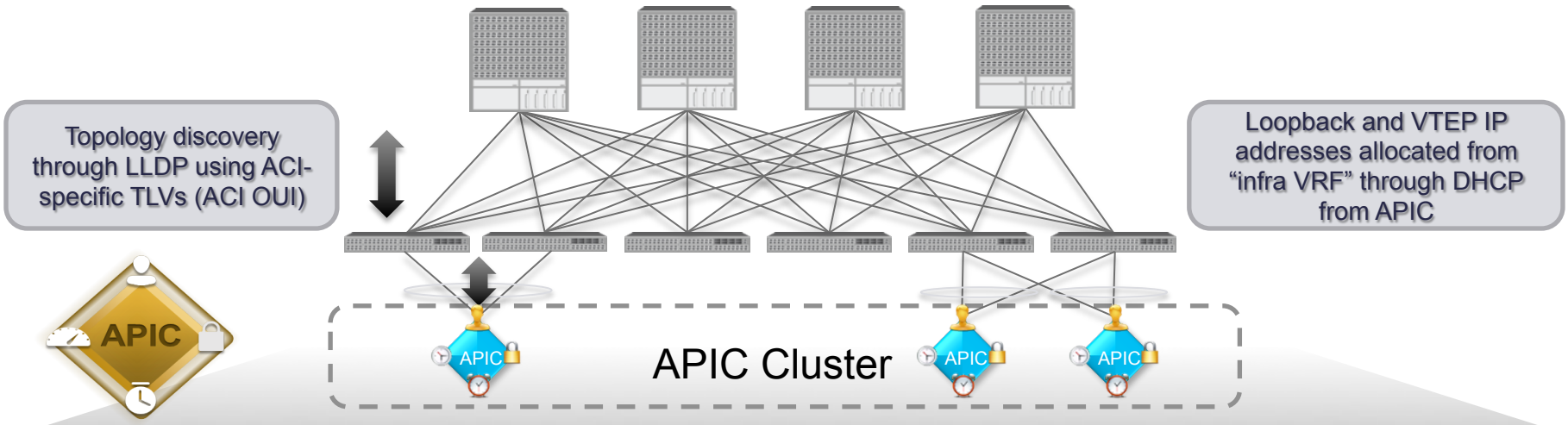
- It's a highly redundant cluster of 3+ Servers

Hypervisor Integration with ACI
Policy Resolution Immediacy



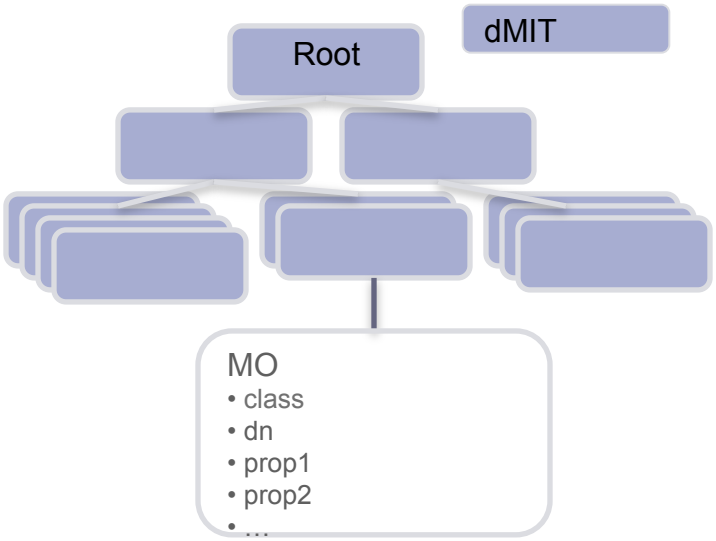
- Policies are pushed to Leaf nodes based on Resolution Immediacy defined upon association of EPG to VMM Domain
  - **Immediate**: All policies (VLAN / NVGRE / VXLAN bindings, Contracts, Filters) pushed to leaf node upon Hypervisor pNIC attachment. LLDP or OpFlex used to resolve Hypervisor to Leaf node attachment.
  - **Lazy:** Policies only pushed to leaf node upon pNIC attachment AND vNIC association with port-group (EPG)
- Policy programming in Leaf node hardware based on Instrumentation Immediacy
  - **Immediate:** Policies programmed in Policy CAM once received by APIC as defined by Resolution Immediacy Policy
  - Lazy: Polices programmed in hardware Policy CAM only when reachability is learnt through data path

APIC controller is attached in-band



- ACI Fabric supports discovery, boot, inventory, and systems maintenance processes through the APIC
  - Fabric discovery and addressing
  - Image management
  - Topology validation through wiring diagram and systems checks

Object Tree



Full unified description of entities.

No artificial separation of configuration, state, runtime data.

Everything is an object

Objects are hierarchically organized

Distributed Managed 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…

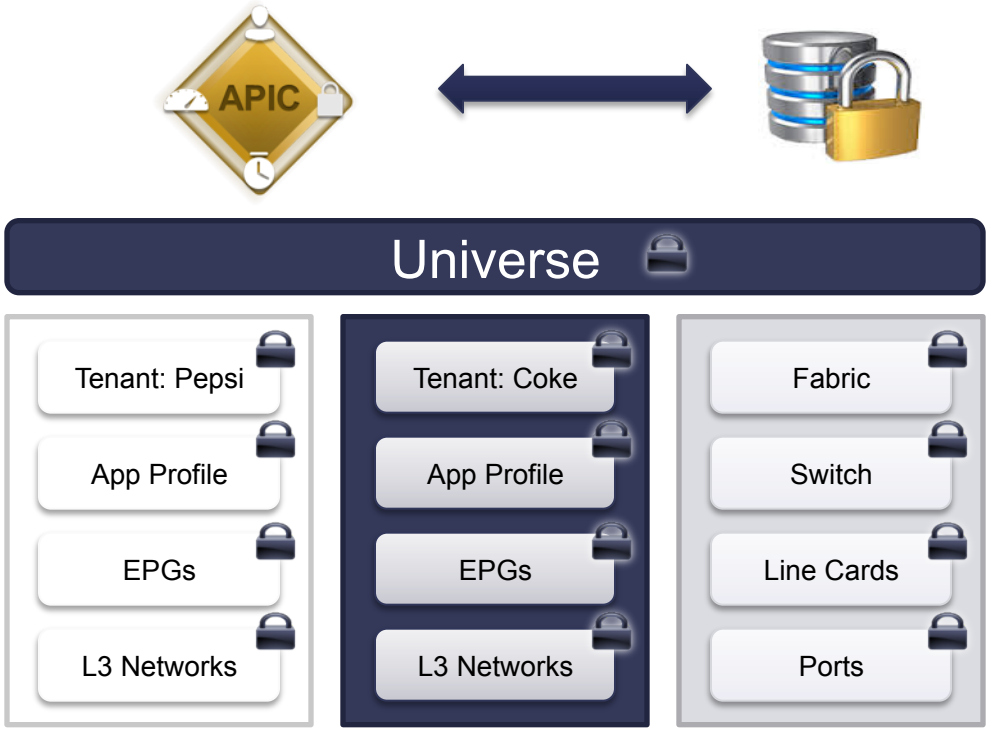
Class Inheritance
Access port is a subclass of port.
A leaf node is a subclass of fabric node.

Set of attributes

identity | states | descriptions | references | lifecycle

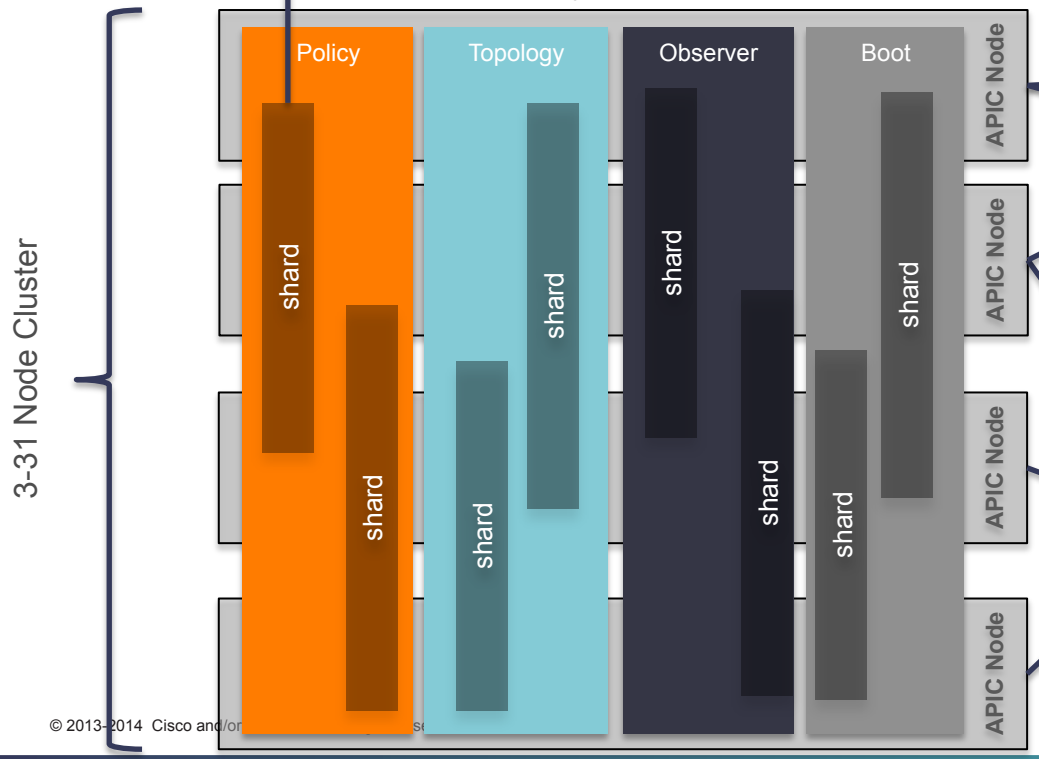
Multi-tenancy

- Local & External AAA (TACACS +, RADIUS, LDAP) Authentication & Authorization
- RBAC to control READ and WRITE for ALL Managed Objects
- RBAC to enforce Fabric Admin and per-Tenant Admin separation



APIC Clustering

→ Shard is a unit of data mgmt
- Data is placed into shards
- Each shard has 3 replicas
- Shards are evenly distributed



Allows horizontal (scale-out) scaling.
Simplifies replications scope.

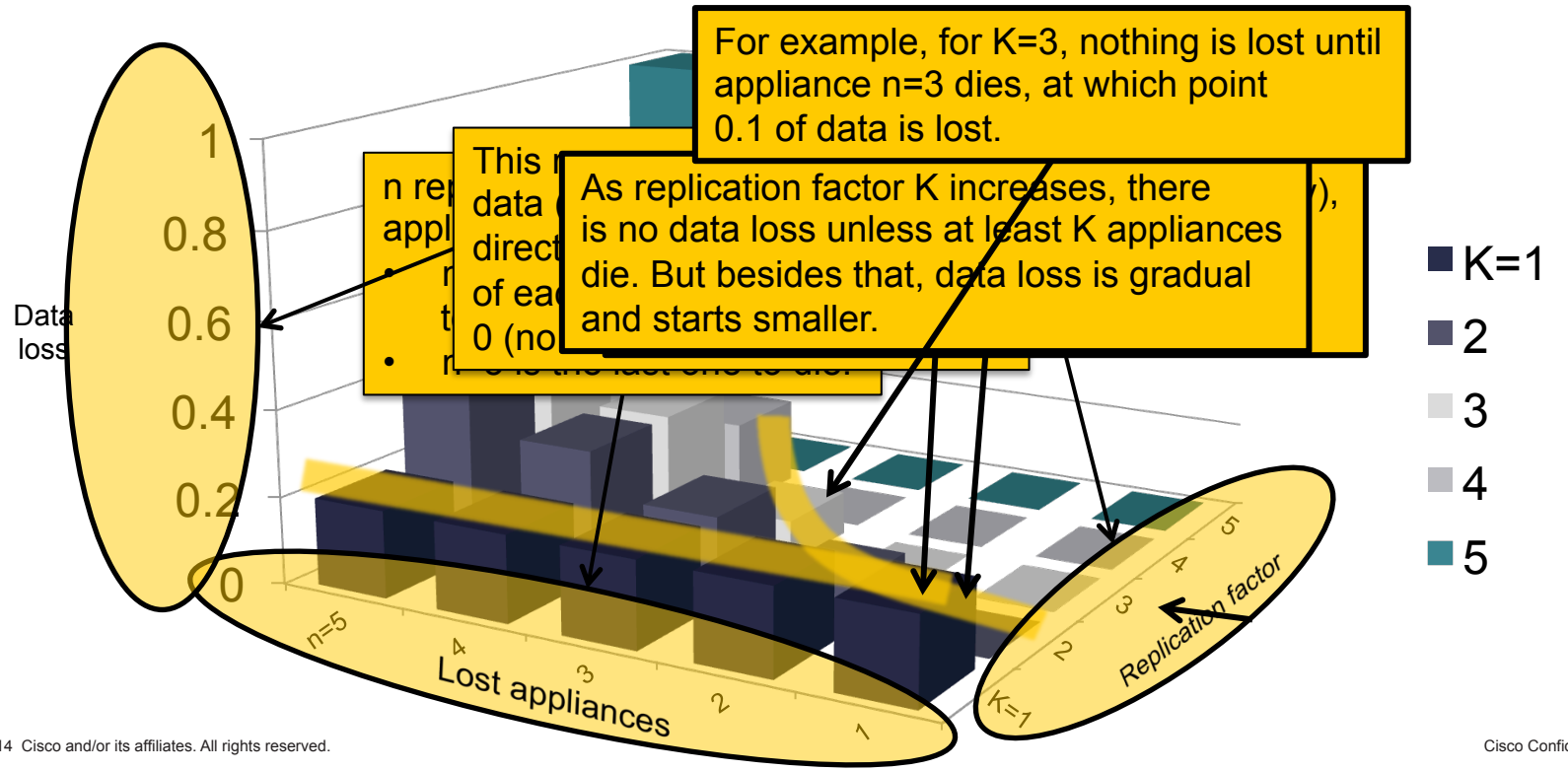
- Shard data assignments are based on pre-determined hash function.
- Static shard layout determines the assignment of shards to appliances

ACI Fabric

- Each replica in the shard has use preference (1..3)
- Writes happen to the highest preference reachable
- Each APIC Node has all APIC functions, however, processing is evenly distributed
- In case of split-brain, automatic reconciliation is performed

Effect of Replication on Reliability

Proportion of data that is lost when n^{th} appliance dies, out of a total of **5** appliances and a variable replication factor K.



Effect of Sharding on Reliability

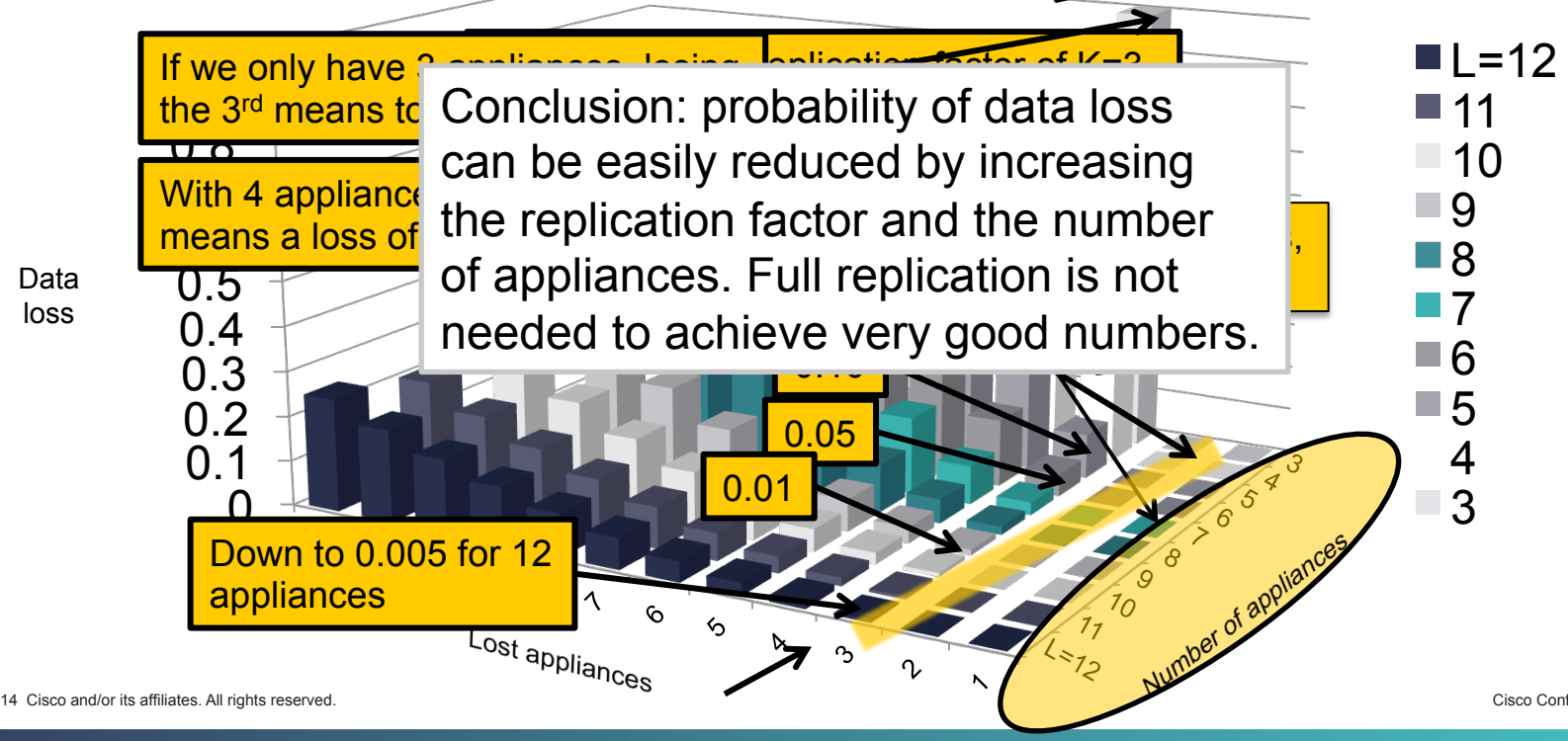
Proportion d … es, for a variable number of appliances **L=3..12** and fixed replica…

Increasing the number of appliances does significantly (and rapidly) improve resilience.

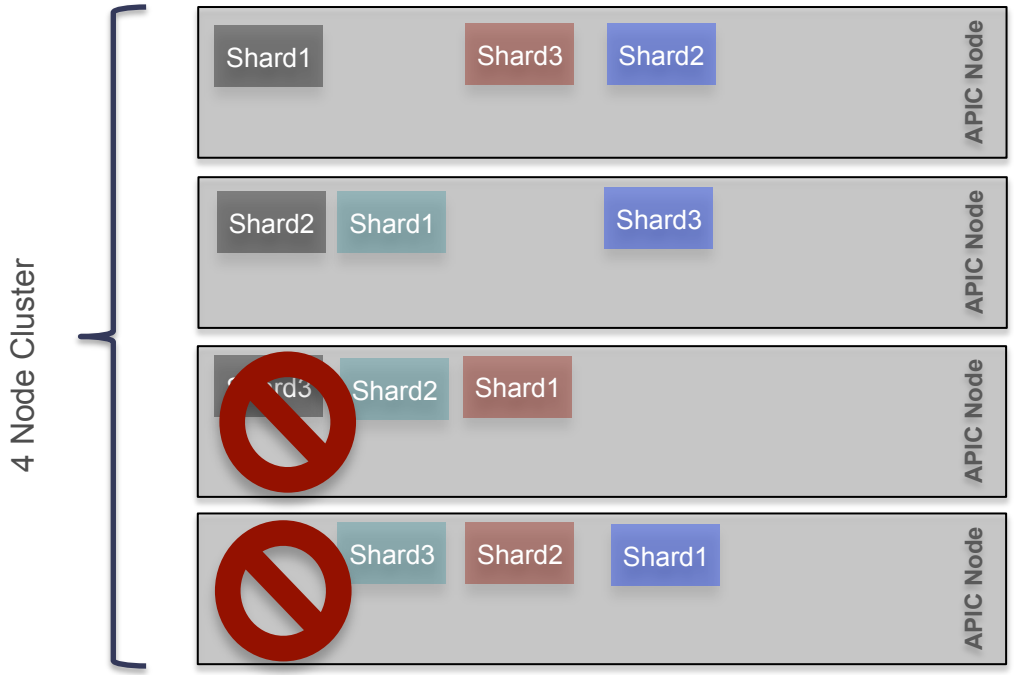
If we only have … the 3rd means to …

With 4 appliance… means a loss of …

Conclusion: probability of data loss can be easily reduced by increasing the replication factor and the number of appliances. Full replication is not needed to achieve very good numbers.



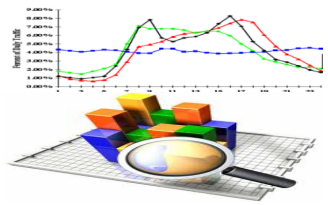
APIC Clustering Sharding and Reliability

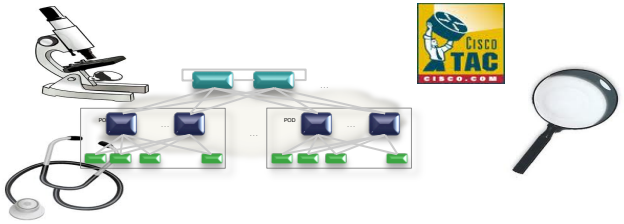


- Each shard has 3 replicas
- Shards are evenly distributed
- No data is lost for as long as just 1 or 2 APIC appliances die.
- Data loss begins only when we lose our third appliance
- With 4 appliances, losing the 3rd means a loss of 0.25

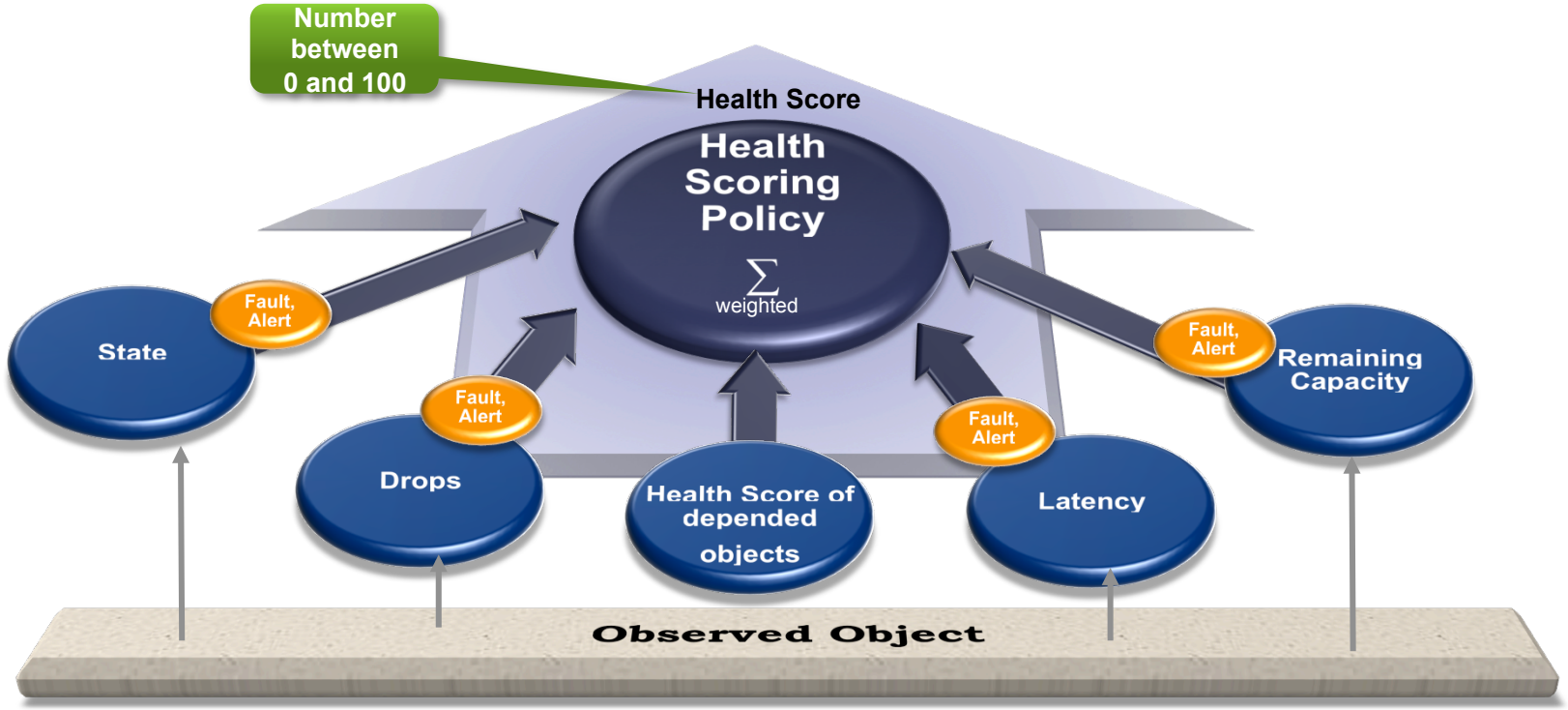
The Observer: Functionalities

Packets
Unicasts
Drops



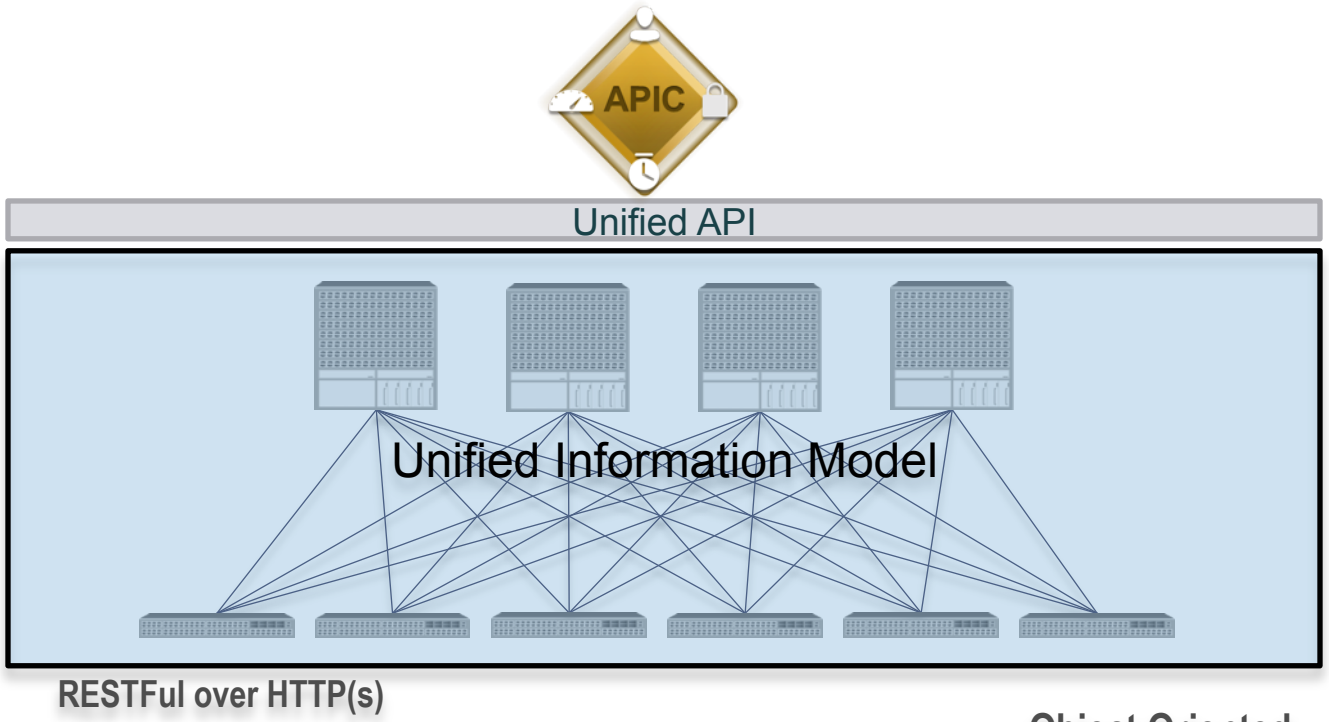


Observer: Health Score



Animation Complete

APIC



- JSON + XML
- ***Unified***: automatically delegates request to corresponding components
- ***Transactional***
- Single Management Entity yet fully independent components

- ***Comprehensive*** access to underlying information model
- Consistent object naming directly mapped to URL
- Supports object, sub-tree and class-level queries

ACI Programmability Overview

Designed around Open APIs & Open Source

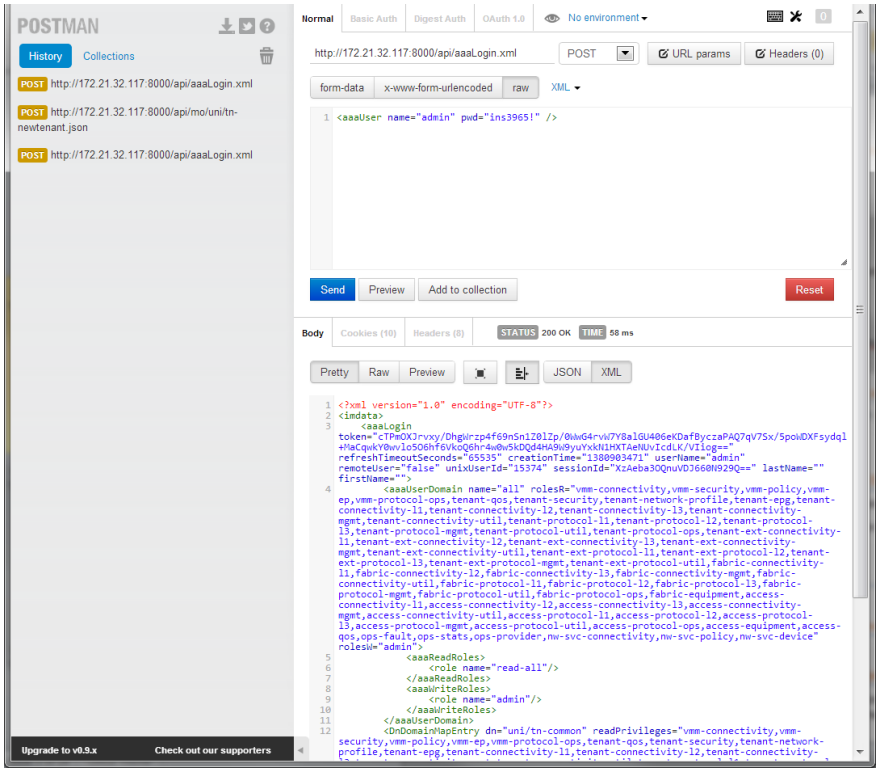
REST API Deep Dive

- APIC Simulator
  - Virtual machine + mininet to simulate leaf-spine
  - Supports all APIC configuration
  - Does not include a datapath
- APIC is based on a hierarchical object model. EVERYTHING is represented as an object and every object can be manipulated via REST.
- REST operations: POST, GET, DELETE
- Support for JSON and XML

REST API Deep Dive (2)

- **Format:** **http://***host*[*:port*]**/api/**{**mo**|**class**}**/**{*dn*|*className*}**.**{**json**/**xml**}[**?***options*]
- /api/—Specifies that the message is directed to the API.
- mo | class—Specifies whether the target of the operation is a managed object (MO) or an object class.
- *dn*—Specifies the distinguished name (DN) of the targeted MO.
- *className*—Specifies the name of the targeted class. This name is a concatenation of the package name of the object queried and the name of the class queried in the context of the corresponding package. For example, the class aaa:User results in a *className* of aaaUser in the URI.
- json | xml—Specifies whether the encoding format of the command or response HTML body is JSON or XML.

REST API Example - Authenticate



Authenticate a user for API Operation

POST: http://apic1/api/aaaLogin.xml

Body (XML):

<aaaUser name="georgewa" pwd="paSSword1" />

Creating a Tenant

REST XML

HTTP Method: POST

Request URL:
http://apic1/api/mo/uni.xml

Payload:

```
<fvTenant name='Tenant1'
status='created,modified'>
</fvTenant>
```

REST JSON

HTTP Method: POST

Request URL:
http://apic1/api/mo/uni.json

Payload:

```
{"fvTenant":{"attributes":
{"dn":"uni/tn-
MyTenant","name":"Tenant1","
rn":"tn-
Tenant1","status":"created"},"c
hildren":[]}}
```

Where to find these examples: https://github.com/datacenter/nexus9000/tree/master/aci

Creating an App Network Profile

REST XML

HTTP Method: POST

Request URL:
http://apic1/api/mo/uni.xml

Payload:

```
<fvTenant name='Tenant1'
status='created,modified'>
<fvAp name='WebApp'>
</fvAp>
</fvTenant>
```

REST JSON

HTTP Method: POST

Request URL:
http://apic1/api/mo/uni.json

Payload:

```
{"fvTenant": {"attributes":
{"name": {"value": "Tenant1"}},
"children":
[{"fvRsTenantMonPol": {}},
{"fvAp": {"attributes": {"name":
{"value":
"WebApp"}}}},
{"fvRsResMonEPGPol": {}}]}}
```

Creating an App Network Profile

REST XML

HTTP Method: POST

Request URL:
http://apic1/api/mo/uni.xml

Payload:

```
<fvTenant name="T1"
status="created,modified">
 <fvAp name="www.T1.com"
status="created,modified">
  <fvAEPg name="WEB"
status="created,modified">
....
```

REST JSON

HTTP Method: POST

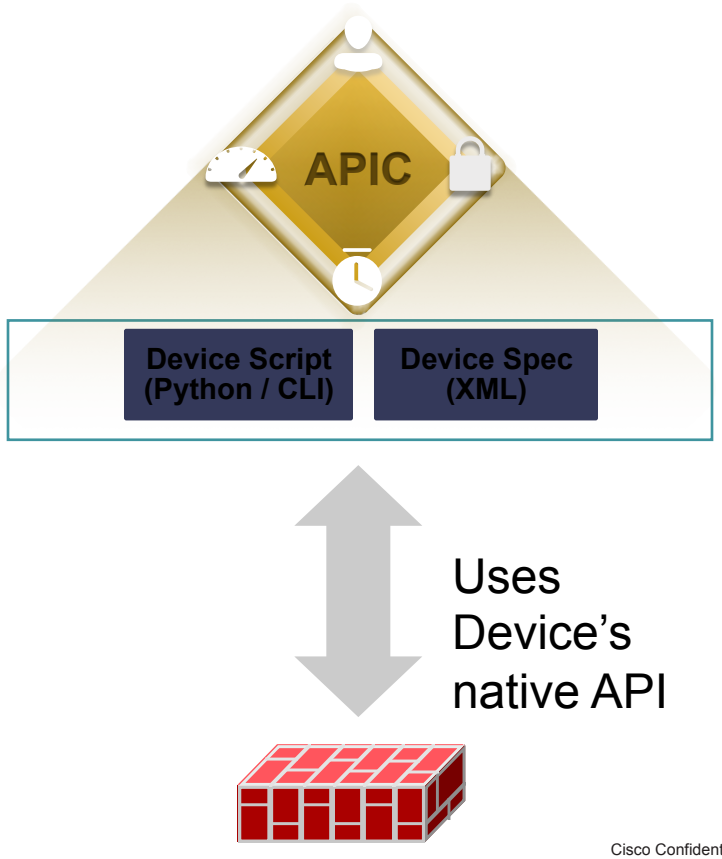
Request URL:
http://apic1/api/mo/uni.json

Payload:

```
{"fvTenant": {"attributes":
{"name": {"value": "T1"}},
"children": [{"fvAp":
{"attributes": {"name":
{"value": "www.T1.com"}},
"children": [{"fvAEPg":
....
```

L4-7 Scripting API

- APIC interfaces with the device using python scripts
- APIC calls device specific python script function on various events
- APIC uses device configuration model provided in the package to pass appropriate configuration to the device scripts
- Device script handlers interface with the device using its REST or CLI interface



Importing/Exporting via XML/ JSON

Scripting Through the API – Tenants, and Bridge Domains

```
<fvTenant dn="uni/tn-TigerTeam" name="TigerTeam">
    <fvCtx name="TigerTeamL3"/>
    <!-- bridge domain -->
    <fvBD arpFlood="true" name="BDTigerTeam" unkMacUcastAct="flood">
        <fvRsCtx tnFvCtxName="TigerTeamL3"/>
        <fvSubnet ip="10.1.100.1/24" scope="private"/>
    </fvBD>
```

Creating a tenant named TigerTeam

Creating an L3-Context

Creating an Bridge Domain

Enabling flooding on the BD for unknown unicast

Associate with L3-Context

Assign Subnet/Gateway

End BD config

Scripting Through the API – Tenants, and Bridge Domains

```
<!-- Security -->

<aaaDomainRef dn="uni/tn-TenantInfra/domain-tenantinfra" name="tenantinfra"/>
```

Enter security config for tenant

```
<!-- Local Contracts -->
```

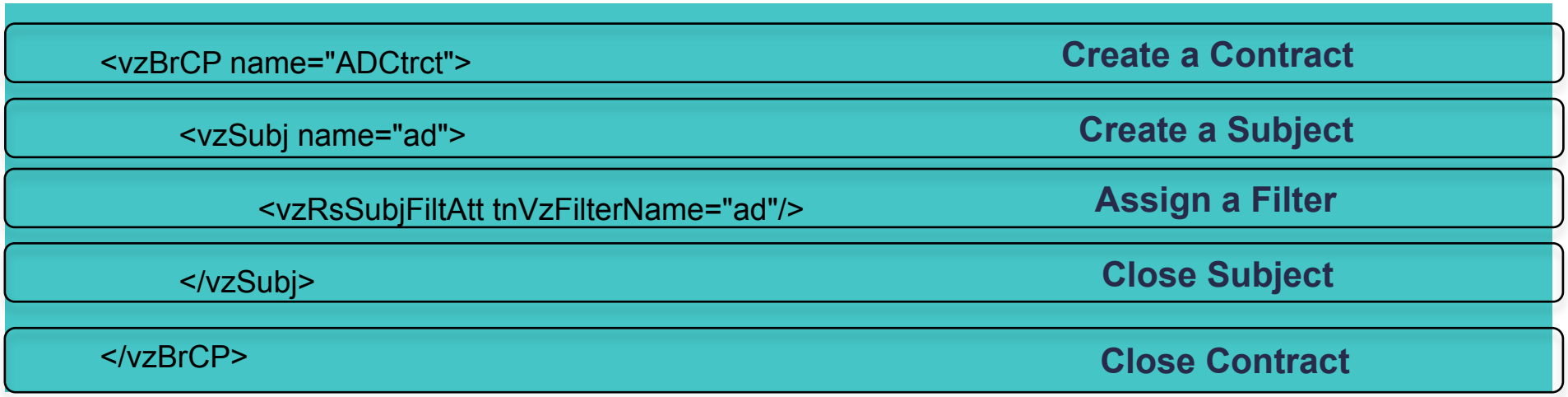
```
<vzFilter name="ad">

    <vzEntry dFromPort="1099" dToPort="1099" etherT="ipv4" name="FilterEntry" prot="6"/>

</vzFilter>
```

Create a filter

Scripting Through the API – Tenants, and Bridge Domains

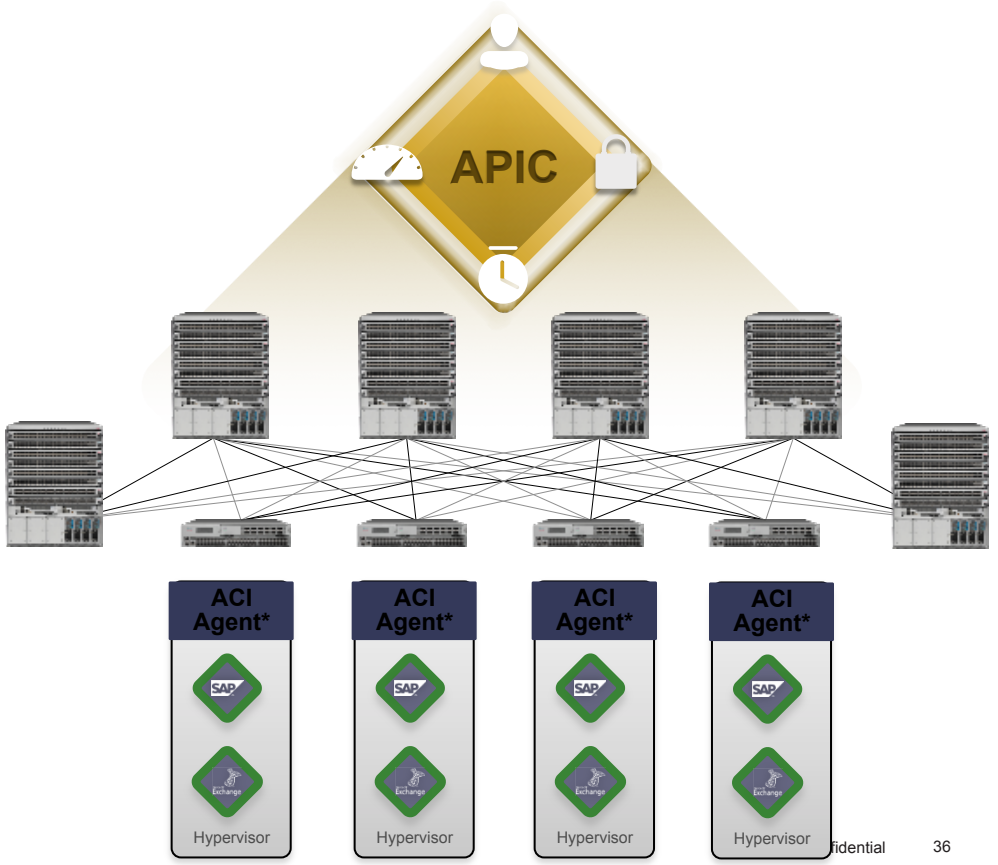


Scripting Through the API – Tenants, and Bridge Domains

```
<!-- Application Profile-->                          Create App Profile
<fvAp name="TenantInfraServ">
    <fvAEPg name="AD">                                 Create EPG
        <fvRsBd tnFvBDName="BDTigerTeam"/>             Assign BD
        <fvRsProv tnVzBrCPName="ADCtrct"/>             Assign Provider Contract
    </fvAPg>
```

ACI Fabric Attached Device API

- Open API between a controller and a set of network devices designed to natively support ACI Policy
- Supported over TCP/SSL/HTTP
- Logical policy model is pushed directly from controller (Policy Authority) to device (Policy Element), which renders it in software / hardware
- Policy Elements could be leaf switches in a network fabric, hypervisor switches, or L4-7 devices



Conclusion & Key Links

- ACI is based on policy model abstraction that models application semantics

- ACI supports a number of open northbound and southbound APIs that are easy to use:
  - REST API
  - Python API
  - L4-7 Scripting
  - ACI Agent (native ACI Policy API)

- Github
  - https://github.com/datacenter/nexus9000
- Devnet
  - https://developer.cisco.com/site/tech/networking/routers-switches/n9k/overview/

Thank you.