Design guide
Cisco public

The bridge to possible

Cisco Secure ADC and Cisco ACI Ansible Automation

Contents

Introduction

Cisco® Secure Application Deliver Controller (ADC) ensures that applications are available, even at times of peak traffic, enhancing reliability and trust within tenant networks and optimizing data center performance. Individual instances or clusters of Cisco Secure ADCs can be configured to act as a virtual server and receive traffic for distribution to backend servers on a specific virtual IP address and port for each application. The required processing is performed on the application traffic (such as SSL offload or Web Application and API Protection functions), and traffic is distributed to the appropriate backend ("real") servers on which the application is running.

This document demonstrates the interoperability of Cisco Secure ADC (OEM Radware Alteon[1]) and Cisco ACI while using Ansible for automation.

It provides an example of Day 1 automation for provisioning an ADC service infrastructure, for an example web application. The application consists of several web servers (real servers) that will be presented as a single application service address (VIP). The resulting ADC service should provide capabilities to allow incoming HTTP/S traffic destined for the application service address to be dynamically/algorithmically distributed (load balanced) to each real server.

Optionally, enhanced features like SSL Inspection, Web Application Firewall (WAF) (aka, Web Application and API Protection [WAAP]), and API protection may be enabled on the Cisco Secure ADC platform.

Automating Application Delivery Services

Whether you are performing the initial configuration or orchestrating workflows within data center service definitions to account for expected or observed capacity, automating L4-7 service enhancements can greatly reduce the time to deploy and dynamically scale applications. This guide provides the common tasks that can be performed via Ansible to automate the provisioning and management of ADC services.

Application delivery can be a provided "service" within itself. Data center operators can provide ADC services to internal or external customers to enhance the performance of the applications that their customers own and maintain. Cisco Secure ADC can also be used to protect applications by providing Web Application Firewall (WAF) and bot management capabilities and can be used to simply provide SSL/TLS processing acceleration or decryption services for security infrastructure.

Supporting Files

The supporting files are located here: https://github.com/Radware/cisco-adc/tree/main/common_ansible_tasks

Supporting Documentation

- Cisco ACI and Cisco Secure ADC Design Guide: www.cisco.com/c/en/us/solutions/data-center-virtualization/application-centric-infrastructure/aci-secure-adc-design-guide.html
- Ansible Documentation: https://docs.ansible.com
- Cisco ACI Fundamentals: www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/4-x/aci-fundamentals/Cisco-ACI-Fundamentals-42x.html
- Ansible Cisco ACI Guide: https://docs.ansible.com/ansible/latest/scenario_guides/guide_aci.html
- Cisco DevNet Learning Labs about ACI: https://developer.cisco.com/learning/tracks/aci-programmability
- Ansible Cisco ACI Collection: https://galaxy.ansible.com/cisco/aci
- Ansible Radware Collection: https://galaxy.ansible.com/radware/radware_modules
- Alteon Multi Cloud: www.radware.com/solutions/alteon-multi-cloud/
- Cisco Secure ADC Datasheet: www.cisco.com/c/en/us/products/collateral/security/secure-adc-alteon-ds.pdf

[1] Alteon is a registered trademark of Radware, Inc.

Demonstration Network Architecture

For illustration purposes, a traditional data center network topology is shown below in Figure 1. The two-tiered, spine-and-leaf architecture is assumed to be the environment where Cisco Secure ADC will reside within these examples. Typically, Cisco Secure ADC will be deployed under the leaf tier without operational visibility to the encapsulated frames. Meaning: Cisco Secure ADC will only see traffic within the unencapsulated overlay networks and routing domains. However, a single Cisco Secure ADC can service multiple tenants due to the segmentation within the platform. The examples provided assume residence within a single tenant.

Figure 1. Typical spine-and-leaf topology

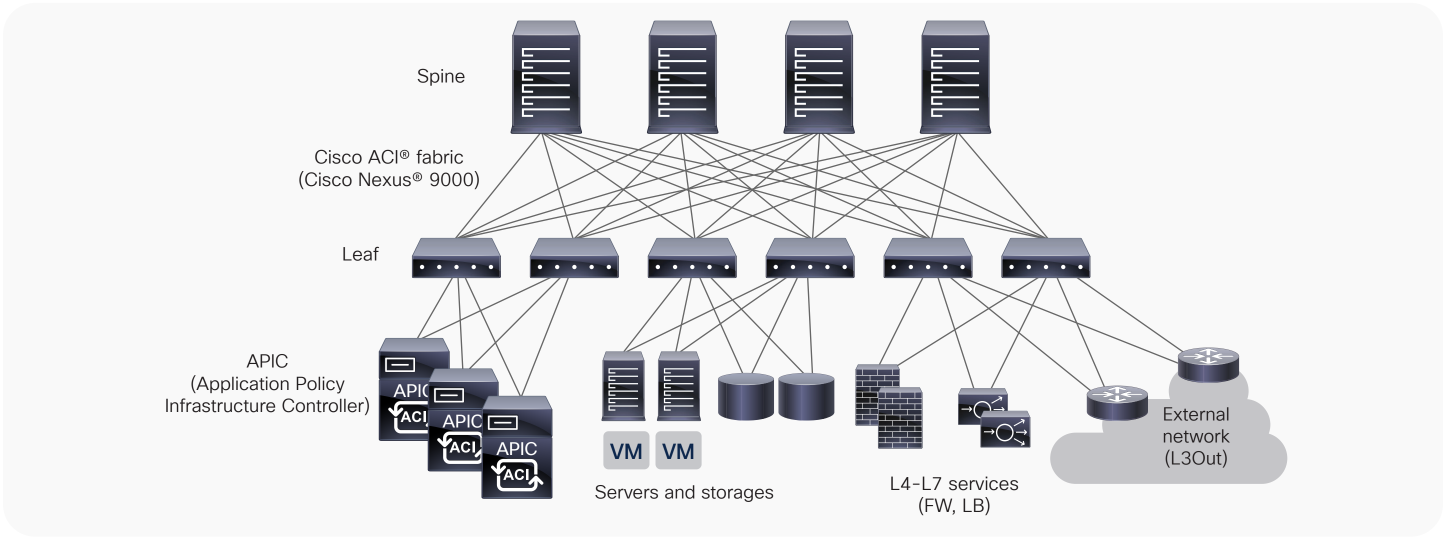
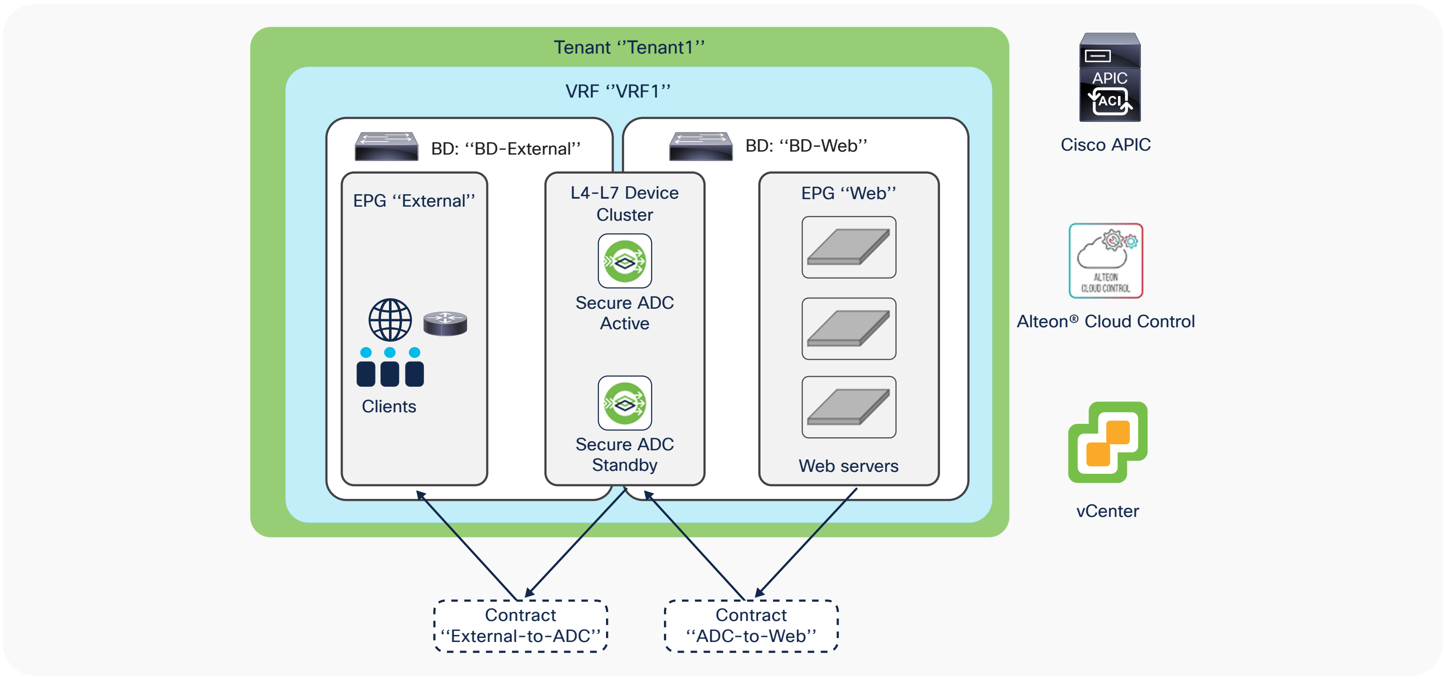


Figure 2 below illustrates the logical relationships between managed objects within Cisco ACI Management Information Model and their correlating physical counterparts.

Figure 2. Logical diagram of ACI objects



Prerequisites

DevOps Station

This document assumes the use of a Linux-based “DevOps Station.” This host is assumed to be the Ansible “control node” within the examples, and it is assumed to be where playbooks and commands are invoked. For this document, a current release of Ubuntu Desktop was used. Any distribution of Linux may be used if the rest of the requirements are met.

Cisco ACI Ansible Support

- Ansible 2.9 or later

Cisco Secure ADC Ansible Support

- Ansible 2.9 or later
- alteon-sdk python package
- Alteon Version 31.0.10.0, 32.2.2.0, or later

Alteon Cloud Control for Cisco ACI

Alteon Cloud Control for ACI is a software module available on Cisco DC App Center that provides integration between Cisco Application Policy Infrastructure Controller (APIC) and Radware Alteon Cloud Control. It enables monitoring and troubleshooting ADC functionality from within Cisco APIC and provides the administrator visibility of the entire application environment (L2-L7) from the ACI interface.

An Alteon Multi Cloud system can deploy Cisco Secure ADC services in multiple AWS, Azure, VMware, and OpenStack cloud environments. Alteon Cloud Control (ACC) allows you to define ADC services for your applications and deploys the Cisco Secure ADC instances required for that service on the application cloud environment, where it continuously monitors the Cisco Secure ADC clusters, performs automatic scaling as needed by the application traffic, and allows you to monitor and troubleshoot application performance. For more information see: www.radware.com/solutions/alteon-multi-cloud/

Alteon Cloud Control is an optional component covered in this guide. The following requirement is only needed if using the Alteon Cloud Control integration for Cisco ACI that is available on the Cisco DC App Center.

- Alteon Cloud Control version 1.4.0 or later

DevOps Station Setup

Once Ubuntu has been installed on the DevOps Station, the following commands must be executed to ensure that remote access, ansible, and supporting modules are present.

```
apt install openssh-server
apt install ansible
apt install python3-pip
apt install python-is-python3
pip install alteon-sdk
ansible-galaxy collection install radware.radware_modules
ansible-galaxy collection install community.vmware
ansible-galaxy collection install cisco.aci
```

Once complete, these commands will result in the installation of:

- An OpenSSH server to gain access to the DevOps Station remotely
- Ansible from the Ubuntu Linux Distribution
- pip Python package installation tool from the Ubuntu Linux Distribution
- The convenience package to set the global Ubuntu Python interpreter to Python version 3 for Ubuntu
- Radware's Python SDK for Alteon/Cisco Secure ADC
- Radware's collection of Ansible modules
- VMware community collection of Ansible modules
- Cisco's collection of Ansible modules for Cisco ACI

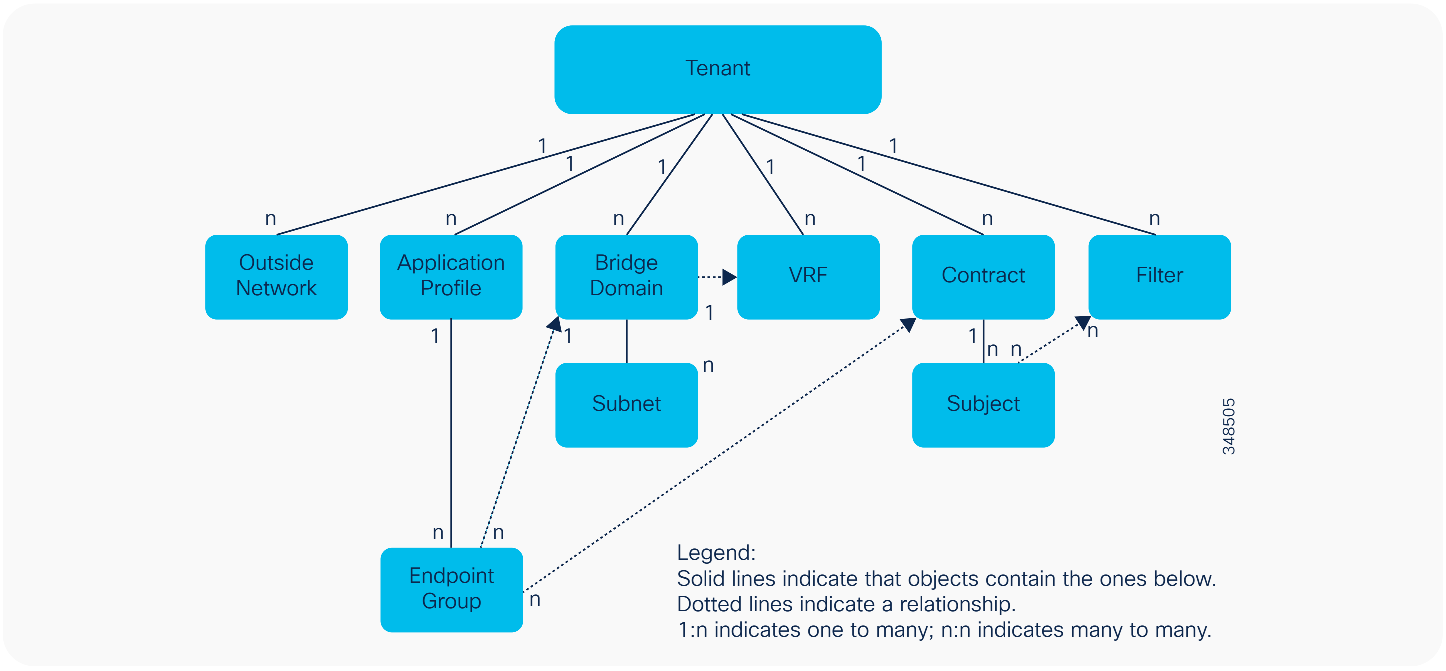
Ansible Automations

This section details common provisioning tasks targeting Cisco ACI and Cisco Secure ADC that can be performed using Ansible. The products, tasks, and playbooks demonstrated within this guide are not limited to the Ansible setup or examples provided. The examples in this guide either assume that inventories and common variables are either managed externally via a customer-specific Ansible configuration or the variables and target systems are explicitly defined within. The variables within the examples are assumed to be defined either within the playbook they are contained or imported.

Any directory hierarchy and configuration of Ansible are only presented as a sample for demonstration purposes. Instructions for deploying full CI/CD frameworks, methods, and tools for full management are outside the scope of this document.

Common Cisco ACI Tasks

For illustration purposes, below is a diagram of a portion of the ACI Management Information Tree (MIT). It is a hierarchical representation of the physical and logical components managed by Cisco APIC. The diagram shows the relationship of different managed objects.


Tenant
Outside Network
Application Profile
Bridge Domain
VRF
Contract
Filter
Subnet
Subject
Endpoint Group
1
n
348505
Legend:
Solid lines indicate that objects contain the ones below.
Dotted lines indicate a relationship.
1:n indicates one to many; n:n indicates many to many.

Create Tenant

In the context of Cisco ACI, a tenant is a unit of isolation from a policy perspective. Tenants can represent a specific customer, application, business unit, application owner, etc. A tenant represents the highest-level container object to differentiate between the objects that define the related networking constructs (such as VRFs, bridge domains, and subnets) and the objects that define the related policies such as application profiles and endpoint groups.

Below is an example of how to create a tenant using the aci_tenant module.

```
    - name: Create APIC Tenant
      cisco.aci.aci_tenant:
        hostname: "{{ apic hostname }}"
        username: "{{ apic_username }}"
        password: "{{ apic_password }}"
        tenant: "{{ apic_tenant }}"
        description: Demo APIC Secure ADC Tenant
        state: present
        validate_certs: false
```

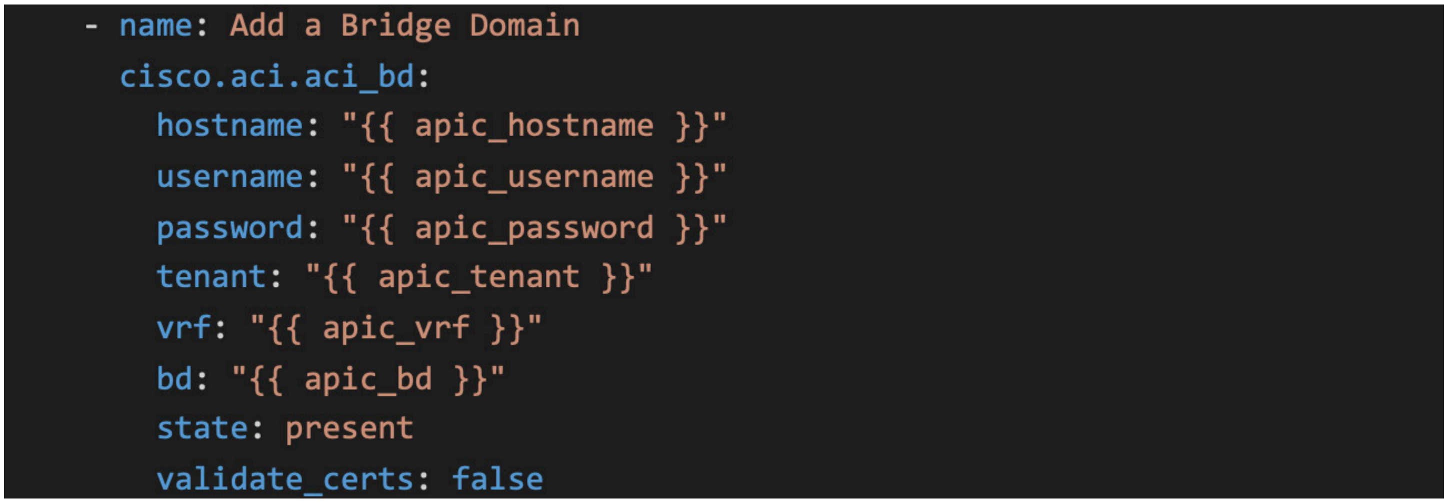
Create Tenant VRF

Virtual Routing and Forwarding (VRF) objects define tenant networks. They are unique Layer 3 forwarding and application policy domains. A tenant can have multiple VRFs.

```
    - name: Create a VRF for the Tenant
      cisco.aci.aci_vrf:
        hostname: "{{ apic_hostname }}"
        username: "{{ apic_username }}"
        password: "{{ apic_password }}"
        vrf: "{{ apic_vrf }}"
        tenant: "{{ apic_tenant }}"
        state: present
        validate_certs: false
```

Create Bridge Domain and Associate with VRF

Within ACI, the bridge domain represents the Layer 2 broadcast boundary and is generally linked to a VRF within a tenant. The example presented creates a bridge domain with the default parameters and associates it with an example VRF.

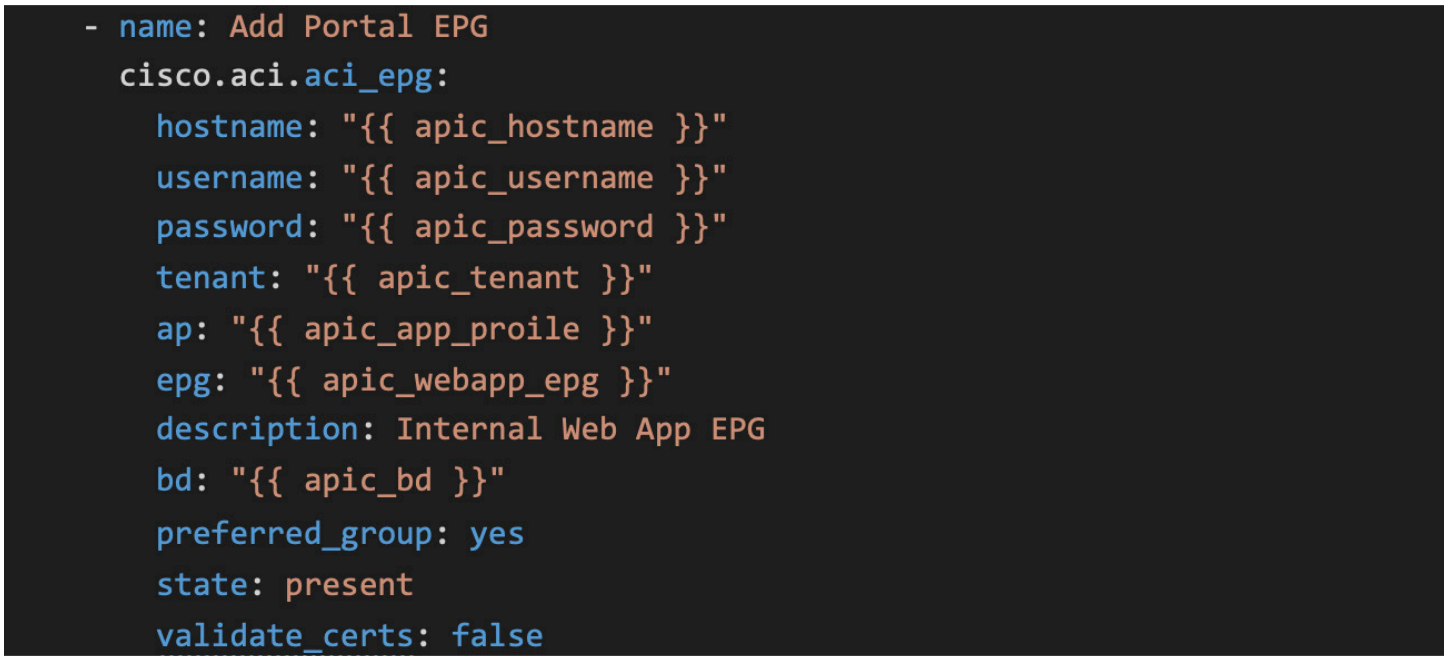


```
  - name: Add a Bridge Domain
    cisco.aci.aci_bd:
      hostname: "{{ apic_hostname }}"
      username: "{{ apic_username }}"
      password: "{{ apic_password }}"
      tenant: "{{ apic_tenant }}"
      vrf: "{{ apic_vrf }}"
      bd: "{{ apic_bd }}"
      state: present
      validate_certs: false
```

Create EPGs

Endpoint Groups (EPGs) are logical groupings of application endpoints within the ACI Management Information Tree (MIT). The design of EPG mappings is specific to application environments. The example provided creates a basic EPG for use in policy enforcement within ACI. The following reference details the usage and design of EPGs within ACI.

www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731630.html



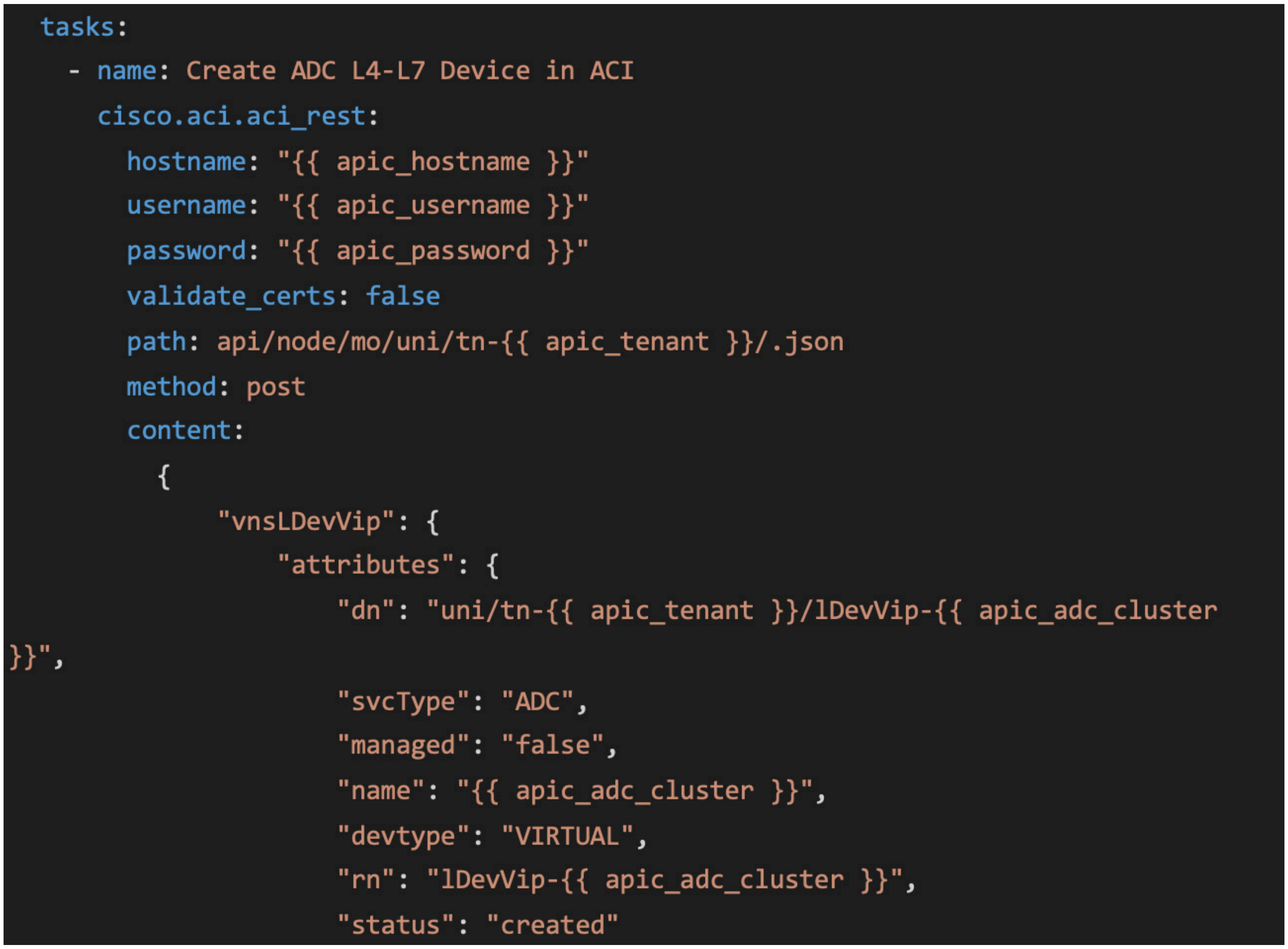
```
  - name: Add Portal EPG
    cisco.aci.aci_epg:
      hostname: "{{ apic_hostname }}"
      username: "{{ apic_username }}"
      password: "{{ apic_password }}"
      tenant: "{{ apic_tenant }}"
      ap: "{{ apic_app_proile }}"
      epg: "{{ apic_webapp_epg }}"
      description: Internal Web App EPG
      bd: "{{ apic_bd }}"
      preferred_group: yes
      state: present
      validate_certs: false
```

Create Logical L4-L7 ADC Device

As of the time of this writing, Ansible modules for adding or removing L4-L7 devices within ACI do not exist. However, the Ansible modules for Cisco ACI include the aci_rest module. This module allows you to automate functions not covered by the rest of the library. The task of creating a logical L4-L7 ADC within ACI must use this module. The example provided below shows how you can send JSON payload via the aci_rest module. Alternatively, YAML-style or XML payload can be used. The XML option may be more suitable for complex tasks that require templating. For this, Ansible provides Jinja2 templating functionality to enable use of dynamic expressions with access to Ansible variables and facts. The XML templates can be explicitly stated in the "content" within your Ansible playbook or included as an external Jinja2 template file.

References

- Cisco ACI REST API Configuration Guide: www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/2-x/rest_cfg/2_1_x/b_Cisco_APIC_REST_API_Configuration_Guide.html
- ACI REST API object model: https://developer.cisco.com/site/apic-mim-ref-api/
- Ansible Jinja2 templating: https://docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html



```
  tasks:
    - name: Create ADC L4-L7 Device in ACI
      cisco.aci.aci_rest:
        hostname: "{{ apic_hostname }}"
        username: "{{ apic_username }}"
        password: "{{ apic_password }}"
        validate_certs: false
        path: api/node/mo/uni/tn-{{ apic_tenant }}/.json
        method: post
        content:
          {
              "vnsLDevVip": {
                  "attributes": {
                      "dn": "uni/tn-{{ apic_tenant }}/lDevVip-{{ apic_adc_cluster
}}",
                      "svcType": "ADC",
                      "managed": "false",
                      "name": "{{ apic_adc_cluster }}",
                      "devtype": "VIRTUAL",
                      "rn": "lDevVip-{{ apic_adc_cluster }}",
                      "status": "created"
                  },
                  "children": [
                      {
                          "vnsCDev": {
                              "attributes": {
                                  "dn": "uni/tn-{{ apic_tenant }}/lDevVip-{{
apic_adc_cluster }}/cDev-{{ adcvm_name }}",
                                  "name": "{{ adcvm_name }}",
                                  "vcenterName": "{{ vcenter_aci_name }}",
                                  "vmName": "{{ adcvm_name }}",
                                  "rn": "cDev-{{ adcvm_name }}",
                                  "status": "created"
                              },
                              "children": []
                          }
                      },
                      {
                          "vnsRsALDevToDomP": {
                              "attributes": {
                                  "tDn": "uni/vmmp-VMware/dom-VMware-VMM",
                                  "status": "created"
                              },
                              "children": []
                          }
                      }
                  ]
              }
          }
```

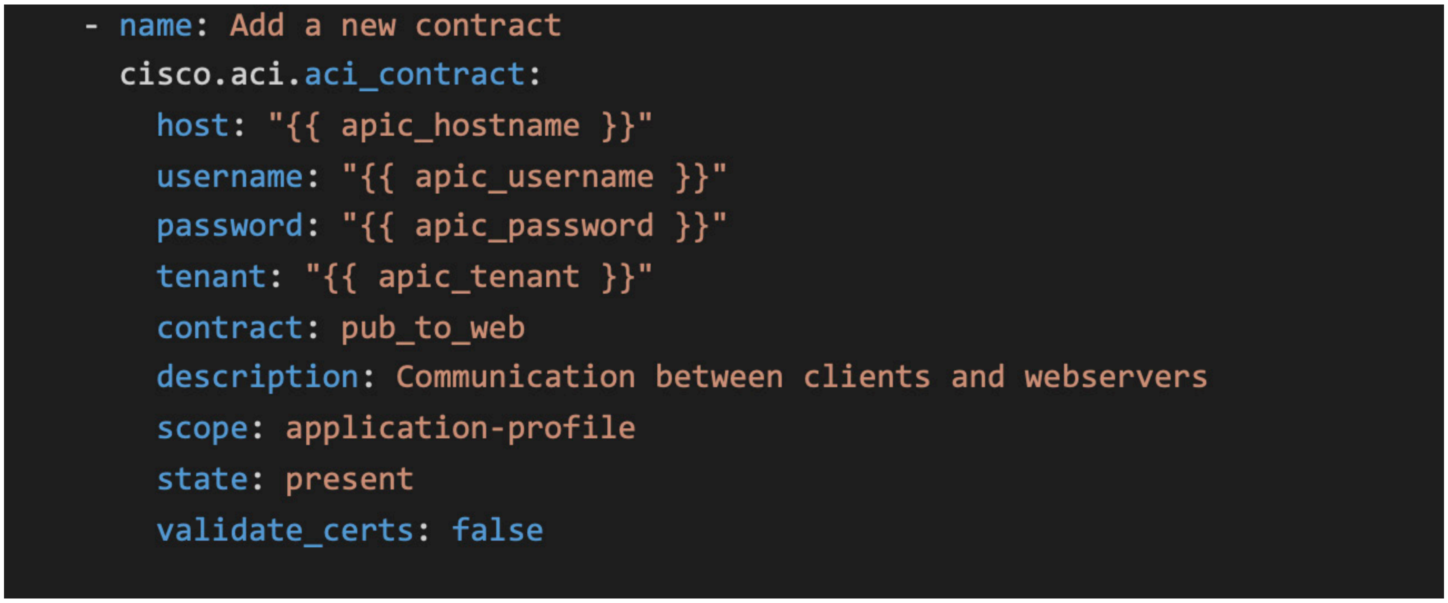
Note: This example will result in a cluster with missing logical interfaces and is only provided as an example for passing API calls via the Ansible module.

Create Contract

Below is a representation of the policy model used by Cisco APIC. An Endpoint Group (EPG) either consumes or provides a contract. The concepts for creating and managing contracts within ACI is detailed here:

www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-743951.html

Contracts can be created and manipulated using Ansible. Below is an example of how to automate the creation of a basic contract.

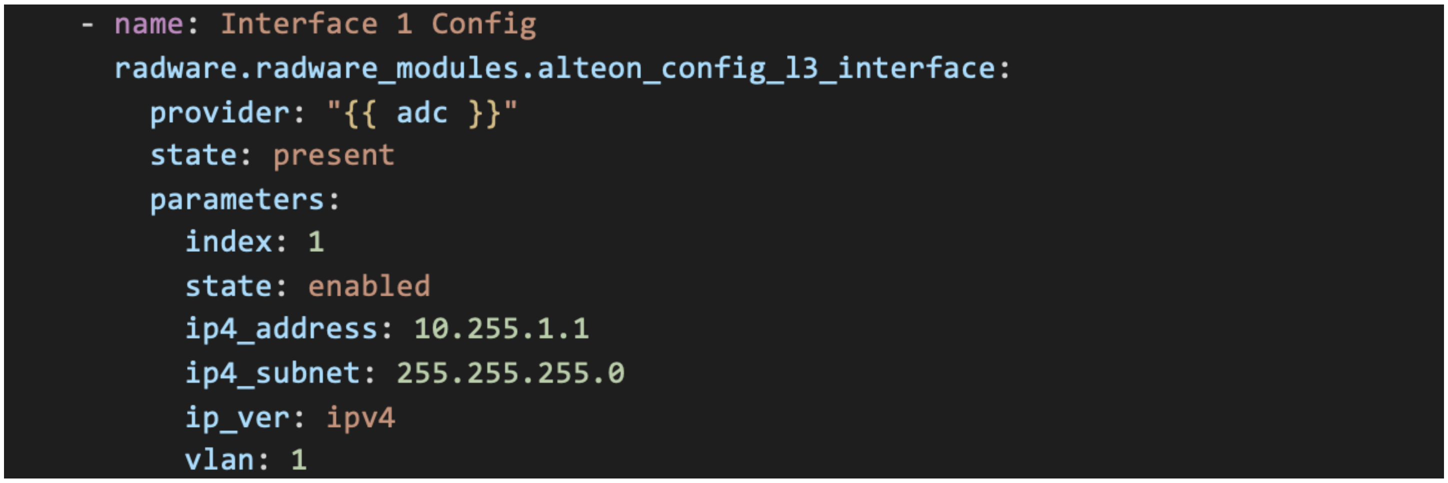


```
  - name: Add a new contract
    cisco.aci.aci_contract:
      host: "{{ apic_hostname }}"
      username: "{{ apic_username }}"
      password: "{{ apic_password }}"
      tenant: "{{ apic_tenant }}"
      contract: pub_to_web
      description: Communication between clients and webservers
      scope: application-profile
      state: present
      validate_certs: false
```

Common Cisco Secure ADC Tasks

Ansible modules have been created to automate the provisioning and management of Cisco Secure ADC instances. The examples provided illustrate the common tasks that can be executed to provision basic network parameters and create a load-balancing service for a managed application.

Create IP Interfaces



```
  - name: Interface 1 Config
    radware.radware_modules.alteon_config_l3_interface:
      provider: "{{ adc }}"
      state: present
      parameters:
        index: 1
        state: enabled
        ip4_address: 10.255.1.1
        ip4_subnet: 255.255.255.0
        ip_ver: ipv4
        vlan: 1
```

Create Default Gateway

```
    - name: Gateway Config
      radware.radware_modules.alteon_config_l3_gateway:
        provider: "{{ adc }}"
        state: present
        parameters:
          index: 1
          state: enabled
          ip4_address: 10.255.1.254
          ip_ver: ipv4
          health_check_type: icmp
```

Create Real Servers

Real servers are the actual servers (physical or virtual) that comprise a pool of resources that Cisco Secure ADC can load-balance incoming traffic to as part of an application.

```
    - name: Real Server 01 Config
      radware.radware_modules.alteon_config_server:
        provider: "{{ adc }}"
        state: present
        parameters:
          index: webserver-01
          state: enabled
          ip_address: 10.255.2.101
```

Create Server Group

Server groups contain the parameters related to the real servers associated with an application and the algorithms used for load balancing and any health check parameters used to determine real server availability.

```
    - name: Server Group Config
      radware.radware_modules.alteon_config_server_group:
        provider: "{{ adc }}"
        state: present
        revert_on_error: true
        parameters:
          index: demo-group
          slb_metric: roundRobin
          health_check_id: tcp
          server_names:
            - webserver-01
            - webserver-02
```

Common ADC Operational Parameters

Administrative Authentication

The following task provisions a primary and secondary RADIUS server for administrative access.

```
- name: Configure RADIUS Authentication
  radware.radware_modules.alteon_config_system_radius_auth:
    provider: "{{ adc }}"
    state: present
    parameters:
      state: enabled
      port: 1812
      primary_ip4_address: 10.10.1.1
      secondary_ip4_address: 10.10.1.2
      timeout_second: 10
      retries: 2
      primary_secret: secret
      secondary_secret: secret
      local_user_priority: localFirst
      local_user_fallback: enabled
      otp: disabled
```

Audit Logging (syslog)

The following task provisions a primary syslog server for Cisco Secure ADC and sets the desired logging level.

```
- name: Configure Logging
  radware.radware_modules.alteon_config_system_logging:
    provider: "{{ adc }}"
    state: present
    parameters:
      show_syslog_on_console: enabled
      configuration_audit: enabled
      extended_log_format: enabled
      session_log_state: enabled
      session_log_mode: disk
      log_trap_system: enabled
      log_trap_management: enabled
      log_trap_virtual_services: enabled
      log_trap_cli: disabled
      log_trap_bgp: enabled
      syslog_servers:
      host1:
        ip4_address: "{{ syslog_01 }}"
        port: 514
        severity: warning4
        facility: local5
        module: all
```

Automation of Compute and Networking Resources

Ansible automation can be employed to manage the lifecycle of networking and compute resources to serve Cisco Secure ADC services. More information on how to use environment-specific modules can be found here: https://docs.ansible.com/ansible/2.9/modules/list_of_cloud_modules.html

In addition to Day 1 automation, virtual resource templates for compute, memory, storage, and networking can be predefined for virtual ADC instances and deployed (or reclaimed) programmatically on demand to service changing capacity requirements.

Within this section of the guide, we adopt the operational service model where individual running instances of Cisco Secure ADC are treated as expendable/repeatable and any critical configurations are present and saved elsewhere (for example: stored in Vision Management, Alteon Cloud Control, a CMDB, etc.). These are then available for other cloned instances to either use or have pushed to them during or shortly after creation; these are "cattle" in the "cattle vs. pets" DevOps model adopted in most public and private cloud environments. In this model, individual instances of Cisco Secure ADC are not "special" and can be easily replicated within the virtual environment. The function they provide is considered critical, but single instances of ADC "appliances" are not. This is in stark contrast to the "monolithic" model where standalone physical appliances are cared for in a much more deliberate way.

Redundancy and high availability are assumed to be provided within the hypervisor environment or within an organization's normal lifecycle and methodologies supporting the care of underlying physical assets or even provided under SLA from an operator's infrastructure-as-a-service vendor. In short, single virtual instances of an ADC are only part of the picture when providing an "ADC service" for an organization—at times, these instances are even temporary in "scale in/scale out" scenarios.

Alteon Cloud Control for Cisco ACI

As previously mentioned, Alteon Cloud Control for ACI is a Cisco DC App Center application that provides integration between Cisco Application Policy Infrastructure Controller (APIC) and Radware Alteon Cloud Control. It enables monitoring and troubleshooting ADC functionality from within Cisco APIC and provides the administrator visibility of the entire application environment (L2-L7) from the ACI interface. Automations are an inherent part of the functionality of Alteon Cloud Control, and compute resources related to Cisco Secure ADC can be optionally provisioned and operated using this solution (either as an augmentation of an existing Ansible-focused framework or as a standalone approach.) The Alteon Cloud Control for ACI application is focused on providing "Day 2" visibility of applications delivered by Cisco Secure ADC and is recommended but not required. The remainder of this section details common tasks to manage compute and networking aspects using only Ansible on the most common hypervisor.

Common VMware ESXi Ansible Tasks

VMware ESX/vSphere is one of the most common hypervisors in use in data center environments today. Naturally, Cisco Secure ADC is supported in this environment. The examples in this guide assume that a VMware environment already exists where the ADC services will reside. Furthermore, it is assumed that Cisco ACI components are already present and managed within (VMM domains, etc.).

Deploy Cisco Secure ADC Virtual Appliance from OVA

The first step to creating an instance of Cisco Secure ADC within VMware ESX/vSphere is to create an instance from the official OVA. Below is an example of how to perform this function within Ansible.

```
tasks:
- name: Deploy ADC from OVA
  community.vmware.vmware_deploy_ovf:
    hostname: "{{ vcenter_hostname }}"
    username: "{{ vcenter_username }}"
    password: "{{ vcenter_password }}"
    validate_certs: no
    datacenter: "{{ adcvm_datacenter }}"
    datastore: "{{ adcvm_datastore }}"
    ova: "{{ adcvm_ova_path }}"
    name: CiscoADC-autobuild-01
    networks: "{u'VM Network':u'{{ adcvm_mgmt_network }}'}"
    power_on: no
    # the following are required when using DRS
    # cluster:
    # resource_pool:
```

This will result in an instantiation of Cisco Secure ADC. The network placement is dependent on the existing environment. The first interface provisioned is assigned to the dedicated management interface of the Cisco Secure ADC instance. Once present, the ADC can be powered up as is or used as a basis for creating a stored template for ease and speed of deployment within the environment.

Deploy Cisco Secure ADC Virtual Appliance from Stored Template

Once the initial OVA has been provisioned within the vCenter, it can be converted to a template within the vCenter interface. This is generally a deliberate task that is assumed to happen infrequently.

Once the template has been created, new ADC instances can be created using it. Below is an example.

```
tasks:
- name: Create an ADC from Template
  community.vmware.vmware_guest:
    hostname: "{{ vcenter_hostname }}"
    username: "{{ vcenter_username }}"
```

```
      password: "{{ vcenter_password }}"
      validate_certs: no
      datacenter: "{{ adcvm_datacenter }}"
      template: "{{ adcvm_template }}"
      name: CiscoADC-01
      folder: "{{ adcvm_folder }}"
      datastore: "{{ adcvm_datastore }}"
      # The following are needed if DRS is configured
      # resource_pool:
      # cluster:
```

Query ESXi for Cisco ADC Virtual Appliance Resource Information

When a new guest ADC is created, there are items that are dynamically created (such as UUIDs) that other modules and tasks require. These can be obtained by using the vmware_guest_info module. Below is an example of a task that returns all the information related to the virtual ADC.

```
  - name: Get VM Info
    register: vminfo
    community.vmware.vmware_guest_info:
      hostname: "{{ vcenter_hostname }}"
      username: "{{ vcenter_username }}"
      password: "{{ vcenter_password }}"
      validate_certs: no
      name: "{{ adcvm_name }}"
      datacenter: "{{ adcvm_datacenter }}"
      folder: "{{ adcvm_folder }}"

  - debug:
      var: vminfo
```

Changing Running State of Cisco Secure ADC VA

The example below shows how to change the "state" of a guest ADC. To change the operational state, the options are:

- poweredoff
- poweredon
- restarted
- suspended

Note: These are not the only options possible for the "state" (this variable also controls if the ADC is present within the inventory or not), so care should be taken to ensure that if the intention is to reboot or power cycle the virtual ADC, you only set it to one of the options above.

```
  - name: Change power state of Cisco Secure ADC
    community.vmware.vmware_guest:
      hostname: "{{ vcenter_hostname }}"
      username: "{{ vcenter_username }}"
      password: "{{ vcenter_password }}"
      validate_certs: no
      name: SecureADC-autoclone-01
      datacenter: "{{ adcvm_datacenter }}"
      state: poweredoff
```

Removing a Cisco Secure ADC VA

The following example details how to remove a Cisco Secure ADC from the VMware inventory. Note that the state of the ADC must be "poweredoff" to be removed (see example above).

```
  - name: Delete Cisco Secure ADC
    community.vmware.vmware_guest:
      hostname: "{{ vcenter_hostname }}"
      username: "{{ vcenter_username }}"
      password: "{{ vcenter_password }}"
      validate_certs: no
      name: AlteonOS-autoclone-01
      datacenter: "{{ adcvm_datacenter }}"
      state: absent
```

Note: This will completely remove the ADC from the inventory within VMware and all data will be lost. Use with care.

Summary

Within this guide we have provided a foundation of common tasks that can be automated using Ansible when managing Cisco Secure ADC deployed within Cisco ACI Fabric. Automation of the provisioning of Cisco Secure ADC services can ensure rapid application deployment and horizontal scaling when on-demand expansion is required. The tasks within this guide can be used for initial provisioning, a basis for repeatable and programmatic operational functions, or even a starting point for reusable workflows to integrate Cisco Secure ADC into CI/CD pipelines.

Ansible can be used for many more operational tasks, and playbooks can be crafted to manage and orchestrate changes across platforms from different vendors. We encourage further exploration into the references section and the Ansible documentation for further details on the possibilities.

Example Playbooks

The following sections provide a collection of related tasks already covered in this document. The purpose is that they can be used to assist in the creation of comprehensive playbooks with site-specific detail (generally provided via Ansible's native external inclusion mechanism) or to merely illustrate the sequence in which the tasks can be performed.

Cisco ACI Basic Setup Playbook

The following playbook will automate many of the manual steps needed to establish a working environment within ACI. The example assumes that certain finalizing steps are performed manually based on operator knowledge about the environment (interfaces, addresses, protocols, etc.).

```
- name: Cisco Secure ADC, Cisco ACI, and VMWare Provisioning
  hosts: localhost
  connection: local
  gather_facts: true

  vars_files:
    - vars.yaml

  tasks:
    - name: Add a new contract
      cisco.aci.aci_contract:
        host: "{{ apic_hostname }}"
        username: "{{ apic_username }}"
        password: "{{ apic_password }}"
        tenant: "{{ apic_tenant }}"
        contract: pub_to_web
        description: Communication between clients and webservers
        scope: application-profile
        state: present
        validate_certs: false

    - name: Create Application Profile
      cisco.aci.aci_ap:
        hostname: "{{ apic_hostname }}"
        username: "{{ apic_username }}"
        password: "{{ apic_password }}"
        tenant: "{{ apic_tenant }}"
        ap: "{{ apic_app_proile }}"
        description: Intranet Portal
        state: present
        validate_certs: false
  - name: Add Client EPG
    aci_epg:
      hostname: "{{ apic_hostname }}"
      username: "{{ apic_username }}"
      password: "{{ apic_password }}"
      tenant: "{{ apic_tenant }}"
      ap: "{{ apic_app_proile }}"
      epg: "{{ apic_client_epg }}"
      description: Internal Clients EPG
      bd: "{{ apic_bd }}"
      monitoring_policy: default
      preferred_group: yes
      state: present
      validate_certs: false

  - name: Add Portal EPG
    aci_epg:
      hostname: "{{ apic_hostname }}"
      username: "{{ apic_username }}"
      password: "{{ apic_password }}"
      tenant: "{{ apic_tenant }}"
      ap: "{{ apic_app_proile }}"
      epg: "{{ apic_webapp_epg }}"
      description: Internal Web App EPG
      bd: "{{ apic_bd }}"
      monitoring_policy: default
      preferred_group: yes
      state: present
      validate_certs: false

  - name: Add a new contract to EPG binding
    aci_epg_to_contract:
      host: "{{ apic_hostname }}"
      username: "{{ apic_username }}"
      password: "{{ apic_password }}"
      tenant: "{{ apic_tenant }}"
      ap: "{{ apic_app_proile }}"
      epg: "{{ apic_webapp_epg }}"
      contract: "{{ apic_portal_contract }}"
```

```
        contract_type: provider
        state: present
        validate_certs: false

    - name: Create ADC L4-L7 Device in ACI
      aci_rest:
        hostname: "{{ apic_hostname }}"
        username: "{{ apic_username }}"
        password: "{{ apic_password }}"
        validate_certs: false
        path: api/node/mo/uni/tn-{{ apic_tenant }}/.json
        method: post
        content:
          {
              "vnsLDevVip": {
                  "attributes": {
                      "dn": "uni/tn-{{ apic_tenant }}/lDevVip-{{ apic_adc_cluster }}",
                      "svcType": "ADC",
                      "managed": "false",
                      "name": "{{ apic_adc_cluster }}",
                      "devtype": "VIRTUAL",
                      "rn": "lDevVip-{{ apic_adc_cluster }}",
                      "status": "created"
                  },
                  "children": [
                      {
                          "vnsCDev": {
                              "attributes": {
                                  "dn": "uni/tn-radware/lDevVip-{{ apic_adc_cluster }}/cDev-{{
adcvm_name }}",
                                  "name": "{{ adcvm_name }}",
                                  "vcenterName": "{{ vcenter_aci_name }}",
                                  "vmName": "{{ adcvm_name }}",
                                  "rn": "cDev-{{ adcvm_name }}",
                                  "status": "created"
                              },
                              "children": []
                          }
                      },
                     {
                         "vnsRsALDevToDomP": {
                             "attributes": {
                                 "tDn": "uni/vmmp-VMware/dom-VMware-VMM",
                                 "status": "created"
                             },
                             "children": []
                         }
                     }
                 ]
             }
         }
     delegate_to: localhost
```

Cisco Secure ADC Basic SLB Playbook

This example illustrates an entire playbook to provision networking and basic load-balancing functions (on a running Cisco Secure ADC) for an example application.

```
- hosts: localhost
  vars:
      adc:
        server: 192.168.86.101
        user: admin
        password: radware
        validate_certs: no
        https_port: 443
        ssh_port: 22
        timeout: 10

  tasks:
    - name: Interface 1 Config
      radware.radware_modules.alteon_config_l3_interface:
        provider: "{{ adc }}"
        state: present
        parameters:
          index: 1
          state: enabled
          ip4_address: 10.255.1.1
          ip4_subnet: 255.255.255.0
          ip_ver: ipv4
          vlan: 1

    - name: Interface 2 Config
      radware.radware_modules.alteon_config_l3_interface:
        provider: "{{ adc }}"
        state: present
        parameters:
          index: 2
          state: enabled
          ip4_address: 10.255.2.1
          ip4_subnet: 255.255.255.0
          ip_ver: ipv4
          vlan: 2

    - name: Gateway Config
      radware.radware_modules.alteon_config_l3_gateway:
        provider: "{{ adc }}"
        state: present
        parameters:
          index: 1
          state: enabled
          ip4_address: 10.255.1.254
          ip_ver: ipv4
          health_check_type: icmp

    - name: Real Server 01 Config
      radware.radware_modules.alteon_config_server:
        provider: "{{ adc }}"
        state: present
        parameters:
          index: webserver-01
          state: enabled
          ip_address: 10.255.2.101

    - name: Real Server 02 Config
      radware.radware_modules.alteon_config_server:
        provider: "{{ adc }}"
        state: present
      parameters:
        index: webserver-02
        state: enabled
        ip_address: 10.255.2.102

  - name: Server Group Config
    radware.radware_modules.alteon_config_server_group:
      provider: "{{ adc }}"
      state: present
      revert_on_error: true
      parameters:
        index: demo-group
        slb_metric: roundRobin
        health_check_id: tcp
        server_names:
          - webserver-01
          - webserver-02

  - name: Virtual Server Config
    radware.radware_modules.alteon_config_virtual_server:
      provider: "{{ adc }}"
      state: present
      parameters:
        index: demo-vserver
        state: enabled
        ip_address: 192.168.255.100

  - name: Service Config
    radware.radware_modules.alteon_config_virtual_service:
      provider: "{{ adc }}"
      state: present
      parameters:
        index: demo-vserver
        service_index: 1
        service_port: 80
        server_group_name: demo-group
        application_type: http
        nat_mode: address
        nat_address: 192.168.255.80
```

```
- name: Apply Configuration
  radware.radware_modules.alteon_mng_config:
    provider: "{{ adc }}"
    command: apply

- name: Save Configuration
  radware.radware_modules.alteon_mng_config:
    provider: "{{ adc }}"
    command: save
```