**WHAT IS VMANAGE**

* Cisco vManage is a centralised network management system and a part of the Cisco SD-WAN Solution, which helps user to configure and manage the overlay network. It also helps users to configure and manage the edge router devices.
* Cisco vManage can be used to store the certificate and configuration of all Cisco edge devices. vManage can also sign certificates, generate bootstrap and decommission devices.
* Cisco vManage is one of the components of Cisco SD-WAN Solution, below image shows the various components of Cisco SD-WAN

**Node Naming Convention**:

   The naming convention when a network topology is created is as follows:

                  - Software based nodes are named VM#

                  - Hardware based nodes are named as PM#

**Management IP address convention**:

   Every device is configured with a Management interface. When configuring a IP address for a Mgmt interface,

               - Software based nodes are assigned 10.0.**1**.**x**, where **x** is the number based on the VM**x.**

**10 + x (if x  < 10)**

**20 + x (if x  >= 10)**

- Hardware based nodes are assigned 10.0.**99.x**, where **x** is the number based on the PM**x.**

**10 + x (if x  < 10)**

**20 + x (if x  >=  10)**

All the devices that are deployed have SSH access through the MGMT IP.

**IP address based on VNET.**

NOTE: In the vnet based assignment, note the highlighted part of the IP address corresponds to the vnet.

Vnet**2**:10.2.**2**.vm#/24

**1.2 Test Suite**

Each vtest feature suite consists of 3 major files.

1. .yaml files (topology yaml and protocol yaml) for topology bringup.
2. .ttf file for the sequence of test cases to be executed.
3. .py file for feature automation code.

All files are explained with Example of Dynamic Tunnel suite.

**1.3 Yaml files**

* Yaml files define how different hardware and/or VMs are connected to each other, flavor of the nodes (eg, vmanage, vmsart, vedge/cedge etc), connectivity w.r.t vnets/links/ip’s etc.
* Many yaml files already exist for direct use w.r.t combinations of multiple cEdge devices or vEdge device combinations, with/without vManage, multiple vsmarts, etc. Check if any of the yaml file can be reused instead of creating new yamls. You can find yamls files under /home/tester/vtest/vtest/yamls folder. Here are some standard yamls used for multiple suites in vTest
  + viptela\_kvm.yaml
  + viptela\_vulcan.yaml
  + viptela\_vulcan\_csr\_kvm.yaml
  + viptela\_vulcan\_2cedge.yaml
* If needed, custom yaml files can also be created per your requirement. For example, /home/tester/vtest/yamls/topology/dynamictunnel.yaml
* There are many cases that you might want to use variables in topology yaml files, instead hard code the value. Please refer to [Using variables in topology YAML](https://wiki.cisco.com/pages/createpage.action?spaceKey=IOSXENAT&title=Using+variables+in+topology+YAML&linkCreation=true&fromPageId=1274642851)
* vTest supports different HW cEdge boot modes, i.e. day0, regression mode, and config\_reset mode. You can refer to [vTest: HW cEdge boot modes](https://wiki.cisco.com/display/SDWAN/vTest%3A+HW+cEdge+boot+modes) on how to define them in the topology yaml file.

**1.4 .ttf file**

.ttf file **must** include description of the test suite, which release the feature was added, any other tools requirement to run the suite, runner command for reference, etc. Here is the example from test\_DynamicTunnel.ttf

.ttf file should have setup, test cases, and cleanup sections. All suites should have cleanup section. The setup can be split into common omp setup, vmanage setup(optional), feature related setup, etc. All sections should have respective cleanup: omp cleanup, vmanage cleanup(optional), feature related cleanup. We should ensure that all configurations are cleaned up for next suite to run on the testbed.

Similar to define tags such as setup, tests, cleanup, etc., in .ttf file, you can also define test case priorities and customer tags. Please refer to [Define and Run Test Cases with Priority and Customer Tags](https://wiki.cisco.com/pages/createpage.action?spaceKey=IOSXENAT&title=Define+and+Run+Test+Cases+with+Priority+and+Customer+Tags&linkCreation=true&fromPageId=1274642851) for more details.

**1.5 .py file**

The actual code for automation of the feature is defined in .py file. The file consists of all library imports that are needed for automating the feature, global variable declaration, feature related configs and deconfigs, and testcases with description and procedure to verify the testcases .

* Library files already exist and can be used directly in test suites.
* Define global variables from yaml files(do not hardcode) and define other global variable if needed. Do not hardcode variables in testcases.

**1.6 Feature Testing Interactions**

* All features should be tested with config/deconfig/modify with vManage template and CLI templates.
* Dual stack IPv4 and IPv6 transport should be tested
* Dual TLOC – it is important to at least have 2 TLOC.
* OMP interactions if any
* Different types of Policy testing local policies(ACL , ZBFW , Route Policy) and centralized policies(Data Policy, AAR, control Policy).
* CFLOWD and DPI interactions if any.
* XML generation for vManage and statistics checks for the feature.
* Active-Active device with TLOC extensions .
* Different size data packets, tweaking MTU values and jumbo frames .
* Multiple VPN at service side with different kinds of traffic patterns (Ex : UDP , TCP , FTP-active , FTP-passive etc)
* Reboot of controller/edge nodes behavior
* Control connections flap.
* IPv4/IPv6 service side traffic testing.
* NAT interactions if any , DIA –Direct to internet access testing
* SDWAN and non-SDWAN traffic testing

**Process to submit code-review and a merge-pull request:**

* git checkout [next or any other release branch]
* Create a sub-branch from the base branch:
* git branch <your branch name> [if this is a bugfix, name your branch as
* bugfix/<bugid>]
* Work on all your changes in the sub-branch created in (2) and git commit to that branch
* Push your changes along with your branch to github:
* git push origin <branch-name>
* Goto :  <https://wwwin-github.cisco.com/cedge-dev/>
* Click on “New pull request”
* Specify the base branch (from step 1) and under compare drop-down, select your branch (from step 2)
* Click on “Create pull request” and specify reviewers as appropriate

**GIT commands**

* git init
* git clone [url]
* git status
* git add [file or directory name]
* git commit -m "[commit message]
* git push [alias] {branch]
* git pull {aias][branch]
* git branch
* git checkout -b [branch name]
* git merge [branch name]

**Python**

Python links: <https://1drv.ms/u/s!Au_L3EhpQ-gCg6AjMet7NtKtSyJ-MA>

Logging: <https://docs.python.org/3/library/logging.html>

psutil: <https://psutil.readthedocs.io/en/latest/>

Anaconda: <https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html>

Collections: <https://docs.python.org/3/library/collections.html#collections.OrderedDict>.