Ansible ACI Automation Immersion Day Workshop

The age of automation and orchestration is here – this is very evident with cloud-based applications that must be easily provisioned, secured, and scaled. Cisco ACI's open API's allow customers to automate and scale policy across both on-premise and cloud environments, and the operational and economic efficiencies gained are strong incentives to develop a suitable automation strategy.

Red Hat's Ansible is a standout option for automation and orchestration of ACI. Ansible carries fewer adoption challenges than many other solutions in the automation space, especially when considering "time-to-market" for target use cases.

Incorporating several ACI network and security provisioning use cases, customer engineers will see the synergy and benefit from real hands-on with this practical introduction to the power and utility of the Ansible and ACI products.

Overview

This Immersion Day commences with an introduction to ACI, Ansible, and Ansible Tower concepts. Lab One follows and familiarizes the user via a hands-on deployment which explores several ACI features, automating the deployment with Ansible while illustrating some preferred practices. Lab Two takes the student through an enhanced version of the Lab One deployment but using the Ansible Tower platform.

You will use Ansible commands and playbooks to explore and configure a live ACI environment. Success at various steps will be verified by the user through examination of the ACI GUI and live traffic tests using ICMP, SSH, etc.

You will be required to modify some files during the course of this workshop. You will not be required to write your own playbooks as this would require much more time. The playbooks used are open source and thus free to use and modify. That said, writing playbooks and running them in a test environment is one of the best ways to learn.

Disclaimer:

The playbooks used in these labs have been created specifically for the lab scenarios used within. Do not run them in any production environment without careful attention to utility and intended function and a complete understanding of their likely outcome.

Outline

- Part 0: Introduction to the technologies
    - ACI
    - Ansible
    - Ansible Tower
- Part 1: Getting setup:
    - Horizon View Access
        - URL
        - Credentials
    - Ansible Terminal Access
        - Docker desktop
    - ACI GUI Access
        - URL
        - Credentials
    - Test VM Access
        - IP Addresses
        - Credentials
- Lab 1: ACI-Ansible Lab
    - Access the ansible docker "jump-box" terminal on your horizon view desktop, and open the ACI GUI using Chrome on the desktop
    - Explore Lab1 directory and enclosed files (.yml files)
    - Follow the Lab1 flow chart in this document carefully, observing the accompanying notes
        - take time to carefully review playbook structure, function, nuance, possible flaws or potential security concerns
- Lab 2: ACI-Ansible Tower Lab
    - Lab 2 enhances the Lab 1 ACI deployment illustrating and incorporating improved security posture.
    - Follow Lab2 flow chart later in this document to complete lab2

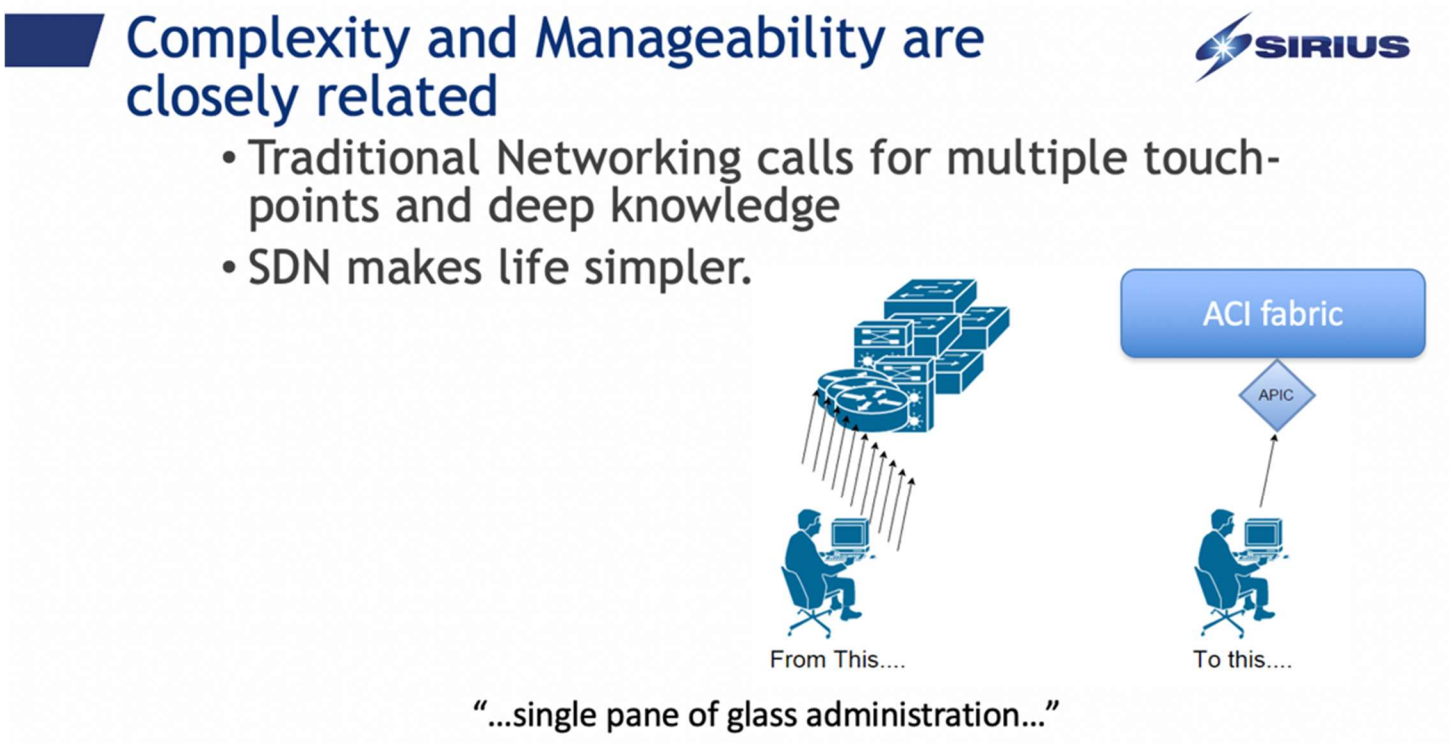
Note: Assume the output shown in the examples below will be different to yours.

Part 0: Introduction to the Technologies

ACI

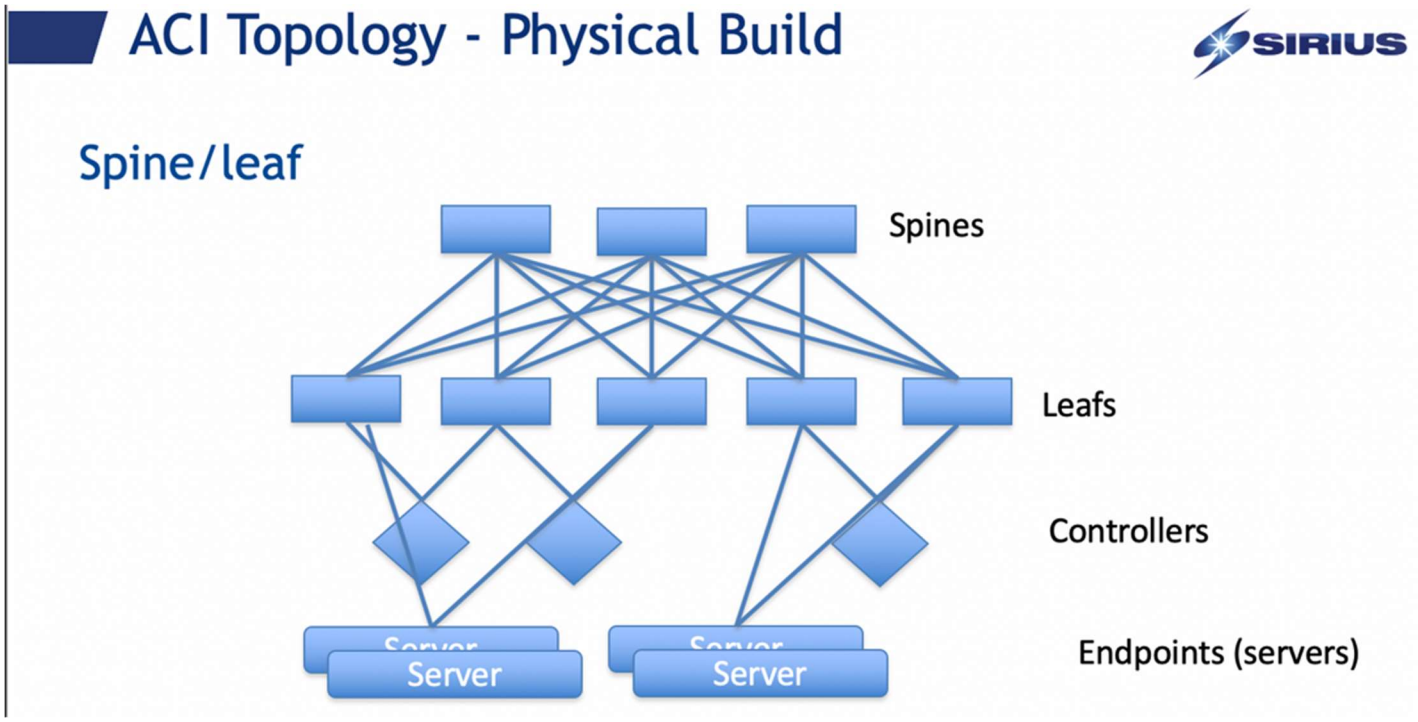
Management: ACI is Cisco's primary software defined network (SDN) solution for data center. In common with the goals of most SDN solutions, it reduces the management plane to a single pane of glass, for an entire data center fabric. The single pane of glass is a controller (the APIC) which is a redundant cluster that distributes the user defined policy to the fabric itself which services the data plane. The controller does not itself participate in the fabric state except to effect change instructions; it merely distributes configurations and accepts telemetry from the fabric components to provide centralized monitoring of network health.

Figure 1: Reducing management complexity with SDN



Physical: Physically, the fabric is engineered as a spine-and-leaf. This physical topology simplifies future scale-up and scale-out requirements, and also creates efficiency in traffic flow as its full mesh means that any attached endpoint is always within two hops of any other. Note in the spine/leaf design, that all endpoints attach directly to leaf switches.

Figure 2: ACI's scalable physical topology; add spines to increase throughput, add leafs to increase endpoints



Secure: ACI can provide fully open network connectivity, or a multi-tenant, secure-zone design (or a hybrid of these). Deep packet inspection, load-balancing and IPS services can be inserted between secure zones as part of an ACI contract - or where appliances like these are not required, the ACI fabric hardware can enforce security flows at the Layer-4 port without protocol inspection.

Figure 3: Security is readily applied in the SDN fabric

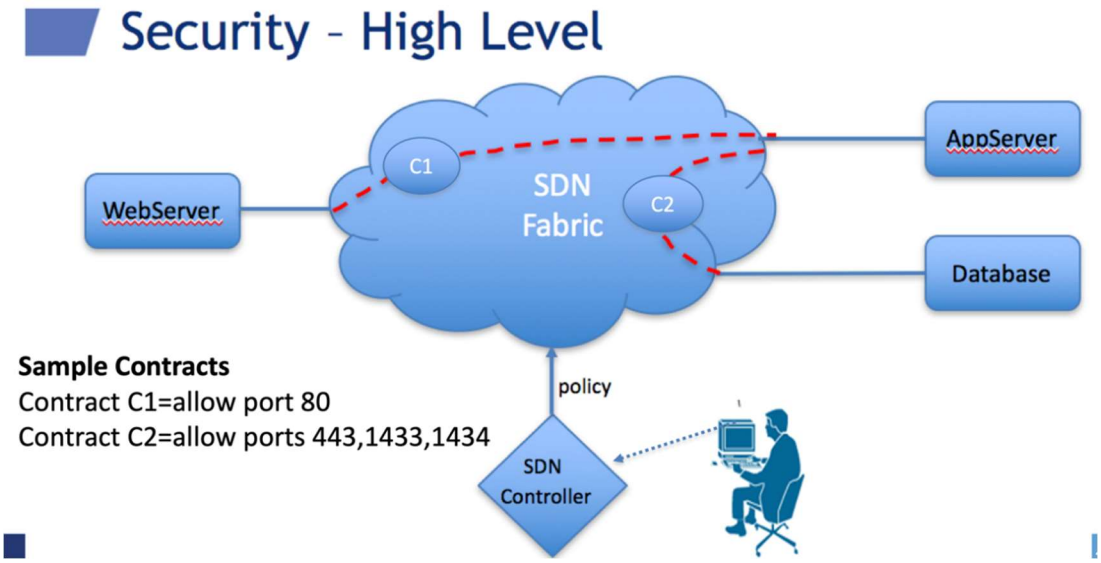
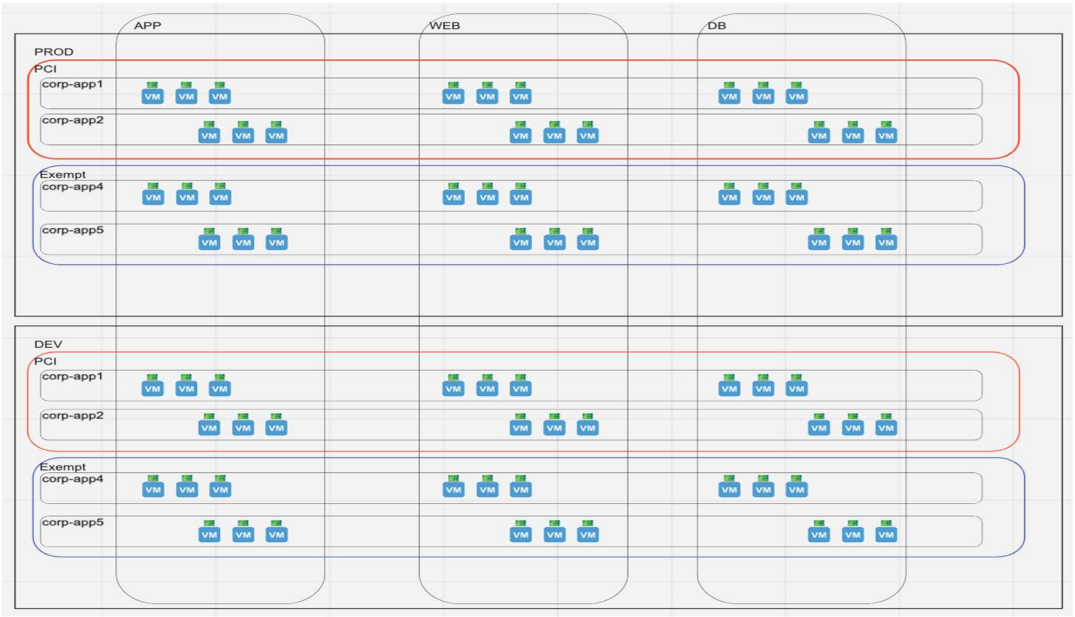


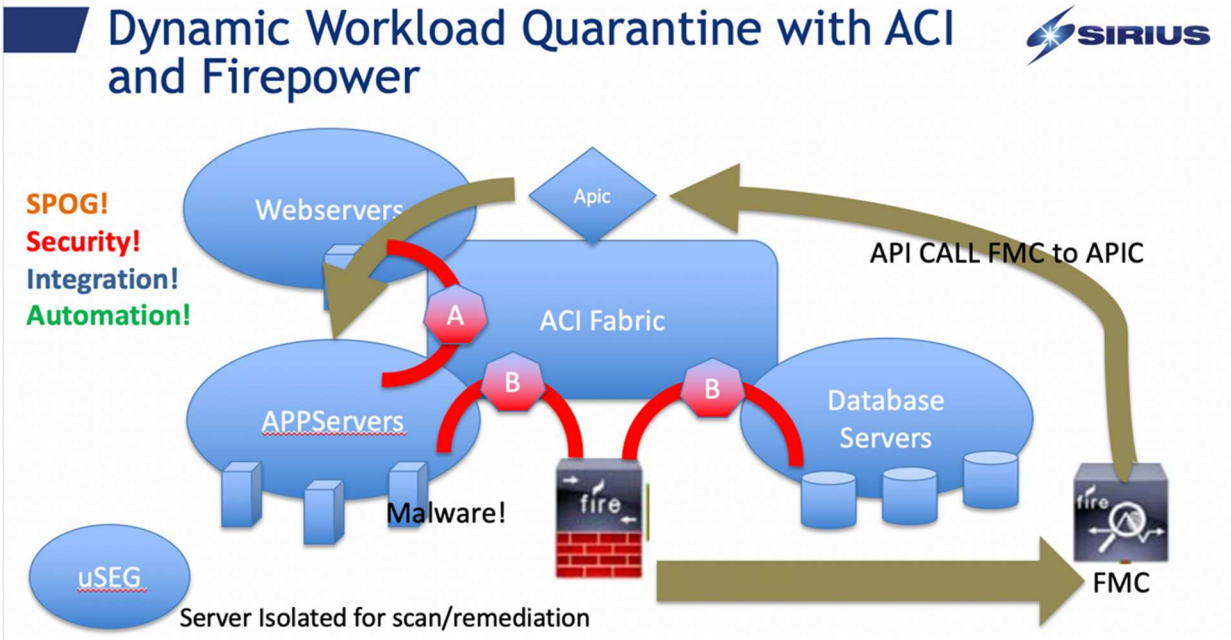
Figure 4: ACI Multi-tenant Segmentation Design: 3-tier application



Note that the security hierarchy in ACI is described by the *Tenant, VRF, Application Profile, and Endpoint Group* constructs. In figure 4 above, PROD and DEV are tenants, PCI and Exempt are VRFs, corp-appX are Application profiles while APP WEB and DB describe the Endpoint groups. The unique endpoint groups are indicated by the 3-vm boundaries; so there are 24 unique endpoint groups in this diagram. Communication rules (not shown here) known as contracts are applied between endpoint groups to enable traffic flow. Thus, the endpoint group can accurately be regarded as a "security zone" and the contracts are a lot like firewall-rules.

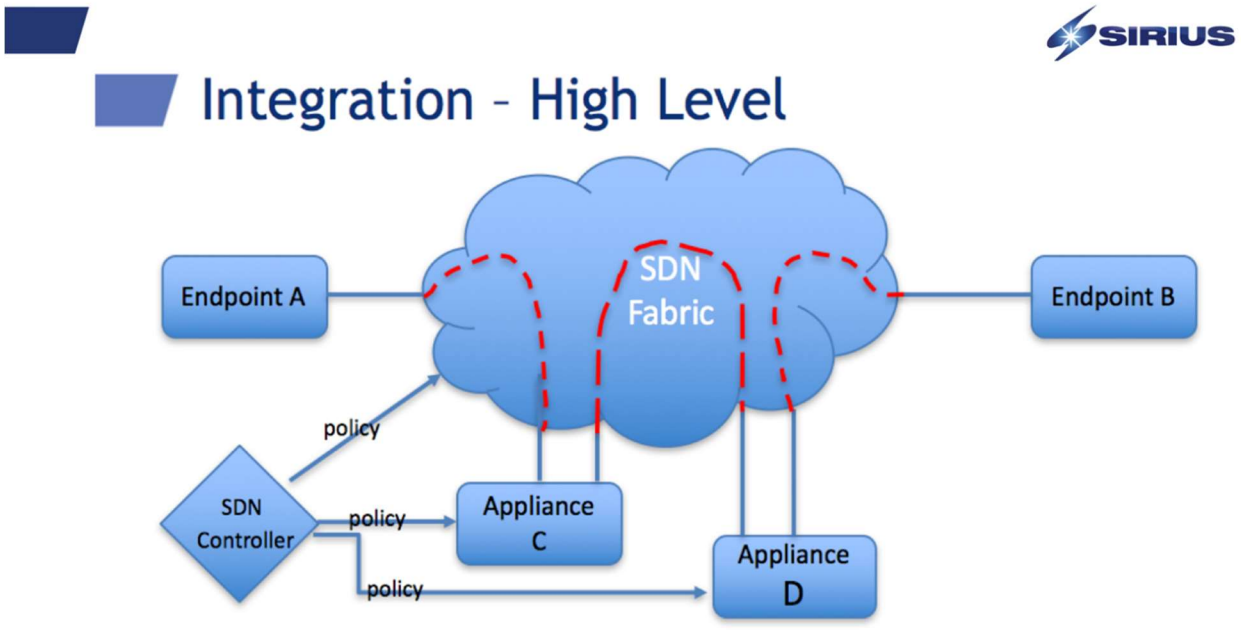
Programmatic Control: With the APIC's single pane-of-glass comes a fully open API. This means that, programmatically, *any* ACI capability can be deployed through industry standard REST calls against the fabric controller. The API is not the only configuration tool, both CLI and GUI are also available - but of note is that they leverage the API also; they are really a "shim" in front of it. But the API is the key to leveraging automation effectively. Some popular automation tools are Puppet, Chef, and Ansible.

Figure 5: A Cisco ACI programmatic integration example illustrates the power of the open API for production use.



Integrations: Open API's and plug-ins cater to feature-rich integrations. Meaning that Orchestration goals can be met with ecosystem partners and "SDN" ultimately becomes part of "SDX", where terms like IaaS and PaaS and SaaS apply and the network is a piece of a larger coherent system that can be defined, provisioned, scaled and tracked by way of code and a true Dev/Ops posture. The data center network can now become unified with the larger environment in a much more cohesive and efficient way, as opposed to the inefficiencies and complexities of a piecemeal approach. This is a grand but realistic view, however it is part of a larger journey for those without prior exposure.

Figure 6: A Generic SDN Fabric Services Insertion Example



Different: ACI can attach endpoints with Ethernet, but Ethernet's constraints (and limitations) can be kept local to the leaf switch port. This allows flexibility in the deployed design and is one reason why ACI can be made truly "Application Centric".

Ansible

Orchestration and automation are not the same thing. But, Ansible can do both. Automation usually is implemented to fulfil a goal to replicate identical or near-identical repetitive actions. For example, the automatic transmission. Using a similar **analogy**, orchestration is akin to what the "self-driving car" does. Its controller must gather *multiple* inputs and react correctly, in near real time and provide adaptive change to critically co-dependent entities (e.g. steering, brakes, throttle).

Ansible is not at the heart of the self-driving car (we hope), but it is a good fit for creating and deleting complete "systems" (or parts of systems) like network, compute, and storage which all have interdependencies. Orchestration of these entities makes sense because the discrete components (combined) create a system. Deployed in isolation they each provide (almost) no function and require subsequent "manual" orchestration to become an actual "system". Taking "manual" siloes out therefore creates opportunities for the value-add of accuracy, efficiency and de-risking repeat orchestrations.

Ansible has some advantages over competitors like Puppet and Chef:

- it functions **without the complexity and dependencies** of the requirement to deploy and maintain agents on target devices
- its deployment is simple and highly scalable, easily distributed
- it is a **relatively simple thing to learn**. Compared to learning a true coding language, it is more like learning a dialect of your own language than coding, which is more like learning a whole new foreign language.

Ansible is well suited to IT infrastructure professionals. They understand the use case for automation and the detail of the desired outcomes that are wanted on the products that they specialize in. Creating an orchestration with Ansible across product lines to produce a full "system" is entirely possible as the sequence of requirements can be established between the owners of each discipline. If cross-disciplinary repetitive combined efforts can be identified, Ansible is likely a good fit to automate them.

The **simplicity of Ansible** lies in the use of declarative, **idempotent (when done right)** playbooks to provide an abstraction from the complexity of the underlying code that pushes the desired commands to the target systems. The playbooks are simple YAML files that can be readily understood; these call python modules that translate the configuration attributes from the YAML into the required API calls to REST or SSH/CLI on the target systems.

A logical learning path toward effective Ansible utility is to start with small automations - like individual plays – and gradually working up to more sophisticated playbooks that sequence more actions to meet actual business-oriented needs.

A great example of a business need is the administrative automation of the environment that you utilize in the enclosed labs. Our playbooks create the environment "day of", and with the press of a button the environment is fully enabled (and deleted afterward) without any specialist technical knowledge required by the operator. Our Ansible playbooks orchestrate

Microsoft, Cisco, Ansible Tower, and CentOS to make this day possible, and the result is identical every time.

- Sufficient VDI terminals are enabled to facilitate the class
- VDI credentials are created, allowing VDI login
- Access information is automatically forwarded to proctors via email for distribution to students
- Credentialed ACI tenants are created and secured
- Virtual machines are created for testing traffic flows in each tenant
- Credentialed, secure Ansible Tower access is created
- At the end of the session all created resources are deleted

Other very practical use cases for massive de-risking and efficiency gains include:

- Migrations
    - older to newer
        - e.g. legacy DC aggregation over to ACI
    - on-prem to cloud
        - or to hybrid model
- Validations
    - Query and compare with a benchmark e.g. for compliance
- Refresh/ return to an earlier known state
    - Orchestrate across compute/storage/networking
- Any repetitive mass changes that are slow or cumbersome in GUI

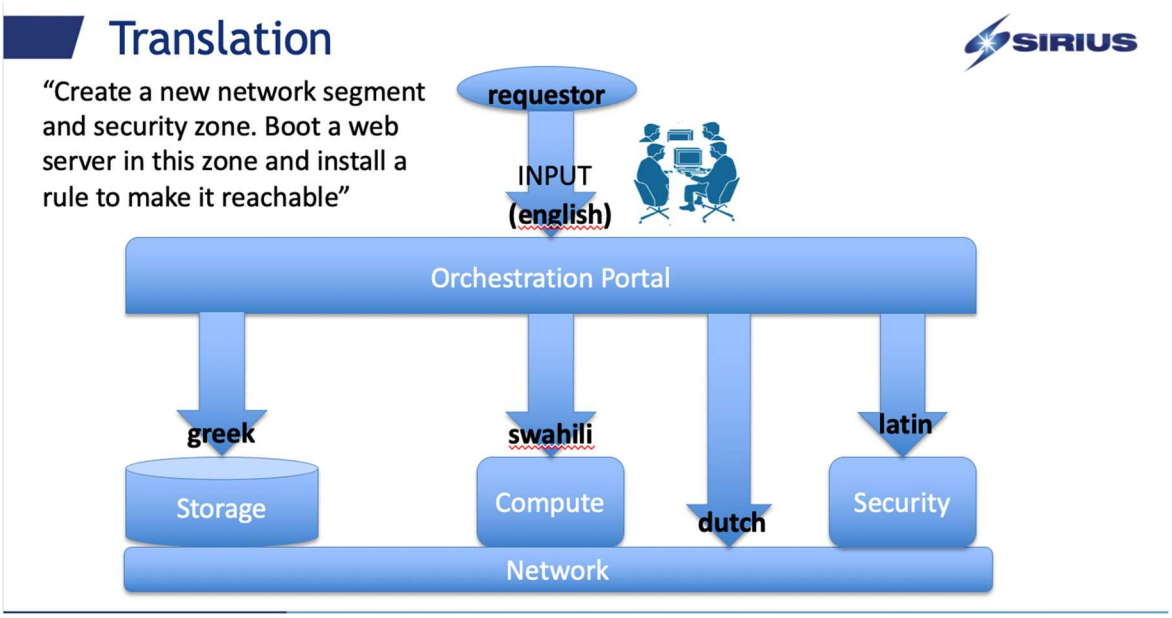
Ansible Tower

Ansible lacks native controls around governance items like authorization (who can do what). It also has only a command line interface. Ansible tower brings a more user-friendly GUI experience and provides role-based-access rights. As such, Ansible playbooks (which are still the engine behind Tower) become subject to structured authorization. Target architectures can now be split into very granular permissions, such that only specifically assigned capabilities will be visible and executable by a user who logs in, based on their assigned rights. This granularity may be mandatory in environments where regulatory compliance is a focus.

Tower also introduces workflow **templates** that allows events or plays to be sequenced in a drag/drop graphical manner. The end user who "kicks off" a play does not necessarily see the playbook itself, but they may be entering string variables, making dropdown selections, or pressing radio buttons in a GUI window to populate it. Thus, Ansible Tower is providing additional abstraction from the underlying technology and **furthering an "intent based" approach** to networking.

Tower does have an API and so can itself be programmatically controlled by higher layer orchestration systems to create and/or kick off workflows. This is especially powerful as an **integration point for example with ITSM and CMDB tools like ServiceNow and Ivanti Heat.**

Figure 7: Ansible Tower: Orchestration to reduce complexity and co-ordinate change:



LABS: Getting Started

1. Connect to your VDI terminal using the https://view.siriussdx.com URL and your provided credentials
2. Wait for approximately one minute and Launch your docker container with the provided desktop shortcut “docker container” once it has completed initializing.
3. Briefly, explore your container directories:
    - Launch the “student container” from the icon
        - Ignore/close the filesharing blurb that pops up
    - cd dev/aci/lab1
    - “ls -a” to view the list playbooks and var files
4. Examine this ansible playbook closely (or just view at Figure 8 below):
    - cat student_id.yml

Figure 8: student_id.yml file

```
---
- hosts: localhost
  connection: local
  gather_facts: false
  vars_prompt:
  - name: student_id
    prompt: "What is your student ID?"
    private: no


  tasks:
  - name: Add correct student number to VAR file
    lineinfile:
      path: ~/dev/aci/lab1/immday_vars.yml
      regexp: sno
      line: 'student_no: {{student_id}}'
```

5. you will run this playbook to input your student ID Number ahead of lab1
6. what does it look like it is meant to do?
7. cat the “target” file and review what this playbook will alter in it
    - Do not alter the file
8. RUN the student_id.yml playbook from the container command line:
    - Command is: ansible-playbook student_id.yml
9. Observe the playbook feedback

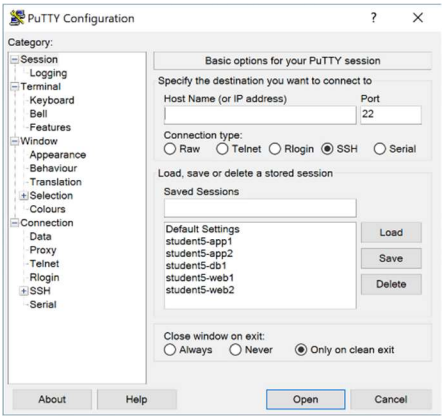
Figure 9: live playbook feedback

```
[root@cent-mgmt-ansible01 ~]# ansible-playbook Sidlab-ansibleACI/ansible-aci/Lab1/student_id.yml
[WARNING]: Invalid characters were found in group names but not replaced, use -vvvv to see details
What is your student ID?: 12345

PLAY [localhost] ***************************************************************************************************
******

TASK [Add correct student number to VAR file] **********************************************************************
******
changed: [localhost]

PLAY RECAP *********************************************************************************************************
******
localhost                  : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

10. Cat the immday_vars.yml file again
    - What has changed?

11. Access your APIC and Ansible Tower GUIs via the provided Chrome desktop shortcuts
    - Log in to the APIC using the provided credentials and examine your tenant configuration (which is initially empty).
    - Log into Ansible Tower using the provided credentials and review the GUI
12. Launch *Putty* and note that shortcuts are present for SSH to your five test vms
    - Lab 1 will provide reachability to the test vms. Username and password on all the vms is student/mysidlab
    - Fig 10 shows the vm DNS names that you can also use from your VDI "cmd" command line
    - Fig 11 also reflects the test vm IP addresses

Figure 10: putty sessions diagram



Labs: Premise Statements

Each lab utilizes the same basic topology components:

1) A student tenant which hosts the test virtual machines within a classic "3-tier" ACI application profile comprising three security zones (EPGs): Web, App, and DB. The lab student can administer this tenant and its logical components
2) An administrative tenant hosts the student VDI terminals, docker container, and Ansible Tower. Students have no administrative rights or visibility of this tenant but it is included in topology diagrams for clarity.

Lab 1 covers the student tenant buildout introducing Ansible as the automation engine, but stops short of securing the zones.

Lab 2 deploys the same environment and then layers security functionality over it, achieving the entire deployment using Ansible Tower job templates and workflow templates. The student creates their own Tower project, and builds and runs the templates under proctor guidance.

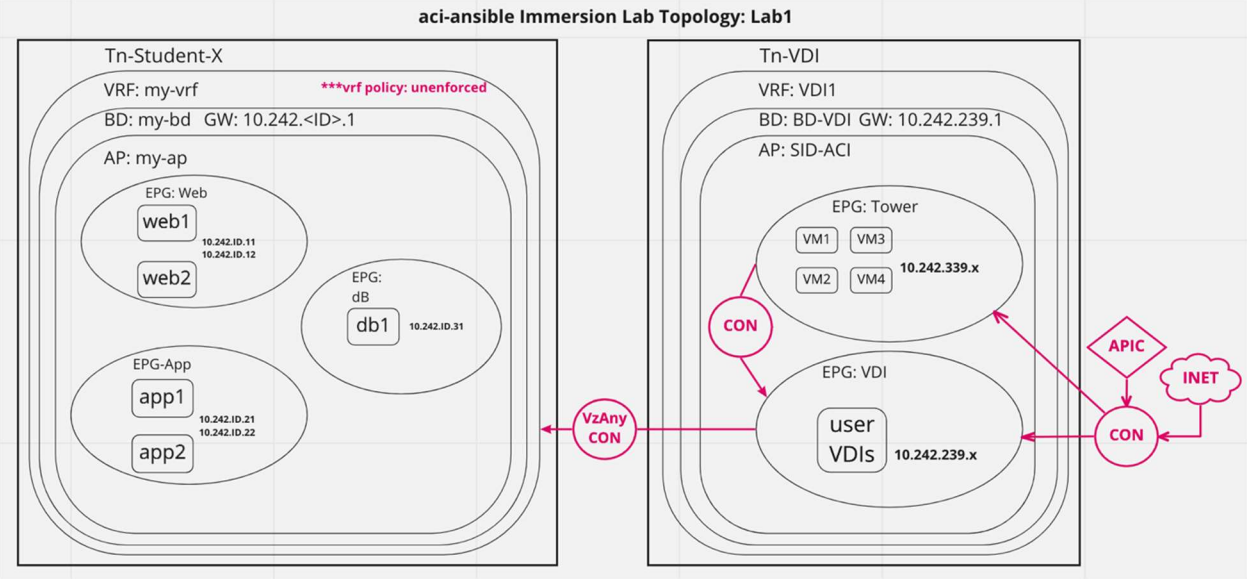
Lab 1:

Please review the topology diagram at Figure 11, below. At the conclusion of lab 1, you will be able to ping all vm's in your student tenant from your jump box. Additionally, all vm's in your student tenant will be able to ping each other.

Explanation:

- VRF policy enforcement is "disabled" in your VRF
- "consumed contract interface" using "VzAny" is attached to VRF allowing jumpbox access

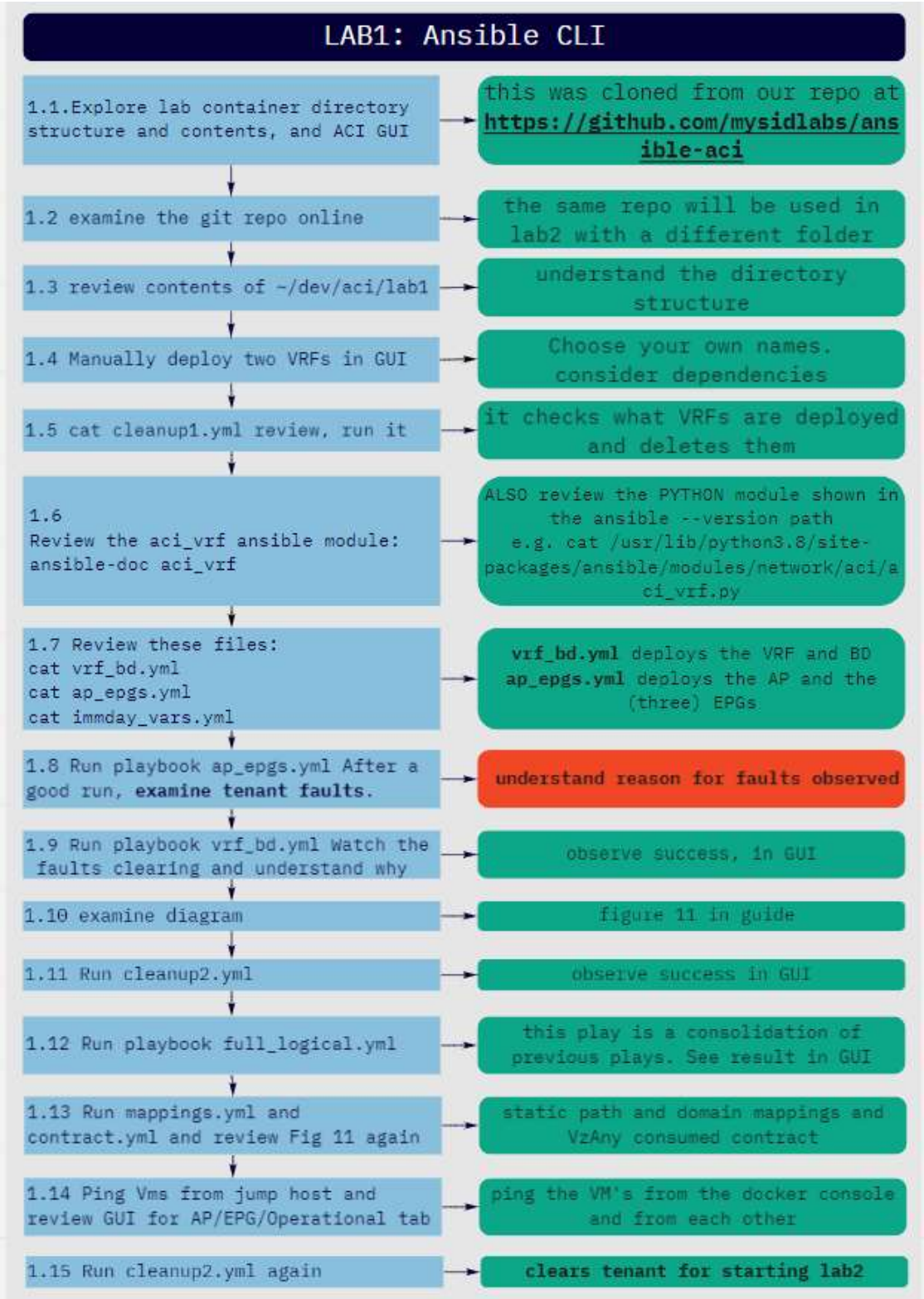
Figure 11: Lab 1 Topology Diagram



Notes:

- When VRF policy enforcement is disabled, intra-VRF traffic flows are allowed without a contract, but a regardless of enforcement status, a contract is *always mandatory* for flows to be allowed *inter-tenant* or *inter-VRF*.
- Web, App, and dB EPG's are "static-pathed" into the physical environment where the vms have been pre-attached to the required VLANs on standard vswitches. This is the same methodology by which "baremetal" servers are typically attached to ACI. The static pathing configurations are deployed in mappings.yml and are visible in the GUI under EPG/domain and EPG/static ports
- VMM integration is a much stronger ACI integration for virtual machines but it is outside of the scope of this lab
- The VzAny contract is a normal contract but applied in a specific way under the VRF instead of the EPG. It attaches all EPG to the contract without having to do this piecemeal and therefore it is useful for things like shared services (DNS, DHCP, etc.). It also economizes on the use of TCAM resources.

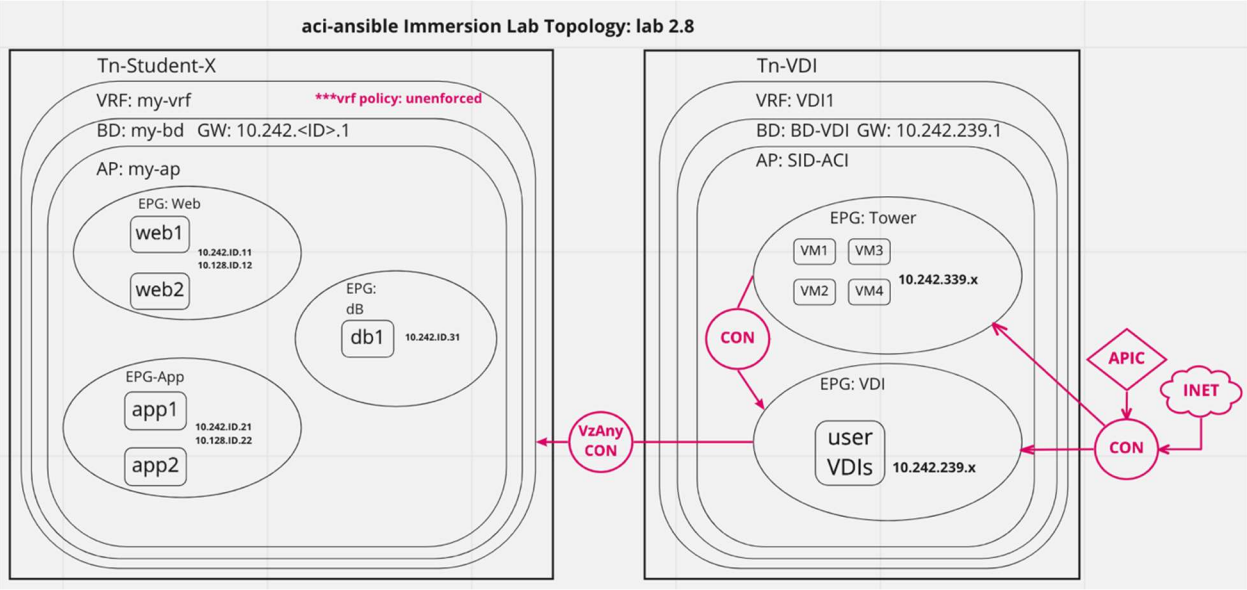
Figure 12: Lab 1 Instructions



Lab 2

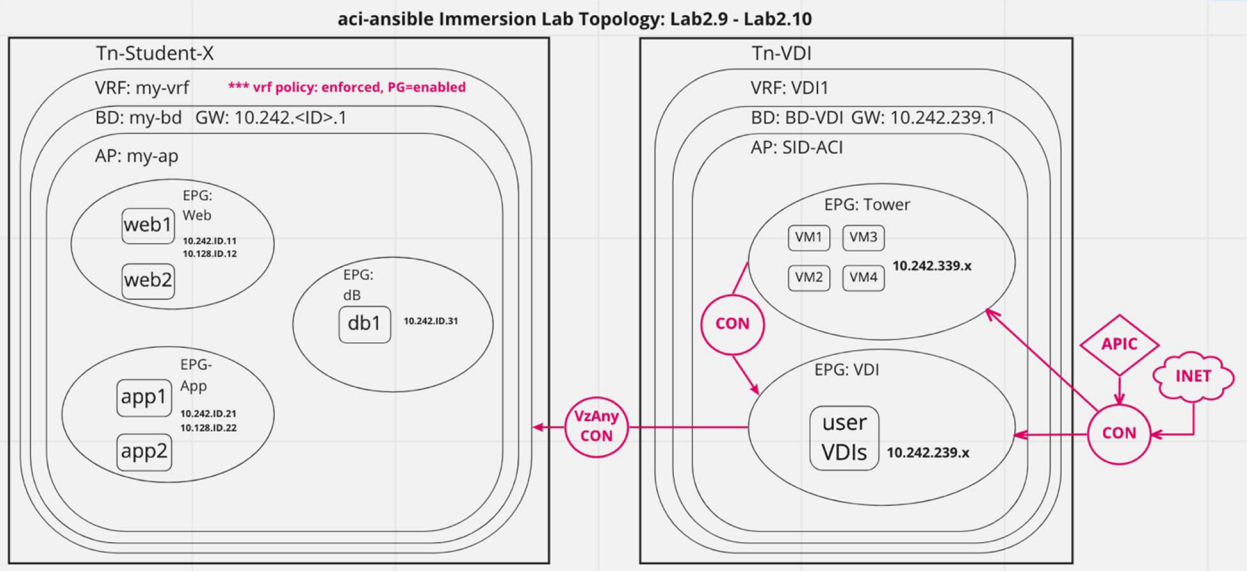
Please review the topology diagrams in Figures 13 to 17 below. These portray how Lab 2 incrementally introduces additional ACI security functionality to better secure the application profile created in Lab 1, but it introduces Ansible Tower to facilitate the changes. The instructions for Lab2 are posted after the diagrams, in Figure 18.

Figure 13: Lab 2.8 Topology Diagram



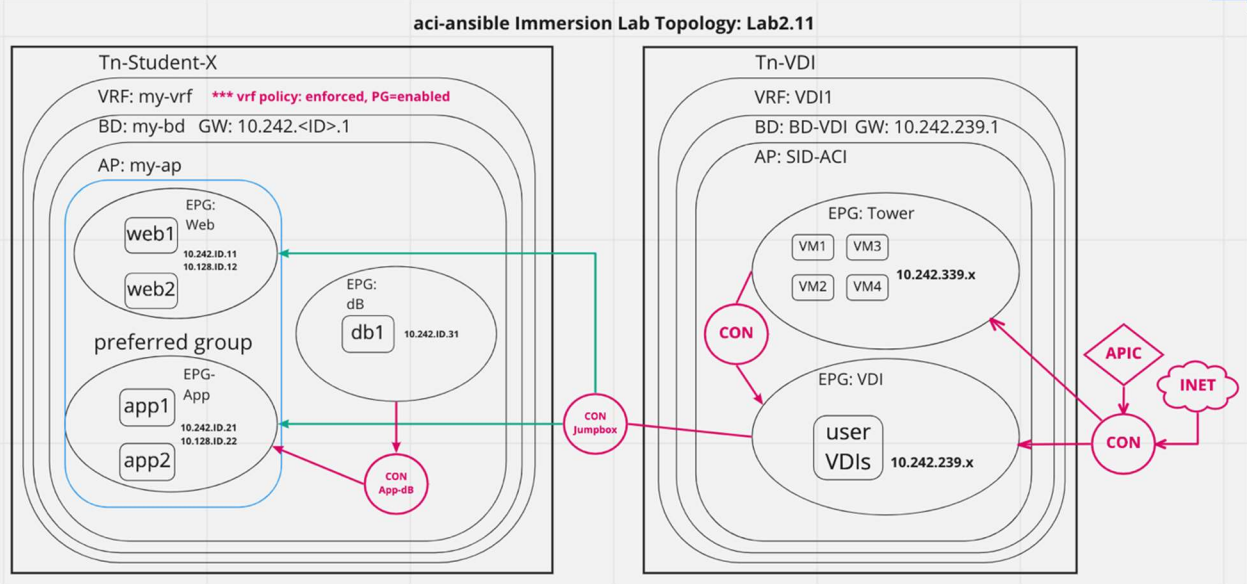
- 2.8 Includes the steps required to initialize the environment such that your test vm's can all reach each other and the jump host. This is the same state of your ACI deployment as it was at the end of Lab1, but this time Ansible Tower performed the orchestration using sequentially run job templates.
    - Explanation: VRF enforcement is disabled. This allows communication between all the vms within the VRF, regardless of their EPG membership. A "consumed contract interface" has been applied to the entire VRF to allow jumphost reachability (from an external tenant) to all vms.

Figure 14: Lab 2.9 – 2.10 Topology Diagram



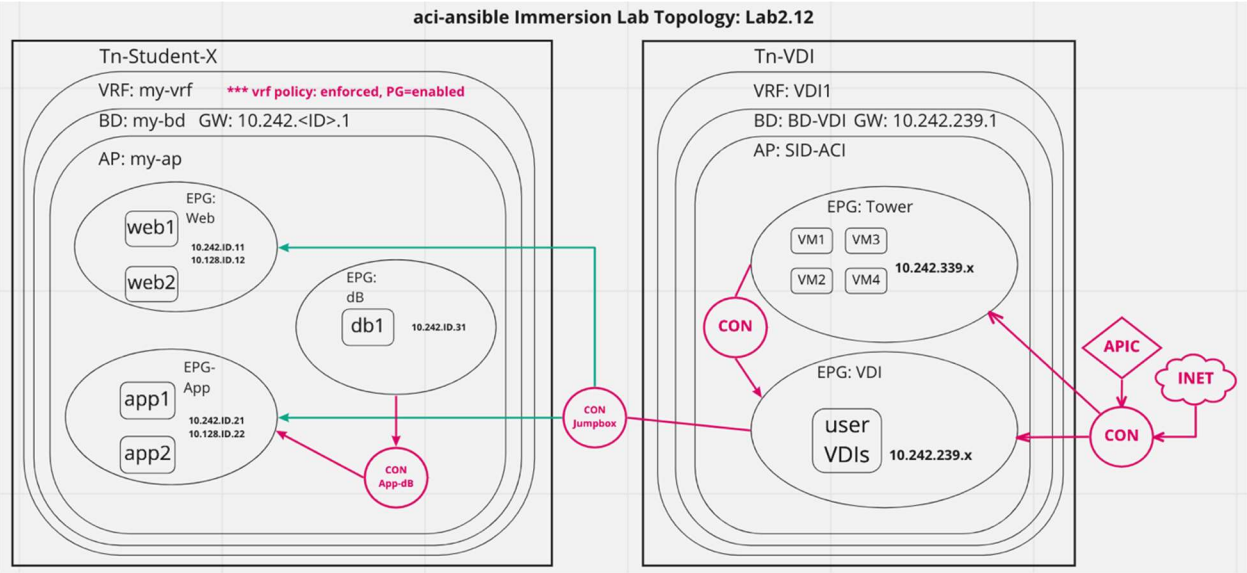
- 2.10, vm's can reach the jump host but not each other.
  - Explanation: VRF enforcement has been enabled.
  - "preferred group" has been enabled for the VRF, but it has not been applied to any EPG and so they will all require contracts to see each other, since VRF enforcement is enabled.
  - The vzAny contract is still in place allowing access from VDI to the vms

Figure 15: Lab 2.11 Topology Diagram



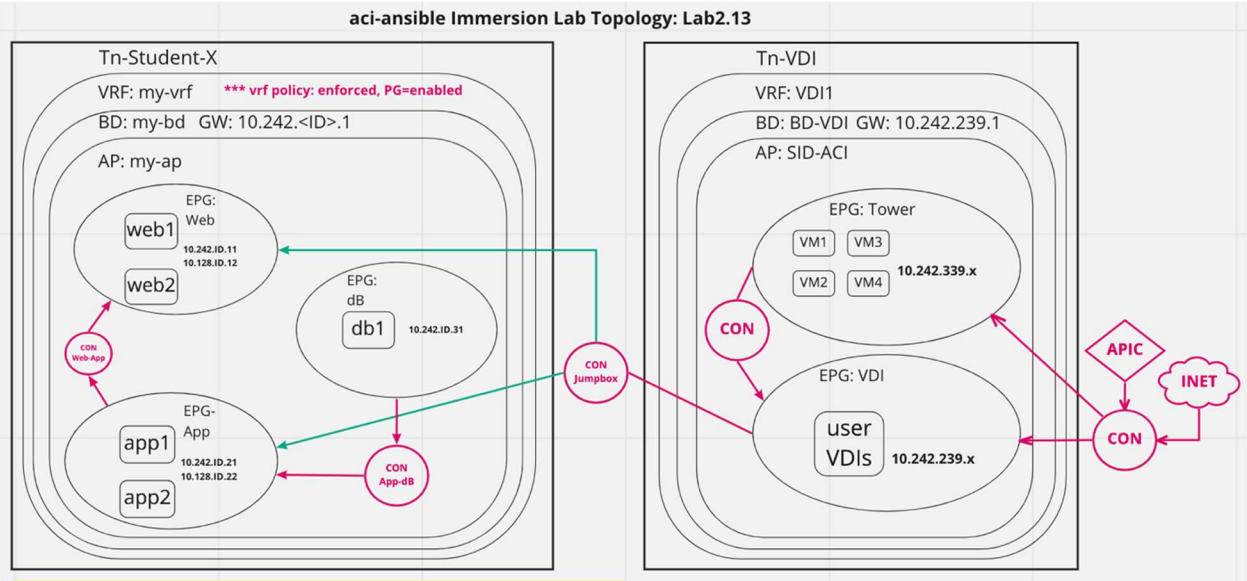
- Workflow enables preferred group for Web and App EPG and they can now reach each other
- Workflow adds App-DB contract for App-DB reachability
- Workflow ensures Jumpbox contract is consumed by Web and App only so DB is not reachable from VDI terminal

Figure 16: Lab 2.12 Topology Diagram



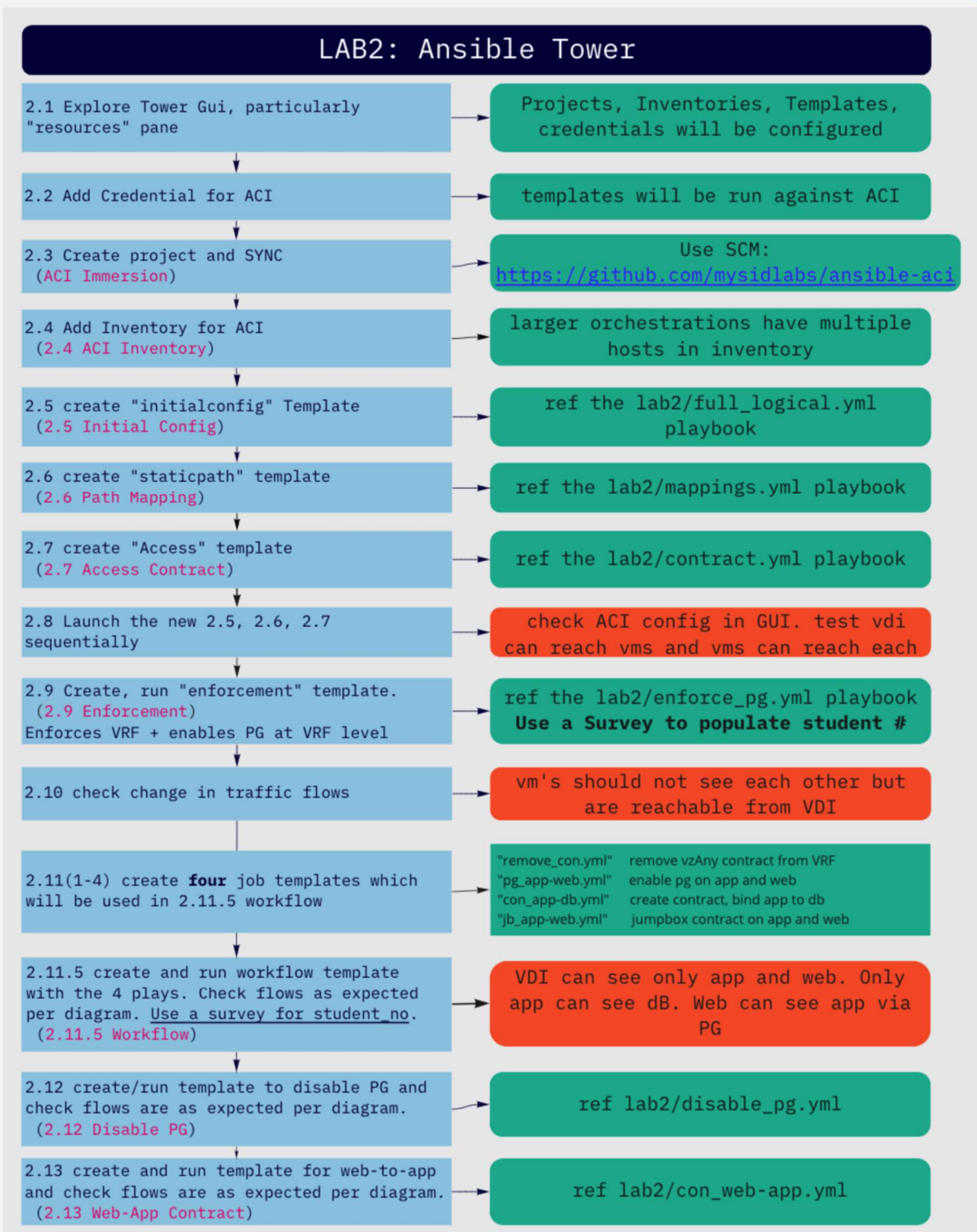
- 2.12 removes the PG functionality, which disables Web-to-App reachability

Figure 17: Lab 2.13 Topology Diagram



- 2.13 adds a Web-to-App contract, re-enabling Web-to-App reachability

Figure 18: Lab 2 Instructions



Useful resource links and information

Links:

Ansible Best Practices

https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html

Ansible Network troubleshooting

https://docs.ansible.com/ansible/latest/network/user_guide/network_debug_troubleshooting.html

Ansible cli command module information

https://www.ansible.com/blog/deep-dive-on-cli-command-for-network-automation

Variable precedence

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable

Additional Notes:

- Remember YAML is very sensitive to correct indentation
- **`Hostvars`** allow us to access meta-data about our inventory hosts.
- The use of an Ansible role is best practice when there is a well-defined scope with a high possibility of re-use.
- If you copy and paste text for a playbook you may get indentation issues. Ansible provides a simple syntax checker, try ansible-playbook --syntax-check backup.yml to verify. A Best Practice is to use a linter, for example ansible-review. Ansible provides excellent online documentation, which is also available from the command line, for example ansible-doc ios_config. For a full list of modules try ansible-doc –l
- There where multiple ways of implementing a playbook where specific tasks or groups of tasks execute against specific hosts. For example, we could have used 1 playbook for configuring every router in the lab utilizing the “when:” statement to ensure specific tasks are only applied to a specific router. Although this is not necessarily following best practices.
- The use of `handlers:` which can be used in any playbook. A handler is a special way of calling a task whenever an action needs to be taken after a previous task. For example, both installing and configuring an application may require a restart. A handler would be notified by both tasks but would only run once when the playbook finishes.