



MODULE 7: INFRASTRUCTURE AND AUTOMATION

DevNet Associate v1.0





Module Objectives

- Module Title: Infrastructure and Automation
- Module Objective: Compare software testing and deployment methods in automation and simulation environments.
- It will comprise of the following sections:

| Topic Title | Topic Objective |
|--|--|
| 7.1 Automating Infrastructure with Cisco | Describe deployment environments that benefit from automation. |
| 7.2 DevOps and SRE | Explain the principles of DevOps. |
| 7.3 Basic Automation Scripting | Describe the use of scripting in automation. |
| 7.4 Automation Tools | Explain automation tools that speed the development and deployment of code. |
| 7.5 Infrastructure as Code | Explain the benefits of storing infrastructure as code. |
| 7.6 Automating Testing | Explain how automation tools are used in the testing of application deployments. |
| 7.7 Network Simulation | Describe the use of the Cisco VIRL network simulation test environment. |



7.1 AUTOMATING INFRASTRUCTURE WITH CISCO





Introduction to Automating Infrastructure

- Automation is using code to configure, deploy, and manage applications together with the compute, storage, and network infrastructures and services on which they run.
- For automation with Cisco infrastructure, the platforms can integrate with the common tools such as Ansible, Puppet, Chef and so on, or provide direct API access to the programmable infrastructure.

Cisco Automation Solutions

- There are several use cases for automation for the network. Depending on the operational model to be followed, there are choices to programmatically control the network configurations and infrastructure.

| Walk: Read only automation | Run: Activate policies and provide self-service across multiple domains | Fly: Deploy applications, network configurations, and more through CI/CD |
|---|--|---|
| <ul style="list-style-type: none"> Using automation tools, information can be gathered about the network configuration. The ability to reduce risk by implementing security policies across the network infrastructure via automated changes. In the Automation Exchange, this shift is categorized as a walk-run-fly progression. | <ul style="list-style-type: none"> With these Run stage automation scenarios, the users can safely provision their own network updates. On-boarding workflows can be automated, day-to-day network configurations can be managed, and daily scenarios can be pervaded. | <ul style="list-style-type: none"> For more complex automation and programmable examples, the Fly stage of the DevNet Automation Exchange is used. Here the needs can be managed by monitoring and proactively managing the users and devices, while also gaining insights with telemetry data. |

- Read-only automation can gather SD-WAN inventory data using Python and recording the values in a database.



Why Do We Need Automation?

- Speed and agility enable the business to explore, experiment with, and exploit opportunities ahead of their competition.
- Developers need to accelerate every phase of software building: coding, iterating, testing, and staging. DevOps practices require developers to deploy and manage apps in production, so developers should also automate those activities.

Disadvantages of Manual Operations

- Add to costs, are time taking and are hard to scale.
- Are prone to human error, and documentation meant for humans is often incomplete and ambiguous, hard to test, and quickly outdated.

Why Do We Need Automation?

▪ Dependency risks

- Today's software ecosystem is decentralized. Developers build individual components according to their needs and interests and mix and match components, infrastructure, and services needed to enable complete solutions and operate them efficiently at scale.
- This ecosystem introduces new requirements and new risks:
 - Components need to be able to work alongside many other components in many different situations (this is known as being **flexibly configurable**) showing no more preference for specific companion components or architectures than absolutely necessary (this is known as being **unopinionated**).
 - Component developers may abandon support for obsolete features and rarely-encountered integrations. This disrupts processes that depend on those features.
 - Dependency-ridden application setups tend to get locked into fragile and increasingly insecure deployment stacks.



Why Do We Need Full - Stack Automation?

- Automation is a key component of functional software-defined infrastructure and distributed and dynamic applications. The benefits of full-stack automation are:
 - **Self-service:** Automation provides self-service frameworks which enable users to requisition infrastructure on demand.
 - **Scale on demand:** Apps and platforms need to be able to scale up and down in response to traffic and workload requirements and to use heterogeneous capacity.
 - **Observability:** An observable system enables users to infer the internal state of a complex system from its outputs.
 - **Automated problem mitigation:** The apps and platforms should be engineered to minimize the effects of issues, self-heal and monitor events.



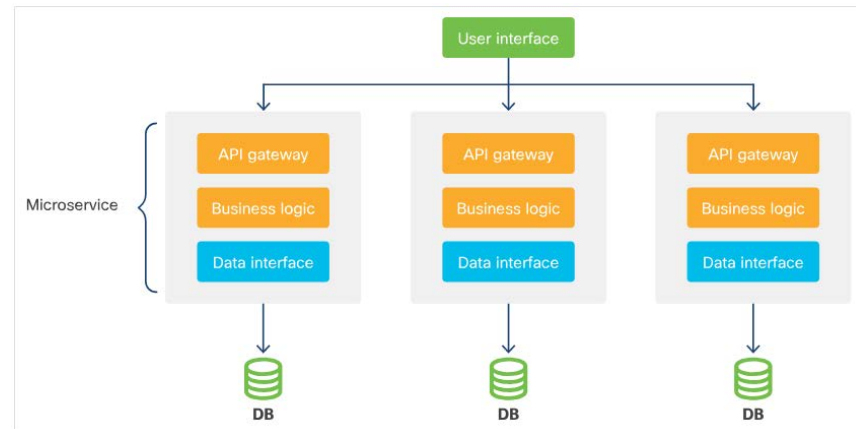
Software-Defined Infrastructure: A Case for Automation

- Software-defined infrastructure is also known as **cloud computing**. It lets the developers and operators to use the software to requisition, configure, deploy and manage bare-metal and virtualized compute, storage and network resources.

| | |
|--|---|
| Benefits of cloud paradigms | <ul style="list-style-type: none">• Self-service (platforms on demand) - automated tools and methods that deliver preconfigured developer platforms on demand.• Close specification, consistency, repeatability• Platform abstraction - allows a containerized app can run on a generically-specified host environment. |
| Challenges faced in cloud paradigms | <ul style="list-style-type: none">• Developers must pay close attention to platform design, architecture, and security.• Access control is critical as cloud users with the wrong permissions can do a lot of damage to their organization's assets.• When cloud resources can be self-served quickly via manual operations, consumption can be hard to manage, and costs are difficult to calculate. |

Distributed and Dynamic Applications: Another Case for Automation

- Modern application architectures are increasingly distributed.
- They are built up of small and relatively light components that are sometimes called microservices.
- These components may be isolated in containers, connected via discovery and messaging services (which abstract network connectivity) and backed by resilient, scalable databases (which maintain state).





Distributed and Dynamic Applications: Another Case for Automation

Benefits of microservices:

- **Scalability** - Microservices can be scaled and load-balanced as needed across many networked servers or multiple geographically-separate public cloud regions. This eliminates single points of failure.
- **Infrastructure Automation tools** - Increasingly, the dynamism of microservice-based applications is provided by infrastructure. These container automate on-demand scaling, self-healing, and more.

Challenges of microservices:

- **Increased complexity** - Microservices mean that there are many moving parts to configure and deploy. There are more demanding operations, including scaling-on-demand, self-healing and other features.
- **Automation is a requirement** - Manual methods can not realistically cope with the complexity of deploying and managing dynamic applications.



Automating Infrastructure Summary

- These business and technical needs, trends, and dynamics, encourage developers and operators to use automation everywhere for the following tasks:
 - Manage all phases of app building, configuration, deployment and lifecycle management. This includes coding, testing, staging, and production.
 - Manage software-defined infrastructures on behalf of the applications you build.
 - Alongside the applications, to preserve, update, and continually improve the automation code. This code helps to develop, test, stage, monitor, and operate the apps at production scales, and in various environments. Increasingly, all this code can be treated as one work-product.



7.2 DEVOPS AND SRE

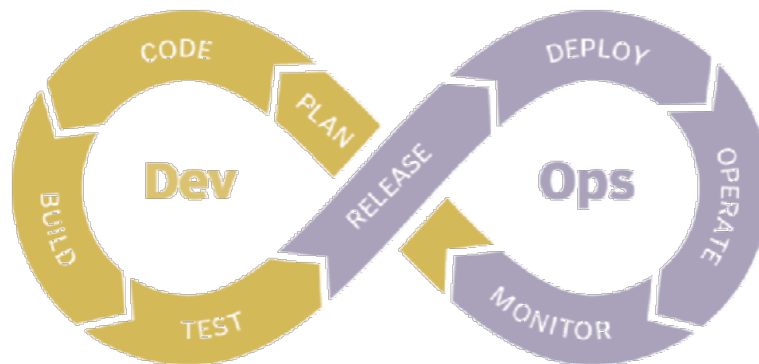




Introduction to DevOps and SRE

- For full-stack automation to be truly effective, it requires changes to organizational culture, including breaking down the historical divides between Development (Dev) and Operations (Ops).
- Historically, creating applications was the job of software developers (Dev), and ensuring that apps work for users and the business has been the specialized province of IT operations (Ops).

DevOps infinity loop





DevOps Divide

- The following table describes the different characteristics of Dev and Ops:

| Characteristics | Dev | Ops |
|----------------------------------|---|--|
| Cares about | Bespoke applications and how they work | Applications and how they run, plus Infrastructure, OS, network, and so on |
| Business treats as | Profit center: demands resources | Cost center: provides and accounts for resources |
| Participates in on-call rotation | Occasionally (only when issues are escalated to dev) | Regularly (point of spear) |
| Performance measured | Abstractly (including bad metrics) | Concretely (SLA compliance, issues resolved) |
| Skills required | More deep than broad: Languages, APIs, architecture, tools, process and so on | More broad than deep: Configuration, administration, OS, automation, and so on |
| Agility required | Move fast, innovate, break things, fix later | Investments must be extensively justified, expectations managed |



DevOps Divide

- In the traditional, pre-virtualization, enterprise IT ecosystem, separating Dev from Ops seemed sensible.
- In the early 2000s, there began a movement to treat Dev and Ops as a single entity:
 - Make coders responsible for deployment and maintenance.
 - Treat virtualized infrastructure as code.



Evolution of DevOps

- DevOps evolved and continues to evolve in many places in parallel. Some key events have shaped the discipline as we know it today.
 - **Site Reliability Engineering (SRE):** Institutionalization of SRE by Google in 2003. Created with the intent of combining the disciplines and skills of Dev and Ops.
 - **Debois and Agile Infrastructure:** Patrick Debois' presentation in 2009 on automating virtual and physical infrastructure using version control and applying Agile methods.
 - **Allspaw and Hammond:** Presentation by John Allspaw and Paul Hammond in 2009 outlining a simple set of DevOps best practices founded on the idea that both Dev and Ops cooperatively enable the business.



Core Principles of DevOps

- DevOps/SRE have many core principles and best practices:
 - A focus on automation
 - The idea that "failure is normal"
 - A reframing of "availability" in terms of what a business can tolerate

Core Principles of DevOps

▪ SLOs, SLIs, and error budgets

- The two linked ideas to DevOps/SRE culture are DevOps must deliver measurable, agreed-upon business value and the statistical reality of doing so perfectly is impossible.
- These ideas are codified in a Service Level Objective (SLO) that is defined in terms of real metrics called Service Level Indicators (SLIs). An SLI is a benchmark metric used to determine and describe SLOs.
- SLIs map to the practical reality of delivering a service to customers.
- SLO/SLI methodology permits cheaper, more rapid delivery of business value by removing the obligation to seek perfection. It can also influence the pace, scope, and other aspects of development to ensure and improve adequacy.
- One way of modeling SLO/SLI results requires establishing an **error budget** for a service for a given period of time and then subtracting failures to achieve SLO from this value.
 - An error budget is the difference between the SLO and 100% availability.



DevOps and SRE Summary

- DevOps/SRE is co-evolving with technologies like virtualization and containerization, enabling a unified approach and unified tool set to support coordinated application and infrastructure engineering.



7.3 BASIC AUTOMATION SCRIPTING





Introduction to Basic Automation Scripting

- Powerful automation tools like Ansible, Puppet, and Chef bring ease of use, predictability, discipline and the ability to work at scale to DevOps work.
- Although automation tooling partly works by wrapping shell functionality, operating system utilities, API functions and other control plane elements for simplicity, uniformity, feature enrichment, and compatibility in DevOps scenarios, these tools still do not solve every problem of deployment and configuration.
- Every automation tool has one or more functions that execute basic commands and scripts on targets and return results.
- It is therefore important to have basic automation scripting skills.

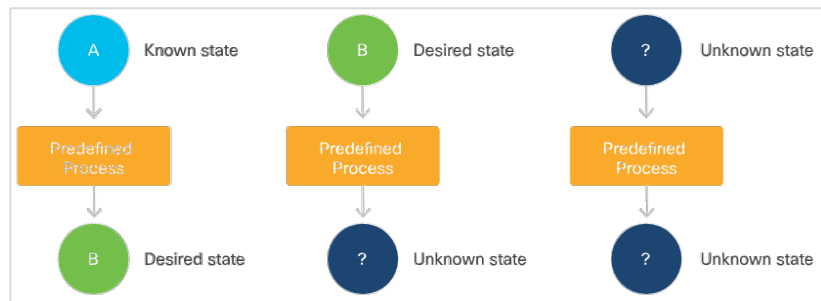


Basic Tools for Automation Scripting

- Shells are ubiquitous, so shell scripting is historically the bedrock of automation.
- **Bash**
 - The Bash is a Unix shell and is a default on most Linux distributions and on macOS. Using commands in a Bash script is much the same as using them directly from the command line.
 - Bash can be used to script access to the AWS CLI, you can use the built-in features and libraries of more sophisticated languages to parse complex returned datasets (such as JSON), manage many parallel operations, process errors, handle asynchronous responses to commands, and more.
- **Programming languages beyond Bash**
 - Sophisticated languages improve on Bash when complexity and scale requirements increase. They are useful when building and operating virtualized infrastructure in cloud environments, using SDKs.

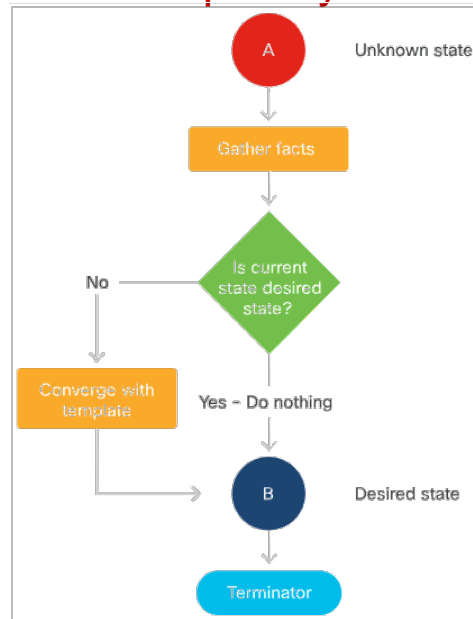
Procedural Automation

- An **imperative procedure** is an ordered sequence of commands aimed at reaching a specific end state. The sequence may include flow-control, conditions, functional structure, classes, and more.
- To ensure efficiency and reusability of scripts one can:
 - Standardize the ordering and presentation of parameters, flags, and errors.
 - Create a code hierarchy that divides tasks logically and efficiently.
 - Create high-level scripts for entire deployments and lower-level scripts for deployment phases.
 - Separate deployment-specific data from the code, making the code as generic and reusable as possible.



Procedural Automation

- Carefully-written procedural scripts and declarative configuration tools examine targets before performing tasks on them, and only perform the tasks needed to achieve the desired state.
- This type of scripting tends to be dangerous if starting state is not completely known and controlled.
- This quality of software is called **idempotency**.
- Basic principles of idempotency are:
 - Ensure the change you want to make has not already been made
 - Get to a known-good state, if possible, before making changes
 - Test for idempotency
 - All components of a procedure must be idempotent



Executing Scripts Locally and Remotely

To configure remote systems, the user need to access and execute scripts on them. There are many ways to do this:

- Store scripts locally, transmit them to target machines with a shell utility like `scp`, then log into the remote machine using `ssh` and execute them.
- Pipe scripts to a remote machine using `cat | ssh` and execute them in sequence with other commands, capturing and returning results to the terminal, all in one command.
- Install a general-purpose secure file-transfer client like SFTP, then use that utility to connect to the remote machine, transfer, set appropriate permissions, then execute the script file.
- Store scripts on a webserver, log into the remote machine and retrieve them with `wget`, `curl`, or other utilities, or store the scripts in a Git repository.
- Install a full remote-operations solution like VNC or NoMachine locally, install its server on the target, transmit/copy and then execute scripts.
- If the target devices are provisioned on a cloud framework, there is usually a way to inject a configuration script via the same CLI command or WebUI action that manifests the platform.



Cloud Automation

- Cloud automation enables the user to provision virtualized hosts, configure virtual networks and other connectivity, requisition services, and then deploy applications on this infrastructure.
- Cloud providers and open source communities often provide specialized subsystems for popular deployment tools, which extract a complete inventory of resources from a cloud framework and keep it updated in real time while automation makes changes.
- The user can also manage cloud resources using scripts written in Bash, Python, or other languages.



Cloud CLIs and SDKs

- IaaS and other types of infrastructure cloud also provide CLIs and SDKs that enable easy connection to their underlying interfaces, which are usually REST-based.

| | |
|--------------------------------|---|
| Cisco UCS - a bare metal cloud | <ul style="list-style-type: none"> • Cisco Intersight RESTful API • Range of SDKs for the Intersight RESTful API, including ones for Python and Microsoft PowerShell • Range of Ansible modules |
| VMWare | <ul style="list-style-type: none"> • Datacenter CLI • vSphere CLI for Linux and Windows • PowerCLI for Windows PowerShell • Host of SDKs for popular languages, aimed at vSphere Automation, vCloud Suite, and other products |
| OpenStack | <ul style="list-style-type: none"> • OpenStack Client (OSC) • OpenStack Compute, Identity, Image, Object Storage, and Block Storage APIs • OpenStack Python SDK • OpenStack Toolkits |



Summary of Basic Automation Scripting

- Basic automation scripting techniques are great to have in the toolbox and understanding them will improve the facility as an operator and user of mature automation platforms.



7.4 AUTOMATION TOOLS





Introduction to Automation Tools

- In this topic, the three most popular automation tools, Ansible, Puppet, and Chef, are being discussed.
- There will also be an option to install one or all of them on the local workstation.
- To try this, one must have access to a Linux-based workstation, such as Ubuntu or macOS and must refer to the tool's own installation documentation for the operating system.



What Do Automation Tools Do For Us?

- What do automation tools do for us?
 - Automation tools offer powerful capabilities compared to ad-hoc automation strategies using BASH, Python, or other programming languages. These tools enable developers to:
 - Simplify and standardize
 - Accelerate development with out-of-the-box features
 - Facilitate reusability, segregate concerns, promote security
 - Perform discovery and manage inventory
 - Handle scale
 - Engage community



Critical Concepts

▪ Idempotency: a review

- An Idempotent software produces the same desirable result each time that it is run.
- In a deployment software, Idempotency enables convergence and composability and allows to:
 - More easily gather components in collections that build new kinds of infrastructure and perform new operations tasks.
 - Execute whole build/deploy collections to safely repair small problems with infrastructure, perform incremental upgrades, modify configuration, or manage scaling.



Critical Concepts

▪ Procedure vs. Declarative

- Procedural code can achieve idempotency, but many infrastructure management, deployment, and orchestration tools have adopted another method, which is creating a declarative.
- A declarative is a static model and is used by middleware that incorporates deployment-specific details, examines present circumstances, and brings real infrastructure into alignment with the model, and usually least time-consuming path.

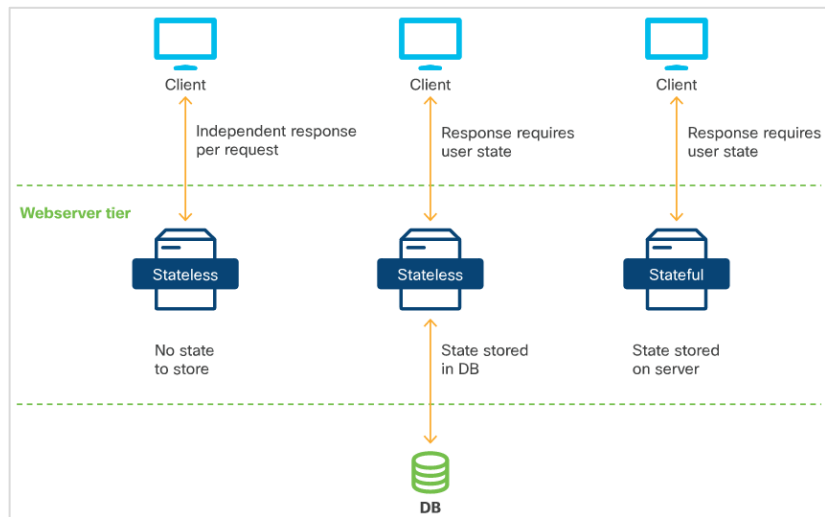
Critical Concepts

▪ Provisioning vs. configuration vs. deployment vs. orchestration

| Provisioning | Configuration | Deployment | Orchestration |
|--|--|---|---|
| <p>This refers to obtaining compute, storage, and network infrastructure (real or virtual), enabling communications, putting it into service, and making it ready for use by operators and developers.</p> | <p>This means installing base applications and services, and performing the operations, tasks, and tests required to prepare a low-level platform to deploy applications or a higher-level platform.</p> | <p>This involves building, arranging, integrating, and preparing multi-component applications or higher-level platforms, often across multiple nodes.</p> | <p>This may refer to several things:</p> <ul style="list-style-type: none">• User-built or platform-inherent automation aimed at managing workload lifecycles and reacting dynamically to changing conditions.• Processes or workflows that link automation tasks to deliver business benefits, like self-service. |

Critical Concepts

- **Statelessness**
- Automation works best when applications can be made stateless. This means that redeploying them in place does not destroy or lose track of the data that users or operators need.
- The two states of an application are:
 - **Not Stateless** – An app that saves important information in files or in a database on the local file.
 - **Stateless** – An app that persists its state to a separate database or that provides service that requires no memory of state between invocations.





Popular Automation Tools

- The first modern automation tool was Puppet which was introduced in 2005 as open source, and then commercialized as Puppet Enterprise by Puppet Labs in 2011.
- The most popular automation tools are Ansible, Puppet, Chef. They share the following characteristics:
 - Relatively easy to learn
 - Available in open source versions
 - Plugins and adapters enable them to directly or indirectly control many types of resources
- Many other solutions also exist. Private and public cloud providers often endorse their own tools for use on their platforms such as OpenStack's HEAT project, AWS' CloudFormation, SaltStack and Terraform.

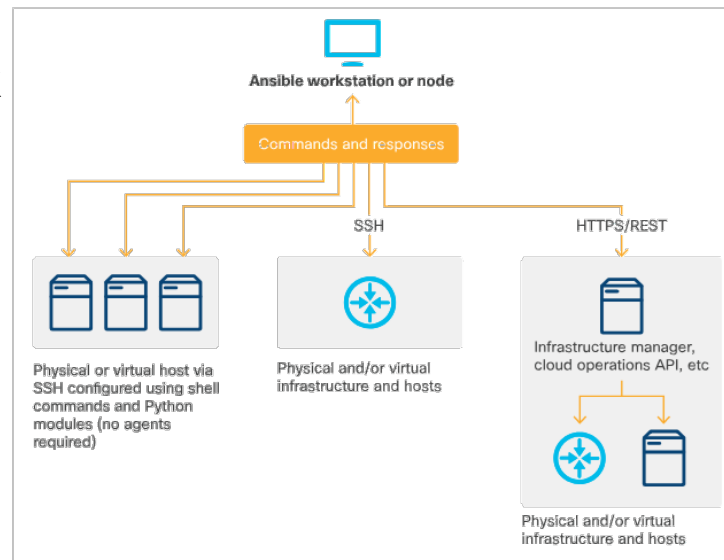


Ansible

- Ansible is available as open source, and in a version with added features, from IBM/Red Hat, called Ansible Tower.
- Ansible is substantially managed from the Bash command line, with automation code developed and maintained using any standard text editor.

Ansible

- Ansible Architecture: Simple and Lightweight
- Control node runs on any Linux machine running Python 2 or 3. All system updates are performed on control node.
- Control node connects to managed resources over SSH and enable Ansible to:
 - Run shell commands on a remote server, or transact with a remote router, or other network entity, via its REST interface.
 - Inject Python scripts into targets and remove them after they run.
 - Install Python on target machines if required.
- Plugins enable Ansible to gather facts from and perform operations on infrastructure that can't run Python locally.





Ansible

▪ Installing Ansible

- The Ansible control node application is installed on a Linux machine from its public package repository. To install Ansible on a workstation, refer to the installation documentation appropriate to the device.

▪ **Ansible code structure**

- In the Ansible code structure, work is separated into YAML (**.yml**) files that contain a sequence of tasks, executed in top-down order. Ansible has hundreds of pre-built Python modules that wrap operating-system-level functions and meta-functions.

▪ Playbooks and roles

- An Ansible playbook can be written as a monolithic document with a series of modular, named tasks.
- Developers build a model of a complex DevOps task out of low-level playbook task sequences called roles and then reference these in higher-level playbooks, sometimes adding additional tasks at the playbook level.
- This segregation of concerns ensures clarity, reusability and shareability of roles.

Ansible

- **Ansible project organization**

- Ansible projects are organized in a nested directory structure. The hierarchy is easily placed under version control and used for GitOps-style infrastructure as code.

- **Ansible folder hierarchy elements** include Inventory files, Variable files, Library and utility files, Main playbook files.

- **Ansible at scale**

- There are scaling challenges for large organizations, such as managing and controlling access to many Ansible nodes flexibly and securely. This also includes putting remote controllers seamlessly and safely under control of centralized enterprise automation.
- For this, there are two control-plane solutions: Red Hat Ansible Tower and AWX project.
- Larger-scale Ansible implementations also benefit from Ansible Vault, a built-in feature that enables encryption of passwords and other sensitive information.



Ansible

▪ Cisco Ansible resources

- Cisco and the Ansible community maintain extensive libraries of Ansible modules for automating Cisco compute and network hardware including:
 - A very large set of built-in modules for configuring Cisco Application-Centric Infrastructure fabrics via the Application Policy Infrastructure Controller (APIC).
 - Remote control of Cisco network devices running IOS-XR, plus modules for sending commands and retrieving results from these devices via CLI, or via the standard NETCONF REST interface.
 - Ansible modules for configuring Cisco UCS infrastructure via the Intersight REST interface.



Ansible Example

- **Ansible normally uses ssh to connect with remote hosts and execute commands.**
- Let's see how to create a simple website on a remote host.
- **Prerequisites**
 - A target host running a compatible operating system (such as Ubuntu 18.04 server)
 - **SSH and keywise authentication configured on that host**
 - Ansible installed on your local workstation

Ansible Example

- Building an Ansible project file tree
- For the purposes of this exercise, the target machine's (DNS-resolvable) hostname is target.
- With your target machine SSH-accessible, begin building a base folder structure for the Ansible project.
- At the top level in your project folder, you need:
 - An inventory file, containing information about the machine(s) on which you want to deploy.
 - A top level `site.yml` file, containing the most abstract level of instructions.
 - A role folder structure to contain your `webserver` role.

```
mkdir myproject  
cd myproject
```

```
touch inventory  
touch site.yml  
mkdir roles  
cd roles  
ansible-galaxy init webserver
```

Ansible Example

▪ Creating your inventory file

- A file named **hosts** is created to manage remote network device connections.
- Your inventory file for this project can be very simple. Make it the DNS-resolvable hostname of your target machine:

```
[webservers]
target # can also be IP address
```

- You are defining a group called **webservers** and putting your target machine's hostname (or IP) in it.
- You could add new hostnames/IPs to this group block, or add additional group blocks, to assign hosts for more complex deployments.

```
[webservers]
target1
target2
target3
[dbservers]
target4
target5
target6
```



Ansible Example

- Testing the communication to the remote device ensures:
 - The IP addressing is correct
 - SSH is functioning properly
 - The remote device can understand and return information

```
devasc@labvm:~/labs/ansible/ansible-apache$ ansible webservers -m ping

192.0.2.3 | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: ssh: connect to
host 192.0.2.3 port 22: Connection refused",
  "unreachable": true
}

devasc@labvm:~/labs/ansible/ansible-apache$
```



Ansible Example

▪ Creating your top-level playbook file

- A top-level playbook typically describes the order, permissions, and other details under which lower-level configuration acts, defined in roles, are applied.
- In this example, `site.yml` file identifies which hosts you want to perform an operation on, and which roles you want to apply to these hosts.
- The line `become: true` tells Ansible that you want to perform the roles as root, via sudo.

```
---  
- hosts: webservers  
  become: true  
  roles:  
  - webservers
```

Creating your webservers role

- Next step is to create the role that installs and configures your web server.
- You've already created the folder structure for the role using `ansible-galaxy`.
- Code for the role is contained in a file called `main.yml` in the role's `/tasks` directory.



Ansible Example

- You can edit `roles/webserver/tasks/main.yml` file directly, as shown here.
- The role has two tasks:
 - Deploy Apache2.
 - Copy a new `index.html` file into the Apache2 HTML root, replacing the default `index.html` page.
- In the `apt:` stanza, you name the package, its required state, and instruct the apt module to update its cache.
- In the second stanza, Ansible's copy routine moves a file from your local system to a directory on the target and also changes its owner and permissions.

```
---  
- name: Perform updates and install apache2  
  apt:  
    name: apache2  
    state: present  
    update_cache: yes  
- name: Insert new homepage index.html  
  copy:  
    src: index.html  
    dest: /var/www/html  
    owner: myname  
    mode: '0444'
```




Ansible Example

- **Creating your `index.html` file**
 - Of course, you will need to create a new `index.html` file as well.
 - The Ansible copy command assumes that such files will be stored in the `/files` directory of the role calling them, unless otherwise specified.
 - Navigate to that directory and create the `index.html` file, saving your changes afterward.

```
<html>
<head>
<title>My Website</title>
</head>
<body>
<h1>Hello!</h1>
</body>
</html>
```



Ansible Example

▪ Running your deployment

- Now you're ready to run your deployment. From the top-level directory of your project, you can do this with the statement:

```
ansible-playbook -i inventory -u myname -K site.yml
```

- **-i** names your inventory file.
- **-u** argument names your sudo user.
- **-K** tells Ansible to ask us for your sudo password, as it begins execution.
- **site.yml** is the file that governs your deployment.



Ansible Example

- If all is well, Ansible should ask us for your BECOME password (sudo password), then return results similar to the following:

```
BECOME password:
PLAY [webservers] *****
TASK [Gathering Facts] *****
ok: [192.168.1.33]
TASK [webservers : Perform updates and install apache2] *****
changed: [192.168.1.33]
TASK [webservers : Insert new homepage index.html] *****
changed: [192.168.1.33]
PLAY RECAP *****
192.168.1.33      : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```

- And now, if you visit the IP address of your target machine in a browser, you should see your new homepage.



Ansible Example

▪ Ansible CI/CD walkthrough

- Let's walk through the example as if they were part of a CI/CD pipeline.
- Developer collaborating with you on GitHub commits a change to the website such as in `index.html` file.
- Next, tests in the repository execute syntax and sanity checks as well as code review rules against each pull request.
- Next, the CI/CD system prepares an environment and runs predefined tests for any Ansible playbook. It should indicate the version expected each time and install it. Here's an example pipeline:

```
pip install ansible==2.9.2
ansible-version
ansible-playbook main.yml --syntax-check
ansible-playbook -i staging-servers.cfg main.yml -check
ansible-playbook -i staging-servers.cfg main.yml -vvvv
```

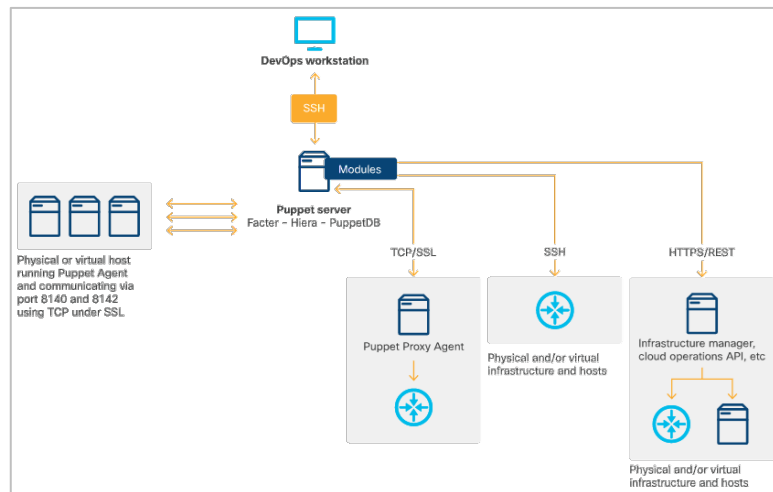
- After Jenkins is done running the job, you can get a notification that all is ready for staging and you can push these changes to production with another pipeline, this time for pushing to production.

Puppet

- Founded as an open source in 2005 and commercialized as Puppet Enterprise by Puppet Labs in 2011.

Architecture components

- A designated server to host main application components.
- A secure client, also known as a Puppet Agent.
- Modules to enable connections for cloud APIs and hardware that cannot run an agent.
- In scaled-out implementations, a proxy agent to offload the work of directly connecting to device CLIs and exchanging information.





Puppet

▪ Installing the Puppet

- Puppet Server requires powerful hardware (or a big VM), and a Network Time Protocol client to be installed, configured, and tested.
- The agents will need the puppet.conf file configured to communicate with the Puppet Server.
- After client service started, it will have certificate signed by the server. The Server will now be able to gather facts from the client and update the client state with any configuration changes.



Puppet

▪ Puppet Code Structure

- Puppet stores components of a project or discrete configuration in a directory tree (`/etc/puppetlabs/code/environments`).
- Subsidiary folders are created according to the configuration in `puppet.conf` or by the operator.
- Puppet comes with a set of basic resources built in. Many additional resources for performing all sorts of operations can be downloaded and installed from Puppet Forge using the `puppet module` command.

Puppet

▪ Puppet at Scale

- Puppet Server is somewhat monolithic, and a monolithic installation is recommended by the (open source) implementors.
- The step to accommodate more hosts is to create additional "compile masters", which compile catalogs for client agents and place these behind a load balancer to distribute work.
- Puppet Enterprise customers can further expand capacity by replacing PuppetDB with a stand-alone, customized database called **PE-PostgreSQL**.

▪ Cisco Puppet resources

- Cisco and the Puppet community maintain extensive libraries of modules for automating Cisco compute and network hardware. These include:
 - Cisco IOS modules enabling management of IOS infrastructure
 - Cisco UCS modules enabling control of UCS via UCS Manager



Puppet Example

- This exercise describes how to install Puppet and then use Puppet to install Apache 2 on a device.
- This approximates the normal workflow for Puppet operations in an automated client/server environment.
- Note that modules can be completely generic and free of site-specific information, then separately and re-usably invoked to configure any number of hosts or infrastructure components.
- **Installing Puppet Server**
 - Puppet Server requires powerful hardware (or a big VM), and a Network Time Protocol client to be installed, configured, and tested.
 - Instructions for installing the server can be found in Puppet's documentation.



Puppet Example

▪ Installing Puppet Client

- When you have the Puppet Server running, you can install Puppet Agents on a host. For example, on a Debian-type Linux system, you can install Puppet Agent using a single command:

```
sudo apt-get install puppet-agent
```

▪ Modify

- When installed, the Puppet Agent needs to be configured to seek a Puppet Server.
- Add the lines to the file `/etc/puppet/puppet.conf` which tells the client, the hostname of your server and name of the authentication certificate that you will generate in the next step.

```
[main]
certname = puppetclient
server = puppetserver
environment = production
runinterval = 15m
```



Puppet Example

- Start the puppet service on the Client:

```
sudo /opt/puppetlabs/bin/puppet resource service puppet ensure=running enable=true
```

- You should get a response similar to the following:

```
Notice: /Service[puppet]/ensure: ensure changed 'stopped' to 'running'  
service { 'puppet':  
  ensure => 'running',  
  enable => 'true',  
}
```

Puppet Example

▪ Certificate signing

- Puppet Agents use certificates to authenticate with the server before retrieving their configurations.
- When the Client service starts for the first time, it sends a request to its assigned server to have its certificate signed, enabling communication.
- On the Server, issue the `ca list` command returns a list of pending certificates.

```
sudo /opt/puppetlabs/bin/puppetserver ca list
```

- The response should be similar to the following:

```
Requested Certificates:
```

```
  puppetclient  (SHA256)
```

```
44:9B:9C:02:2E:B5:80:87:17:90:7E:DC:1A:01:FD:35:C7:DB:43:B6:34:6F:1F:CC:DC:C2:E9:DD:72:61:E6:B2
```



Puppet Example

- You can then sign the certificate, enabling management of the remote node:

```
sudo /opt/puppetlabs/bin/puppetserver ca sign --certname puppetclient
```

- The response:

```
Successfully signed certificate request for puppetclient
```

- The Server and Client are now securely bound and able to communicate.
- This will enable the Server to gather facts from the Client, and let you create configurations on the Server that are obtained by the client and used to converge its state (every 15 minutes).



Puppet Example

▪ Creating a configuration

- Puppet lets you store components of a project or discrete configuration in a directory tree.

```
/etc/puppetlabs/code/environments
```

- Subsidiary folders are created according to the configuration in `puppet.conf` or by the operator.
- In this example, having declared `environment = production`, Puppet has already created a directory for this default site, containing a `modules` subdirectory in which we can store subsidiary projects and manifests for things we need to build and configure.

```
/etc/puppetlabs/code/environments/production/modules
```



Puppet Example

- You will now install Apache2 on your managed client. Puppet operations are typically performed as root, so become root on the Server temporarily by entering `sudo su -`.
- Navigate to the `/modules` folder in the `/production` environment.

```
cd /etc/puppetlabs/code/environments/production/modules
```

- Create a folder structure to contain the `install apache` module.

```
mkdir -p apache2/manifests  
cd apache2/manifests
```

- Inside the manifests folder, create a file called `init.pp`, which is a reserved filename for the initialization step in a module.



Puppet Example

- The class definition orders the steps we want to perform:
 - Step 1. Invoke the package resource to install the named package by `ensure => installed`.
 - Step 2. Invoke the `service` resource to run if its requirement is met. Instruct it to ensure that the service is available, and then enable it to restart automatically when the server reboots.

```
class apache2 {  
  package { 'apache2':  
    ensure => installed,  
  }  
  service { 'apache2':  
    ensure => true,  
    enable => true,  
    require => Package['apache2'],  
  }  
}
```




Puppet Example

- Navigate to the associated manifests folder.

```
cd /etc/puppetlabs/code/environments/production/manifests
```

- Create a `site.pp` file that invokes the module and applies it to the target machine.

```
node 'puppetclient' {  
    include apache2  
}
```

Puppet Example

▪ Deploying the configuration

- You have two options to deploy the completed configuration:
 - Restarting the Puppet Server will now cause the manifests to be compiled and made available to the Puppet Agent on the named device. The agent will retrieve and apply them, installing Apache2 with the next update cycle:

```
systemctl restart puppetserver.service
```

- For development and debugging, you can invoke **Puppet Agent** on a target machine. The agent will immediately interrogate the server, download its catalog and apply it. The results will be similar to the following:

```
sudo puppet agent -t
```



Puppet Example

- The agent will immediately interrogate the server, download its catalog and apply it. The results will be similar to the following:

```
root@target:/etc/puppetlabs/code/environments/production/manifests# puppet agent -t
Info: Using configured environment 'production'
Info: Retrieving pluginfacts
Info: Retrieving plugin
Info: Retrieving locales
Info: Caching catalog for puppetagent
Info: Applying configuration version '1575907251'
Notice: /Stage[main]/Apache2/Package[apache2]/ensure: created
Notice: Applied catalog in 129.88 seconds
```

- After the application has been successfully deployed, enter the target machine's IP address in your browser. This should bring up the Apache 2 default homepage.

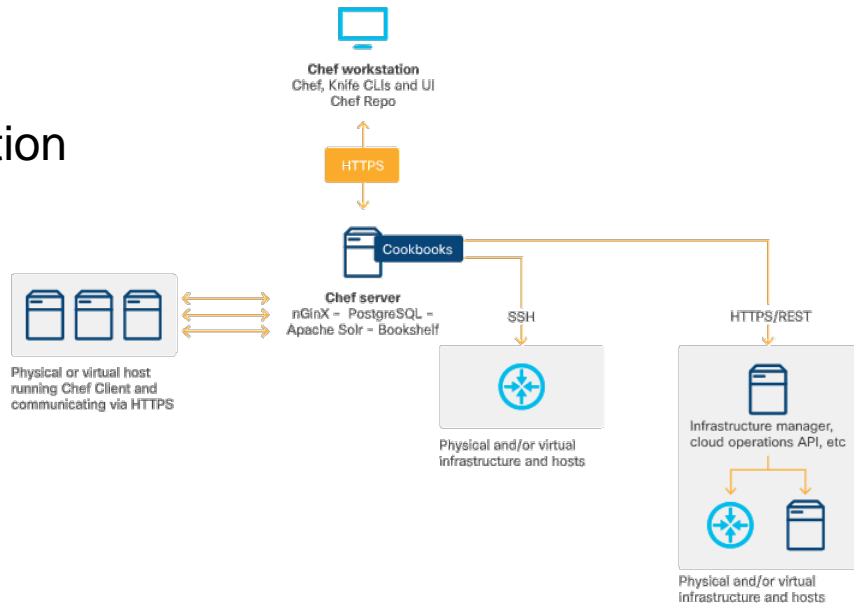


Chef

- Chef provides a complete system for treating infrastructure as code.
- Chef products are partly licensed, but free for personal use (in Chef Infra Server's case, for fewer than 25 managed nodes).
- Chef's products and solutions enable infra-as-code creation, testing, organization, repository storage, and execution on remote targets, either from a stand-alone Chef Workstation, or indirectly from a central Chef Infra Server.

Chef

- Chef Architecture – Components
 - Chef Workstation
 - Chef Infra Client (the host agent)
 - Chef Infra Server
- Most configuration tasks can also be carried out directly between Chef Workstation and managed nodes and devices.



Chef

- Components of Chef Workstation include command-line tools, interaction with Chef Infra Servers, Test Kitchen, ChefSpec and InSpec.
- **Installing Chef Workstation**
 - To begin using Chef, the first step is to install Chef Workstation, which provides a complete operations environment. Refer to the Chef Workstation downloads page for more information.
 - The Workstation is available for Linux and Windows.
- **Running Chef at scale**
 - Chef Infra Server can be configured for high availability by deploying its front-end services into an array of load-balanced proxies.
 - Chef also provides an array of products that together solve most of the problems enterprises face in dealing with increasingly-complex, large-scale, hybrid infrastructures.



Chef

▪ Cisco Chef Resources

- Cisco has developed modified Chef Infra Agents that run in the guest shell of NX-OS switch equipment, enabling this hardware to work with Chef as if it were a managed host.
- It has also developed and maintains a Cisco Chef Cookbook for NX-OS infrastructure, available on Chef Supermarket.
- A GitHub public repo of cookbook and recipe code is also maintained to enable control of a wide range of Cisco products.
- Cisco UCS infrastructure is easily managed with Chef through a cookbook enabling integration with Integrated Management Controllers.

Chef Example – Install and Use Chef

- This example describes how to install Chef and use it to install Apache 2 on a device.
- **Installing Chef Workstation**
 - Chef Workstation provides a complete operations environment. The following example assume that you are installing on an Ubuntu 18.04 LTS virtual machine.
 - If your machine is set up with a standard desktop, browse to the Chef Workstation downloads page, find the download for Ubuntu 18.04, and install it automatically with the Debian package manager.
 - Alternatively, you can install from the command line by copying the URL of the **.deb** package.

```
wget https://packages.chef.io/files/stable/chef-workstation/0.12.20/ubuntu/18.04/chef-workstation_0.12.20-1_amd64.deb
sudo dpkg -i chef-workstation_0.12.20-1_amd64.deb
```


Chef Example – Install and Use Chef

Basic Configuration Management

- After Workstation is installed, start making configuration changes on accessible hosts.
- You will use the `chef-run` command for this. The first time you use `chef-run`, you may be asked to accept licensing terms for the utility.
- For the first configuration exercise, you will provide the information Chef needs to install the `ntp` package. In the process, you will provide the remote username, their sudo password, the name of the remote host `target` and the name of the resource verb.

```
chef-run -U myname -sudo <password> target package ntp action=install
```

- When the client is installed, the task is handed to it, and the process completes. You get back as follows:

```
[✓] Packaging cookbook... done!  
[✓] Generating local policyfile... exporting... done!  
[✓] Applying package[ntp] from resource to target.  
└─ [✓] [target] Successfully converged package[ntp].
```

Chef Example – Install and Use Chef

▪ Install Chef Infra Client

- Chef Infra Client runs locally on conventional compute nodes.
- Chef Workstation can bootstrap Infra Client onto target nodes. You can also preinstall Infra Client on nodes, for example, while creating new nodes on a public cloud. Below is an example script you might run on a target host to do this.
- The script uses a Chef-provided installer called Omnitruck to do this. A Windows version of this script is also available that runs on PowerShell:

```
#!/bin/bash
apt-get update
apt-get install curl
curl -L https://omnitruck.chef.io/install.sh | bash -s once -c current -p chef
```

- Note that the parameters shown above will install the latest version of the Chef client, and do not pin the version.

Chef Example – Install and Use Chef

▪ Chef Infra Server prerequisites

- Before installing Chef Infra Server, install openssh-server and enable keywise access. You would also need to install NTP for time synchronization. You can do this with Chef, or manually:

```
sudo apt-get install ntp ntpdate net-tools
```

- On an Ubuntu system, turn off the default timedatectl synchronization service to prevent it from interfering with NTP synchronization:

```
sudo timedatectl set-ntp 0
```

- After NTP is installed, ensure that it is synchronizing with a timeserver in its default pool. This may take a few minutes, so repeat the command until you see the following:

```
ntpstat
synchronised to NTP server (198.60.22.240) at stratum 2
time correct to within 108 ms
polling server every 256 s
```

- When this shows up, you can install Chef Infra Server.

Chef Example – Install and Use Chef

▪ Install Chef Infra Server

- Chef Infra Server stores configuration and provides it to Clients automatically, when polled, enabling Clients to converge themselves to a desired state.
- To install Chef Infra Server on Ubuntu 18.04, you can perform steps similar to the manual Workstation install after obtaining the URL of the .deb package. At time of writing, the current stable version was 13.1.13-1.

```
wget https://packages.chef.io/files/stable/chef-server/13.1.13/ubuntu/18.04/chef-server-core_13.1.13-1_amd64.deb
sudo dpkg -i chef-server-core_13.1.13-1_amd64.deb
```

- After Chef Infra Server is installed, issue the following command to tell it to read its default configuration, initialize, and start all services.

```
sudo chef-server-ctl reconfigure
```

Chef Example – Install and Use Chef

▪ Install Chef-Manage

- You can also install the web interface for Chef server. This can be done by entering:

```
sudo chef-server-ctl install chef-manage
```

- When the process completes, restart the server and manage components. These are Chef operations, and may take a while, as before.

```
sudo chef-server-ctl reconfigure  
(lots of output)  
sudo chef-manage-ctl reconfigure --accept-license  
(lots of output)
```

- The argument `--accept-license` prevents `chef-manage-ctl` from stopping to ask you about the unique licenses for this product. When this process is complete, you can visit the console in a browser via `https://<IP_OF_CHEF_SERVER>`.

Chef Example – Install and Use Chef

▪ Finish configuring Workstation

- Before Chef Workstation can talk to your Infra Server, you need to do a little configuration.
- To begin, retrieve the keys generated during Server configuration, and store them in the folder `/home/myname/.chef` created during Workstation installation:

```
cd /home/myname/.chef
scp myname@chefserver:./path/*.pem .
```

`/path/` is the path from your home directory on the Server to the directory in which the Server stored keys.

- If you are not using keywise authentication to your Server, `scp` will ask for your password.
- The `.` after `user@host:` refers to your original user's home directory, from which the path is figured.
- The wildcard expression finds files ending in `.pem` at that path. The closing dot means copy to the current working directory.
- Run the `ls` command from within the `.chef` folder to see if your keys made it.



Chef Example – Prepare to Use Knife

▪ Prepare to use Knife

- Knife is a tool for managing cookbooks, recipes, nodes, and other assets, and for interacting with the Chef Infra Server.
- Within the `.chef` folder, edit the (initially empty) file named `config.rb` and include the following lines of code, adapting them to your environment:

```
chef_server_url 'https://chefserver/organizations/<short_name>'
client_key '/home/<myname>/.chef/<userkey.pem>'
cookbook_path [
  '/home/<myname>/cookbooks'
]
data_bag_encrypt_version 2
node_name '<username>'
validation_client_name '<short_name>-validator'
validation_key '/home/<myname>/.chef/<short_name>-validator.pem'
```



Chef Example – Prepare to Use Knife

- Save the `config.rb` file and then create the directory `/home/myname/cookbooks`.
- Finally, issue the command `knife ssl fetch`.
- If you have correctly set up the `config.rb`, Knife will consult with your server, retrieve its certificate, and store it in the directory.
- Chef will find this automatically when it is time to connect with the server, providing assurance that the server is authentic.



Chef Example – Prepare to Use Knife

▪ Bootstrap a target node with knife

- After Knife is configured, you can bootstrap your target node.
- To bootstrap, issue the following command, replacing variable fields with your information. The command is set up to use keywise authentication to the target machine.
- The redundant `--sudo` and `--use-sudo-password` commands tell Knife to use sudo to complete its work.
- The `-P` option provides your `sudo` password on the target machine.
- `<name_for_your_node>` is an arbitrary name. The `--ssh-verify-host-key` never flag and argument cause the command not to pause and ask your permission interactively if it finds that you've never logged into this server before.



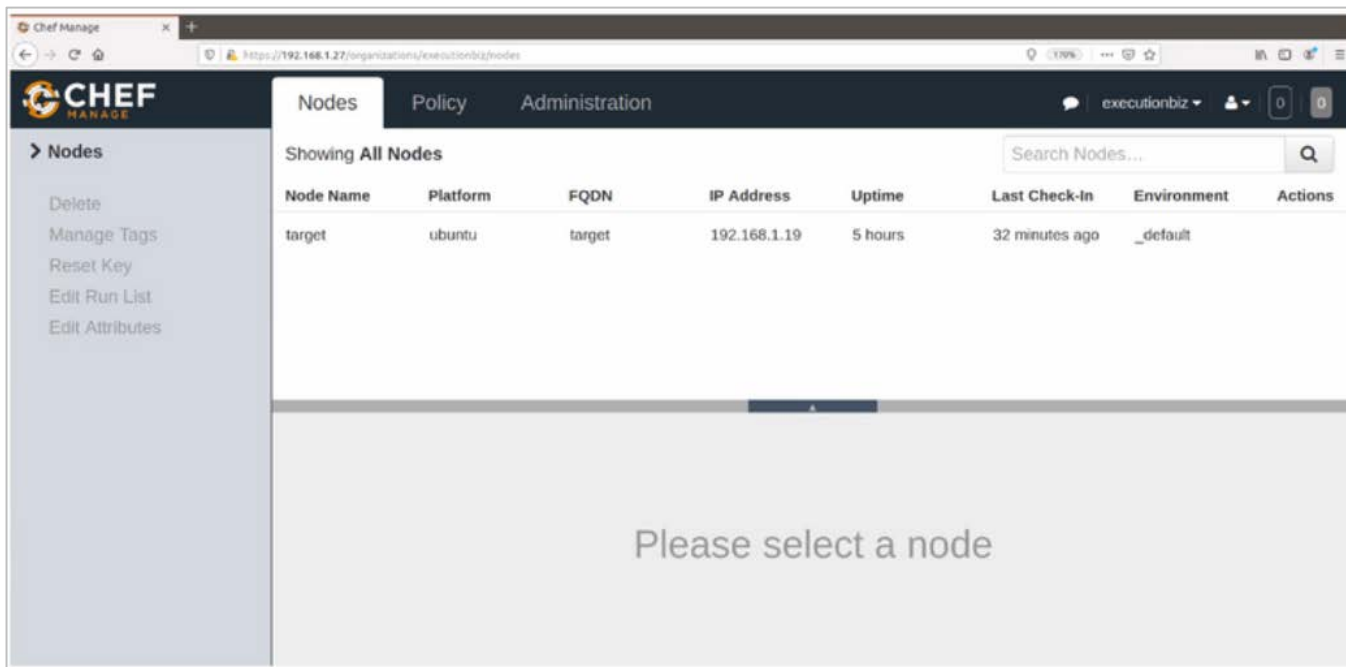
Chef Example – Prepare to Use Knife

- If the command works correctly, you would get back the given output. Note that Chef has detected the earlier installation and has not overwritten it.

```
[target] [sudo] password for myname:
[target] ----> Existing Chef Infra Client installation detected
[target] Starting the first Chef Infra Client run...
[target] +-----+
√ 2 product licenses accepted.
+-----+
[target] Starting Chef Infra Client, version 15.5.17
[target]
[target] Creating a new client identity for target using the validator key.
[target]
[target] resolving cookbooks for run list: []
[target]
[target] Synchronizing Cookbooks:
[target]
[target] Installing Cookbook Gems:
[target]
[target] Compiling Cookbooks...
[target]
[target] [2019-12-10T15:16:56-05:00] WARN: Node target has an empty run list.
[target] Converging 0 resources
[target]
[target]
[target] Running handlers:
[target]
[target] Running handlers complete
[target] Chef Infra Client finished, 0/0 resources updated in 02 seconds
[target]
```

Chef Example – Prepare to Use Knife

- **Chef Manage Displays Your Target Node:** Now, if you check back in your browser and refresh Chef Manage, you should see that your target machine is now being managed by the server.



The screenshot shows the Chef Manage web interface. The browser address bar displays `https://192.168.1.27/organizations/executionbiz/nodes`. The interface has a dark header with the Chef logo and navigation tabs for "Nodes", "Policy", and "Administration". The "Nodes" tab is active, and the page shows "Showing All Nodes" with a search bar. A table lists the nodes, with one node named "target" visible. The table has columns for Node Name, Platform, FQDN, IP Address, Uptime, Last Check-In, Environment, and Actions. Below the table, a large grey area contains the text "Please select a node".

| Node Name | Platform | FQDN | IP Address | Uptime | Last Check-In | Environment | Actions |
|-----------|----------|--------|--------------|---------|----------------|-------------|---------|
| target | ubuntu | target | 192.168.1.19 | 5 hours | 32 minutes ago | _default | |



Chef Example – Putting it all Together

- Now you will use everything together to create an actual recipe, push it to the server, and tell the target machine's client to requisition and converge on the new configuration.
 - To start, create a cookbook to build a simple website. Navigate to your cookbooks directory, create a new cookbook called `apache2` and navigate in it.
 - Check the cookbook folder structure. There are folders already prepared for recipes and attributes. Add an optional directory and subdirectory for holding files your recipe needs.
 - Files in the `/default` subdirectory of a `/files` directory within a cookbook can be found by recipe name alone, no paths are required.

Chef Example – Putting it all Together

- Now create a homepage for your website.

```
vi index.html
<html>
<head>
<title>Hello!</title>
</head>
<body>
<h1>HELLO!</h1>
</body>
</html>
```

- Save the file and exit. Navigate to the recipes directory , where Chef has already created a `default.rb` file for us. The `default.rb` file will be executed by default when the recipe is run with this command.

```
cd ../../recipes
```

- Add some data to the `default.rb` file and again edit the file.

```
vi default.rb
```



Chef Example – Putting it all Together

- The header at the top is created for you. Underneath, the recipe performs three actions.
 - The first resource you are invoking `apt_update` handles the apt package manager on Debian.
 - The package function is used to install the apache2 package from public repositories.
 - Finally, you use the `cookbook_file` resource to copy the `index.html` file from `/files/default` into a directory on the target server.

```
#
# Cookbook:: apache2
# Recipe:: default
#
# Copyright:: 2019, The Authors, All Rights Reserved.
apt_update do
  action :update
end
package 'apache2' do
  action :install
end
cookbook_file "/var/www/html/index.html" do
  source "index.html"
  mode "0644"
end
```

Chef Example – Putting it all Together

- Save the default.rb file and then upload the cookbook to the server.
- You can then confirm that the server is managing your target node.
- The Knife application can interoperate with your favorite editor. To enable this, perform the following export with your editor's name: `export EDITOR=vi`
- This lets the next command execute interactively, putting the node definition into vi to let you alter it manually.
- As you can see, the expression `"recipe[apache2]"` has been added to the `run_list` array which contains an ordered list of the recipes you want to apply to this node.
- Save the file in the usual manner. Knife immediately pushes the change to the Infra Server.

```
knife node edit target{
  "name": "target",
  "chef_environment": "_default",
  "normal": {
    "tags": [
    ]
  },
  "policy_name": null,
  "policy_group": null,
  "run_list": [
    "recipe[apache2]"
  ]
}
```



Chef Example – Putting it all Together

- Finally, you can use the `knife ssh` command to identify the node, log into it non-interactively using SSH, and execute the `chef-client` application.
- If all goes well, Knife gives you back a very long log that shows you exactly the content of the file that was overwritten and confirms each step of the recipe as it executes.
- At this point, you should be able to point a browser at the target machine's IP address, and see your new index page.

Summary of Automation Tools

- This has been a high-level introduction to three modern DevOps toolkits. You should now be ready to:
 - Deploy and integrate free versions of the major components of Ansible, Puppet, and/or Chef on a range of substrates, from desktop virtual machines to cloud-based VMs on Azure, AWS or other IaaS platforms.
 - Experience each platform's declarative language, style of infra-as-code building and organizing, and get a sense of the scope of its library of resources, plugins, and integrations.
 - Get practice automating some of the common IT tasks you may do at work or solve deployment and lifecycle management challenges you set yourself, in your home lab.
 - Hands-on exercises and work will give you a complete sense of how each platform addresses configuration themes, and help you overcome everyday IT challenges.

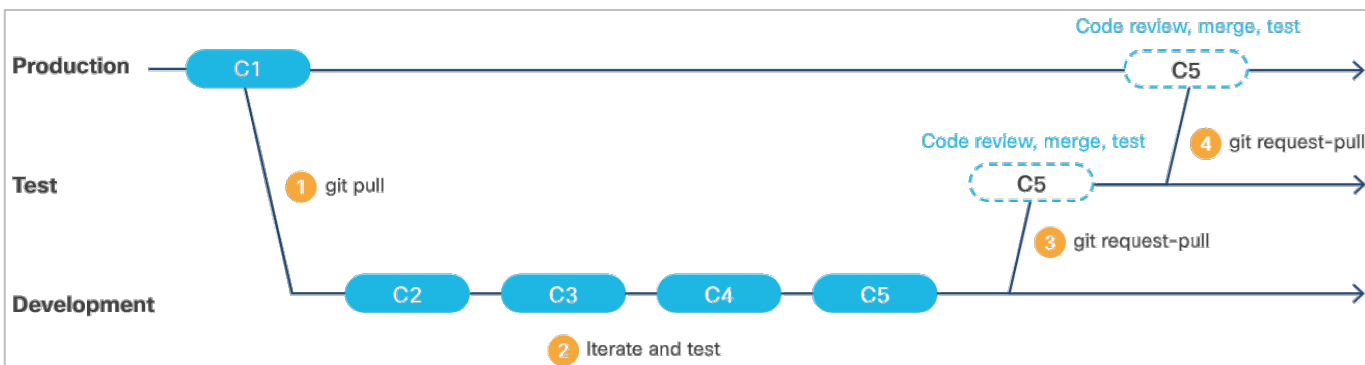


7.5 INFRASTRUCTURE AS CODE



Why Store Infrastructure as Code?

- The term immutability refers to maintaining systems entirely as code, performing no manual operations on them at all.
- GitOps: **modern infrastructure-as-code**
- GitOps is also referred to as "operations by pull request."
- In a typical GitOps setup, you might maintain a repository, such as a private repo on GitHub, with several branches called Development, Testing/UAT and Production.



Why Store Infrastructure as Code?

- **Where can GitOps take you?**

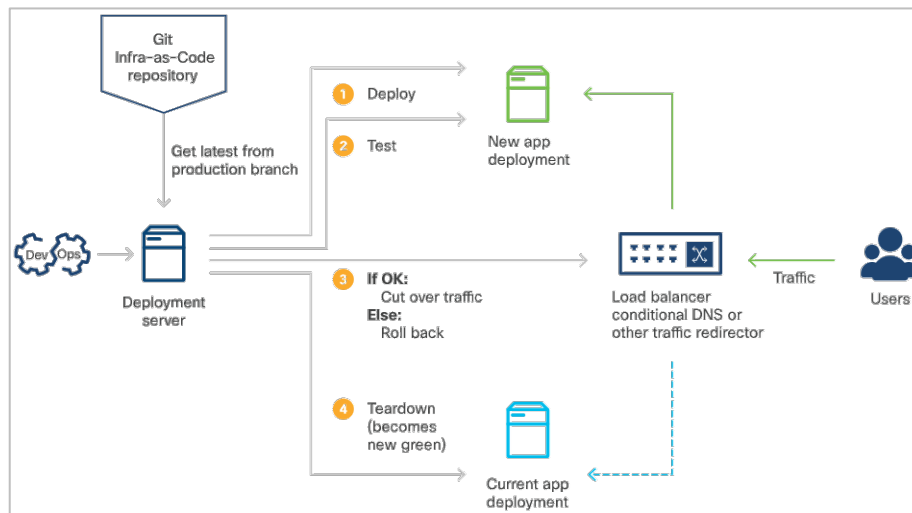
- When all the GitOps procedures, workflow and other components are in place, developers can look at implementing elite deployment strategies.

- **Blue/Green deployment**

- Blue/green deployment is a method for reducing or eliminating downtime in production environments.
- It is required to maintain two identical production environments (Not necessarily, blue and green. Any two colors such as Red and Black will do).
- It is also required to develop the capability of quickly redirecting application traffic to one or the other.
- Note: Some DevOps practitioners differentiate between blue/green and red/black strategies. They say that in blue/green, the traffic is gradually migrated from one environment to the other, so it hits both systems for some period; whereas in red/black, traffic is cut over all at once.

Why Store Infrastructure as Code?

- A release is deployed to the environment not currently in use (**Green**). After acceptance testing, redirect traffic to this environment.
- If problems are encountered, switch traffic back to the original environment (**Blue**).
- If the Green deployment is judged adequate, resources owned by the Blue deployment can be relinquished, and roles swapped for the next release.





Why Store Infrastructure as Code?

▪ Canary Testing

- Canary testing is similar to rolling blue/green deployment, but somewhat more delicate.
- The migration between old and new deployments is performed on a customer-by-customer (or even user-by-user) basis.
- Migration are made to reduce the risk and improve the quality of feedback.



7.6 AUTOMATING TESTING



Manual Testing

- Several manual commands can be used to quickly access the status of interfaces, routing protocols, and other configurations:
 - **show run**
 - **show ip route**
 - **show ip interface brief**
 - **ping**

```
CSR1k0#sh ip interface brief
Interface          IP-Address      OK? Method Status  Protocol
GigabitEthernet1  192.168.56.101 YES DHCP   up      up
CSR1k0#
```

```
CSR1k0#ping 192.168.56.101
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.56.101, timeout is 2 seconds:
!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
CSR1k0#_
```


Automated Test and Validation

- By using unit testing tools like pytest developers can build an environment where the code can be tested automatically when changes are made.
- Unit-testing frameworks make tests a part of the codebase, following the code through developer commits, pull requests, and code-review gates to QA/test and Production. This unit test framework is useful in test-driven development (TDD) environments.
- **The challenges of testing a network**
 - The behavior and performance of a real-world network is collective, maintained by the configurations of many discrete pieces of equipment and software.
 - In traditional environments, connectivity and functionality are manually maintained across numerous individual pieces of equipment via diverse interfaces. This is difficult, time-consuming, extremely error-prone, and risky.
 - As networks become more complex and carry more diverse and performance-sensitive traffic, risks to security and performance degradations are higher.

Automated Test and Validation

- **Testing Software Defined Networks (SDN)**
 - Cisco has made huge progress in developing Software Defined Networks (SDN) and middleware that let engineers to address a physical network as a single programmable entity. In Cisco's case, this includes:
 - Application Centric Infrastructure (ACI)
 - Digital Network Architecture Center (Cisco DNA Center)
 - REST API and SDKs enable integration with automation tools like Ansible, Puppet, and Chef
- **Python Automated Test System (pyATS)**
 - Python Automated Test System (pyATS) is a Python-based network device test and validation solution.
 - pyATS originated as the low-level Python underpinning for the test system as a whole.
 - Its higher-level library system, Genie, provides the necessary APIs and libraries that drive and interact with network devices, and perform the actual testing.



Automated Test and Validation

- pyATS has several key features:
 - pyATS framework and libraries can be leveraged within any Python code.
 - It is modular and includes components such as AETest and EasyPy.
 - EasyPy is used to handle bundling and running jobs.
 - A CLI enables rapid interrogation of live networks, extraction of facts, and helps automate running of test scripts and other forensics.
 - pyATS provides an enormous interface library to Cisco and other infrastructure via a range of interfaces.
 - pyATS can consume, parse, and implement topologies described in JSON, as YANG models, and from other sources.
 - pyATS can also be integrated with automation tools for building, provisioning, and teardown.

pyATS Example

- The following content shows how to use pyATS to create and apply tests.
- **Virtual environments**
 - The pyATS tool is best installed for personal work inside a Python virtual environment (venv).

```
python3 -m venv /path/to/new/virtual/environment
```

- A **venv** is an environment copied from your base environment but kept separate from it.
- This enables you to avoid installing software that might permanently change the state of your system.
- Virtual environments exist in folders in your file system. When they are created, they can be activated, configured at will, and components installed in them can be updated or modified without changes being reflected in your host's configuration.
- The ability to create virtual environments is native to Python 3, but Ubuntu 18.04 may require you to install a **python3-venv** package separately.

pyATS Example

- To create a venv on Ubuntu:
 - Ensure that `python3-pip`, the Python3 package manager, is in place and install `git`.
 - Create a new virtual environment in the directory of your choice.
 - Venv creates the specified working directory and folder structure containing functions and artifacts describing this environment's configuration. At this point, you can `cd` to the `myproject` and activate the venv.
- **Installing pyATS**
 - Install the pyATS from the public Pip package repository (PyPI).
 - Verify that it was installed by listing the help, using `pyats --help`.
 - Clone the pyATS sample scripts repo, maintained by Cisco DevNet, which contains sample files.
- Note: You may see "**Failed building wheel for...<wheelname>**" errors while installing pyATS. You can safely ignore those errors as `pip` has a backup plan for those failures and the dependencies are installed despite errors reported.

pyATS Example

- **pyATS test case syntax**
 - The test declaration syntax for pyATS is inspired by Python unit-testing frameworks like `pytest`.
 - It supports basic testing statements, such as an assertion that a variable has a given value and adds to that the ability to explicitly provide results.
- **pyATS scripts and jobs**
 - A pyATS script is a Python file where pyATS tests are declared.
- **pyATS testbed file**
 - A testbed can be a single YAML file or can be programmatically assembled from YAML and Python.
 - The testbed file is an essential input to the rest of pyATS library and ecosystem as it provides information to the framework for loading the right set of library APIs for each device, and how to effectively communicate to them.
 - Real testbed files for large topologies can be long, deeply-nested, and complex.



pyATS Example

▪ **pyATS Library: Genie**

- Genie is the pyATS higher-level library system that provides APIs for interacting with devices, and a powerful CLI for topology and device management and interrogation.
- When installed, it adds its features and functionalities into the pyATS framework.



7.7 NETWORK SIMULATION



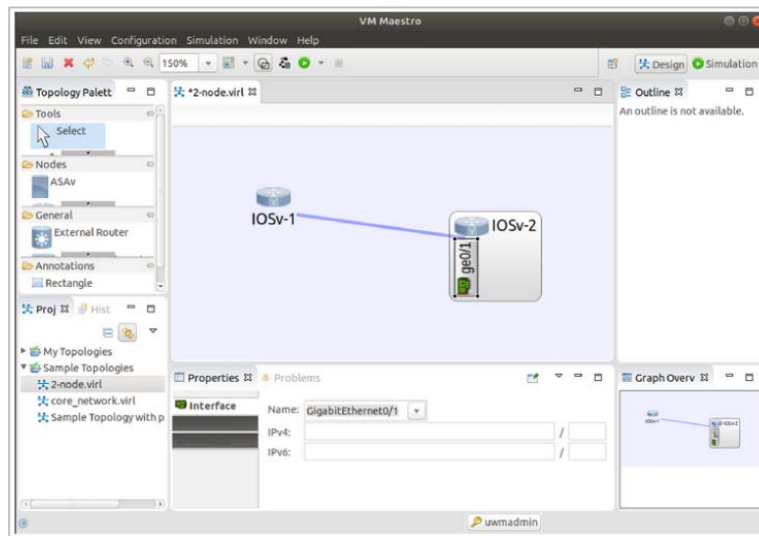


Network Simulation and VIRL

- Network simulation provides a means to test network configurations, debug configuration code, and to work with and learn Cisco infrastructure and APIs in a safe, convenient, and non-cost-prohibitive way.
- Cisco Virtual Internet Routing Laboratory (VIRL) is a commercial product originally developed for internal use at Cisco, with broad and active community support. Now in version 2, VIRL can run on bare metal, or on large virtual machines on several hypervisor platforms.
- The virtual equipment that runs inside VIRL uses the same code that runs inside actual Cisco products.
- **VIRL components and workflow**
 - VIRL provides a local CLI for system management, a REST interface for integration with automation, and a powerful UI that offers a complete graphical environment for building and configuring simulation topologies.
 - The UI comes with several topologies to get started. Among these is a two-router IOS network simulation that can quickly be made active and explored.

Network Simulation and VIRL

- VIRL's **Design Perspective** view allows to modify existing simulations or compose new simulations by dragging, dropping, and connecting network entities, configuring them.
- The visualization has clickable elements that explore configuration of entities and make changes via the WebUI, or by connecting to network elements via console.



Network Simulation and VIRL

▪ VIRL Files

- The individual device configurations, or entire simulated network configs can be extracted as `.virl` files.
- VIRL enables you to define simulations as code, enabling both-ways integration with other software platforms for network management and testing.
- VIRL's native configuration format is called a `.virl` file which is a human-readable YAML file.
- The `.virl` file contains complete descriptions of the IOS routers, their interface configurations and connection, credentials for accessing them, and other details.
- These files can be used to launch simulations via the VIRL REST API and the `.virl` files can be converted to and from testbed files to use with PyATS and Genie.
- The `.virl` file provides a method for determining if configuration drift has occurred on the simulation.
 - A simple `diff` command can compare a newly-extracted `.virl` file with the original `.virl` file used to launch the simulation, and differences will be apparent.



7.8 INFRASTRUCTURE AND AUTOMATION SUMMARY



What Did I Learn in this Module?

- Automation is using code to configure, deploy, and manage applications together with the compute, storage, and network infrastructures and services on which they run.
- Cloud computing, lets developers and operators use software to requisition, configure, deploy, and manage bare-metal and virtualized compute, storage, and network resources.
- For full-stack automation to be truly effective, it requires changes to organizational culture, including breaking down the historical divides between Development (Dev) and Operations (Ops).
- DevOps/SRE have many core principles and best practices: A focus on automation, the idea that failure is normal and a reframing of availability in terms of what a business can tolerate.



What Did I Learn in this Module?

- Cloud automation enables you to provision virtualized hosts, configure virtual networks and other connectivity, requisition services, and then deploy applications on this infrastructure.
- Three of the most popular automation tools are Ansible, Puppet, and Chef.
- Immutability refers to maintaining systems entirely as code, performing no manual operations on them at all.
- The unit test framework is useful in test-driven development (TDD) environments.
- Network simulation provides a means to test network configurations, debug configuration code, and to work with and learn Cisco infrastructure and APIs in a safe, convenient, and non-cost-prohibitive way.
- Cisco Virtual Internet Routing Laboratory (VIRL) can run on bare metal or on large virtual machines on several hypervisor platforms.

