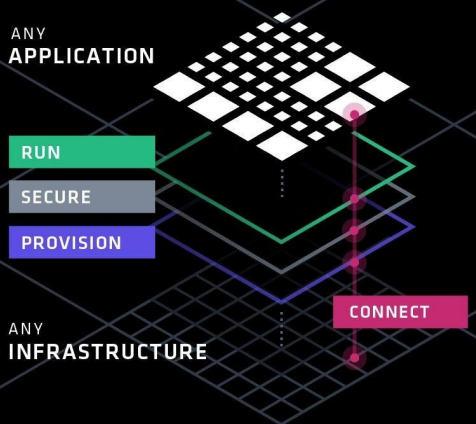
SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Unit-3

Terraform – Getting Started

HashiCorp Suite



OPEN SOURCE

- HashiCorp Nomad
- HashiCorp Vault
- HashiCorp Terraform
- HashiCorp Packer
- HashiCorp Vagrant
- HashiCorp Consul

Provisioning infrastructure through software to achieve ~~consistent~~ and ~~predictable~~ environments.

Core Concepts

Defined in code

Stored in source

Declarative or

Idempotent and consistent

Push or pull

Infrastructure as Code Benefits

Automated deployment

Consistent environments

Repeatable process

Reusable components

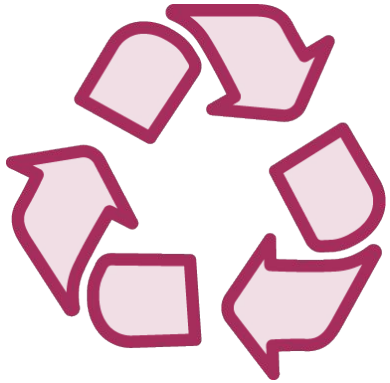
Documented architecture

Automating Infrastructure Deployment

Provisioning Resources

Planning Updates

Using Source Control



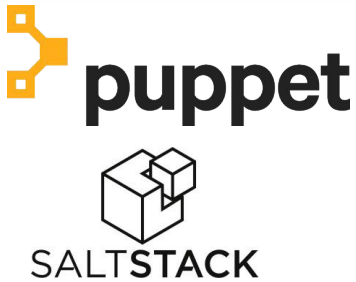
Reusing Templates

Terraform

- A provisioning declarative tool that based on Infrastructure as a Code paradigm
- Uses own syntax - HCL (Hashicorp Configuration Language)
- Written in Golang.
- Helps to evolve you infrastructure, safely and predictably
- Applies Graph Theory to IaaC
- Terraform is a multipurpose composition tool:
  - Composes multiple tiers (SaaS/PaaS/IaaS)
  - A plugin-based architecture model
- Open source. Backed by Hashicorp company and Hashicorp Tao (Guide/Principles/Design)

Other tools

- Cloudformation, Heat, etc.
- Ansible, Chef, Puppet, etc.
- Boto, fog, apache-libcloud, etc.
- Custom tooling and scripting



AWS Cloudformation VS OpenStack Orchestration (Heat)

- AWS Locked-in
- Initial release in 2011
- Sources hidden behind a scene
- AWS Managed Service / Free
- Cloudformation Designer
  - Drag-and-drop interface.
- Json, Yaml (since 2016)
- Rollback actions for stack updates
- Change sets (since 2016)

- Open source
- Initial release around 2012
- Heat provides CloudFormation-compatible Query API for Openstack
- UI: Heat Dashboard
- Yaml

Ansible, Chef, Puppet, etc

- Created for the purpose to be a configuration management tool.
- Suggestion: don't try to mix configuration management and resource orchestration.
- Different approaches:
  - Declarative: Puppet, Salt
  - Imperative: Ansible, Chef
- The steep learning curve if you want to use orchestration capabilities of some
  of these tools.
- Different languages and approaches:
  - Chef - Ruby
  - Puppet - Json-like syntax / Ruby
  - Ansible – Yaml | python

Boto, fog, apache-libcloud, etc.

- low-level access to APIs
- Some libs focused on specific cloud providers, others provide common interface for few different clouds
- Inspires to create custom tooling

Custom tooling and scripting

- Error-prone and tedious
- Requires many human-hours
- The minimum viable features
- Slowness or impossibility to evolve, adopt to quickly changing environments

Terraform is not a cloud agnostic tool

It’s not a magic wand that gives you power over all clouds and systems.

It embraces all major Cloud Providers and provides common language to orchestrate your infrastructure resources.


HashiCorp
Terraform
vs
Azure

Feature Comparison

Feature	ARM	Terraform
Infrastructure as Code (IaC)	Yes	Yes
Readability	JSON	HashiCorp Config Language (HCL)
Execution plans	No	**Yes**
Dependencies	Yes (Explicit)	Yes (Implied)
Multi-Cloud	No	**Yes**
Configuration	Limited	Limited (can do some storage tasks)
Rollback State	Yes – deploy prior template / rollback	Yes – maintains state
Azure Preview features	Yes	Yes – inline ARM snippets
KeyVault support	Yes	Yes
Corrupted State	**State not needed**	Can be an issue
Supports Dev Ops	Yes	Yes
Cost / Support	**Free**, uses Azure support	Free / Paid (purchase support)
Parallel deployments	Yes	Yes
Runs "Locally"	**ARM template is uploaded / deployed in Azure**	Terraform uses REST calls via a client machine
Delete resource in portal and not worry about state	**Yes**	No
Support Comments	Via an Attribute	**Yes** including block comments
Speed	Can take a while	Can be fast since it can deploy just a single item based upon its plan
Math Functions	Yes	Yes
Count / Loops	Yes	Yes
Sub-Templates/Modules	Yes – Linked Templates	Yes – Modules
Deploy to multiple resource groups	Requires many template	Can be done in one template
Reference existing resources	Variable w/resource id path	"data" resource type
Reverse Engineer resources	Export and Visual Studio	Object by Object by importing

AWS
Cloudformation

VS

HashiCorp
Terraform

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Cloudformation	Terraform
Closed Source, maintained/updated by AWS	Open source, many contributors
Suitable for working on AWS Cloud	Cloud Agnostic: Suitable for working with multi-cloud workloads
GUI access for no cost	GUI access requires expensive enterprise licence
No need to Manage State	Need to Manage State yourself
Supports **YAML** and **JSON** for configuration language	Supports **JSON** and **HCL** for configuration language
Nested Stacks lets you work with multiple templates. This concept is hard to grasp for beginners and has limitations	Working with multiple tf files easier .

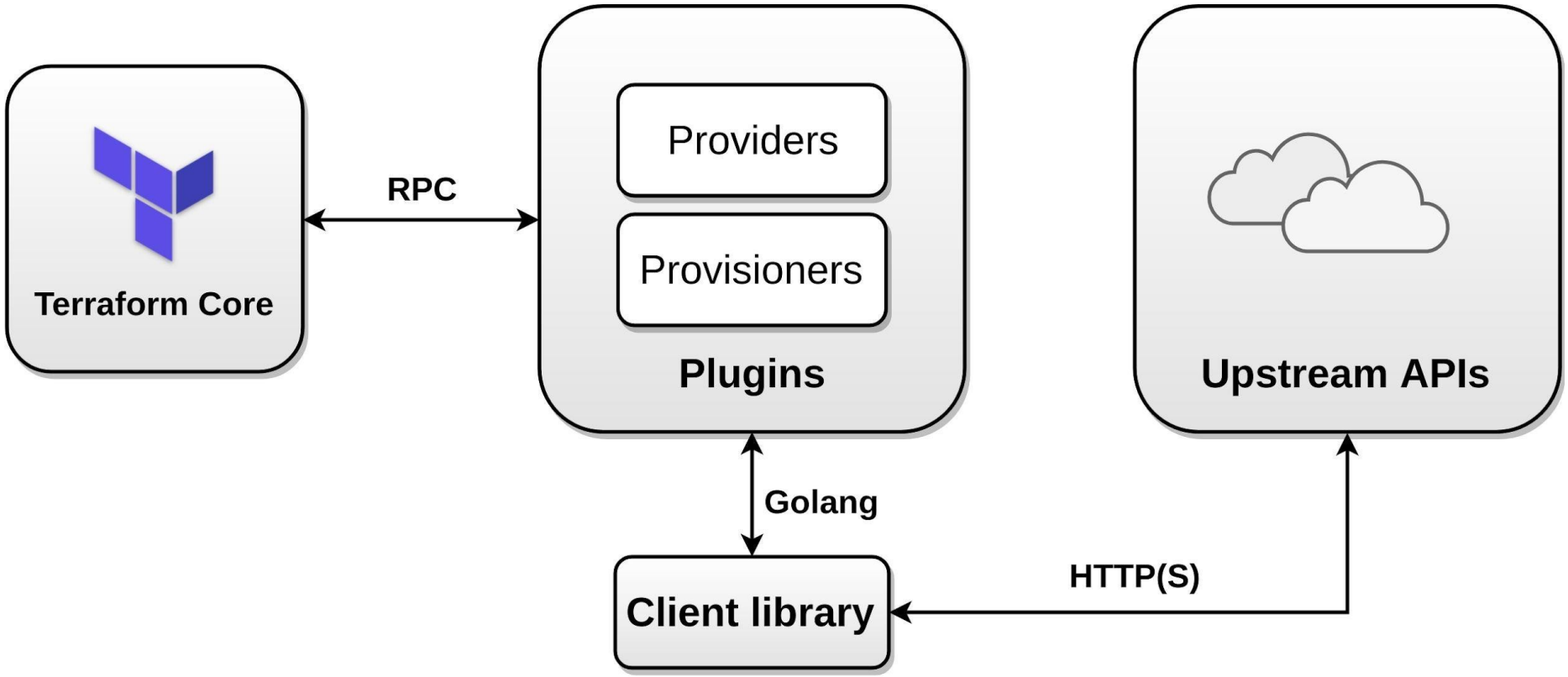
	CloudFormation	Terraform
Scope	CloudFormation covers most parts of AWS.	Terraform covers most AWS resources. It also supports other cloud providers and 3rd party services.
License and Support	CloudFormation is a managed service offered by AWS for free. AWS provides support as per the selected support plan.	Terraform is an open-source project. HashiCorp offers support plans like Terraform SaaS and Enterprise.
Syntax	JSON and YAML-based templates are somewhat convoluted.	HCL (HashiCorp Configuration Language)-based templates are easier to interpret.
Modularization	CloudFormation does not use modules, though it offers multiple ways to create 'modules' with some limitations.	Handling modules with Terraform is simple and they help in creating a reproducible infrastructure.
User Experience	CloudFormation provides a user interface to create, modify, and present resource dependencies graphically.	The open-source version of Terraform can be used only via a command-line interface (CLI). Terraform SaaS and Enterprise provide a user interface.
State Management	CloudFormation manages state within an out-of-the-box managed service.	Terraform stores its state on disk by default. It also offers a remote state where you can configure the 'remote state' yourself.
Import the Existing Infrastructure	CloudFormation cannot be used to manage resources created outside of CloudFormation.	Terraform supports the import and management of resources created outside of Terraform.
Verify Changes	CloudFormation offers changesets that you can use to verify changes.	Terraform provides a command named plan, which gives a very detailed overview of what will be modified if you apply your blueprint.
Rolling Updates and Rollback	CloudFormation can perform the rolling update of Auto Scaling Groups, including a rollback in case of a failure.	No support for rolling updates of Auto Scaling groups or automatic rollback.

Image 2: Comparative Analysis of Terraform and CloudFormation

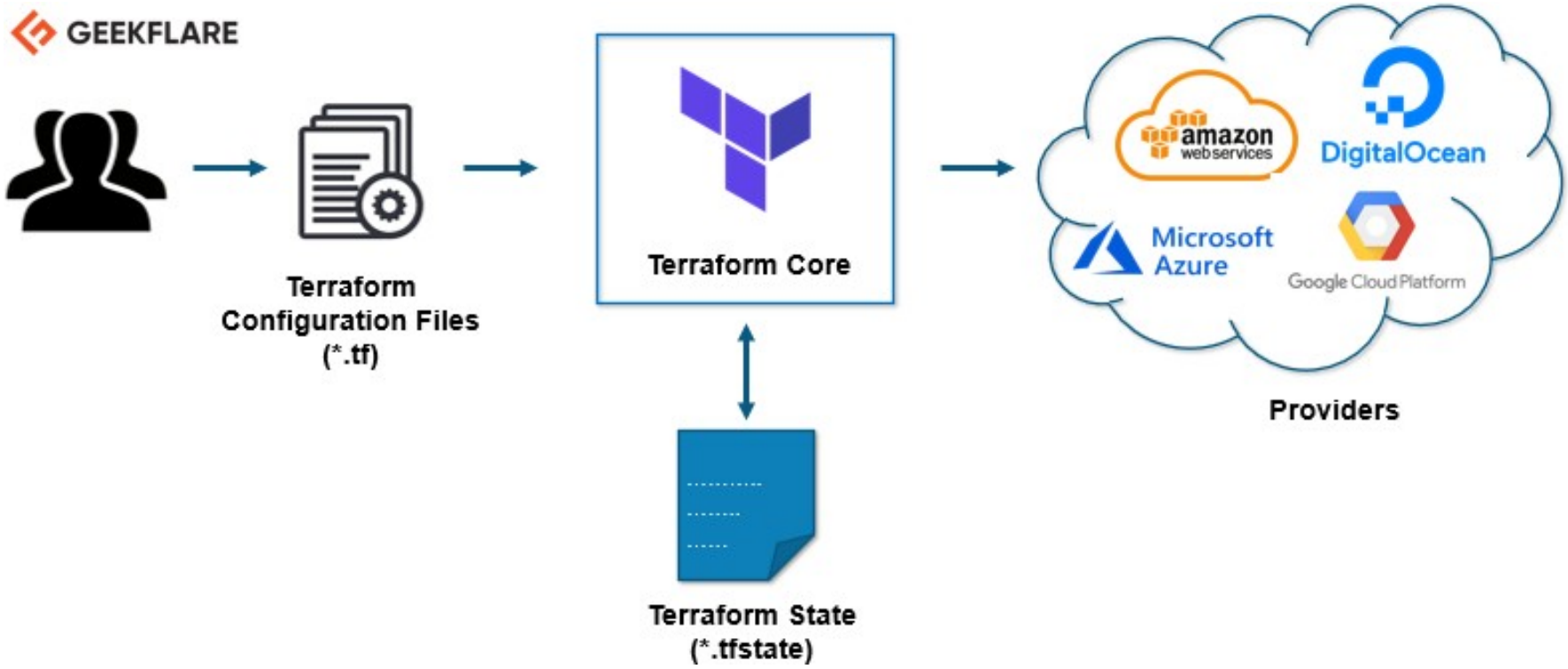
	ARM	TERRAFORM	CLOUDFORMATION
CONFIGURATION LANGUAGE	9	12	9
CODE READABILITY	6	11	6
EXCEPTIONS	9	3	9
ERROR TRACKING	9	6	7
AUDIT	9	3	9
TRACKING CAPABILITIES	11	6	11
DEPLOYMENT MONITORING	9	7	9
TOOLING SUPPORT	14	11	11
MODULARITY	8	12	9
VERSIONING AND MANAGEMENT OF THE STATE	6	8	9
VALIDATION MECHANISM	5	9	5
MAINTAINABILITY	5	9	5
MULTI-PLATFORM	0	15	0

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

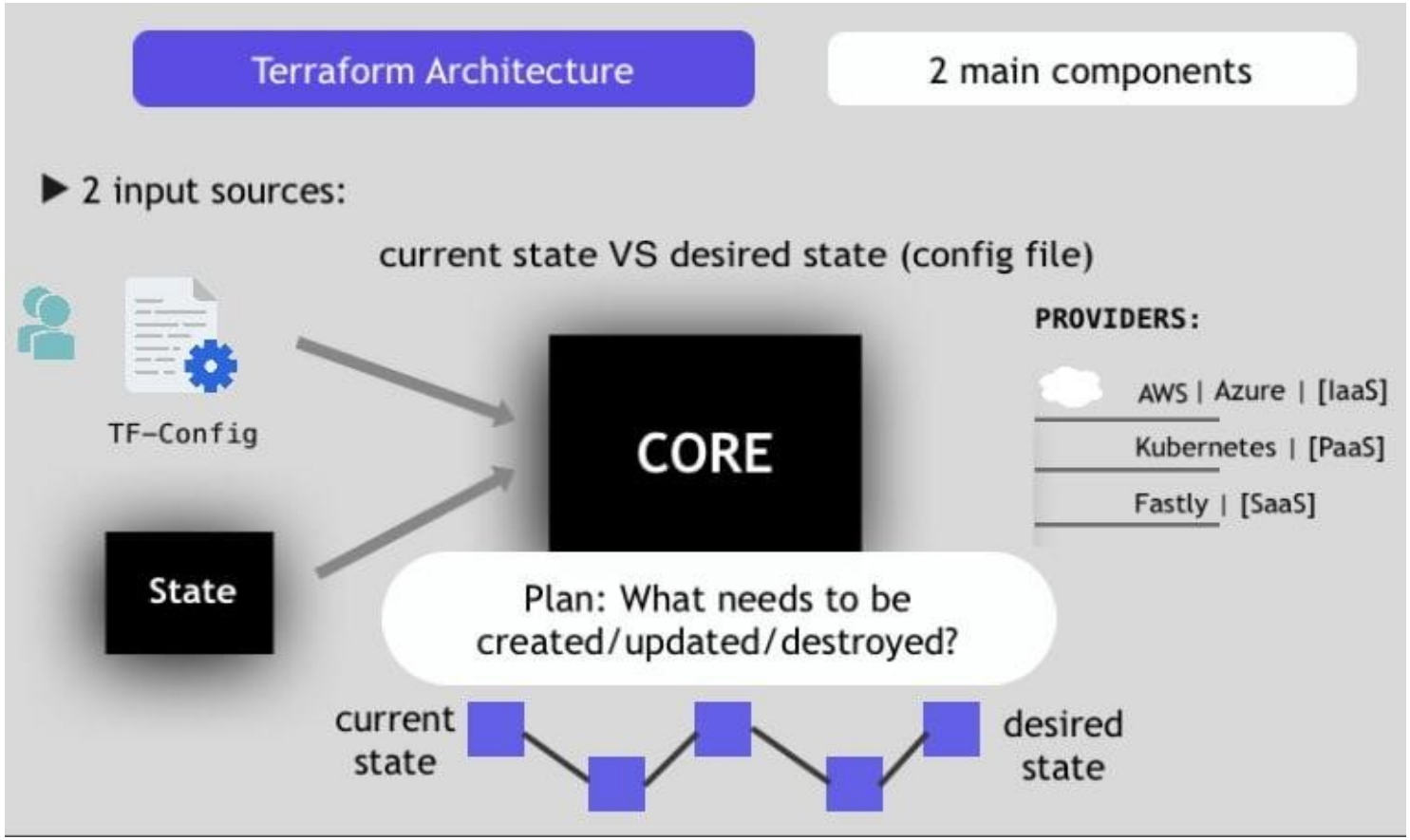
Architecture



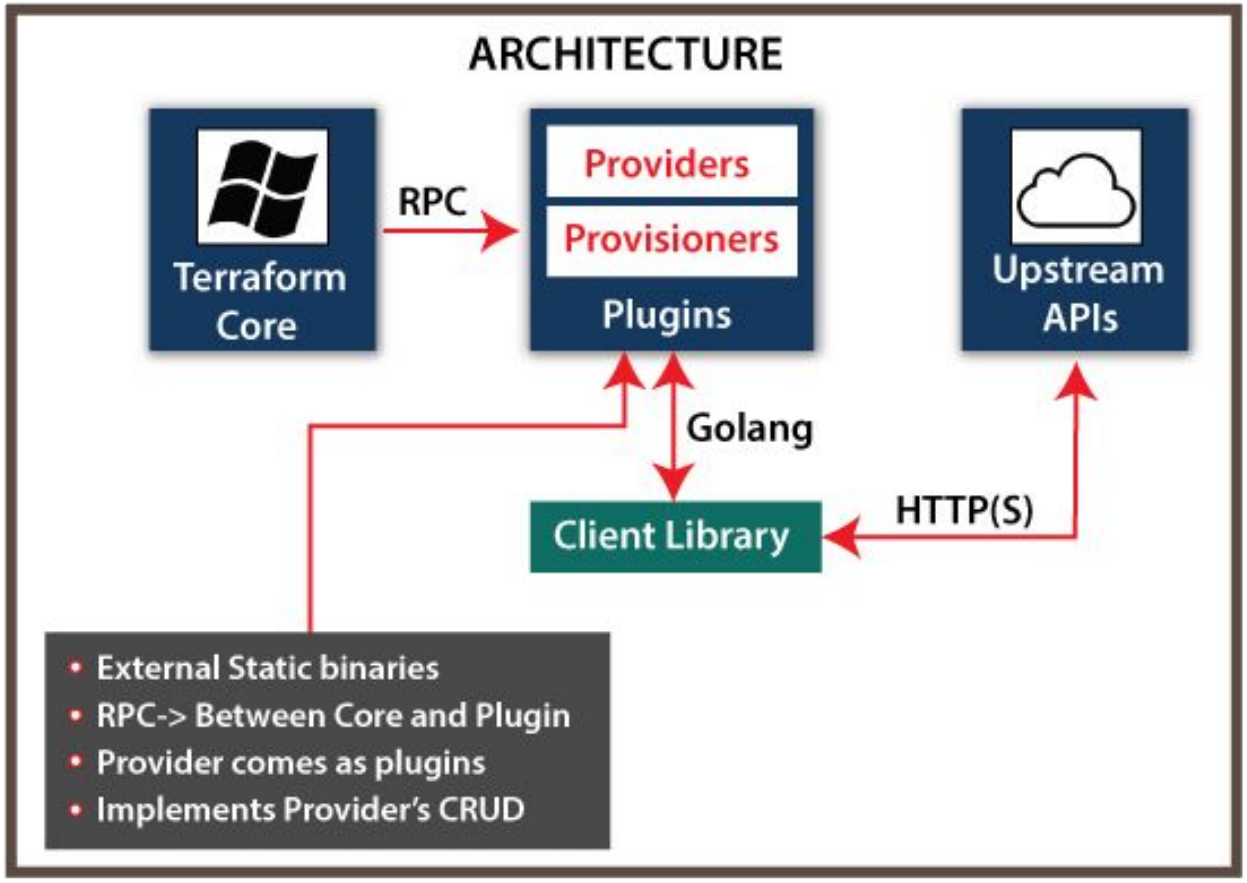
Architecture



Architecture

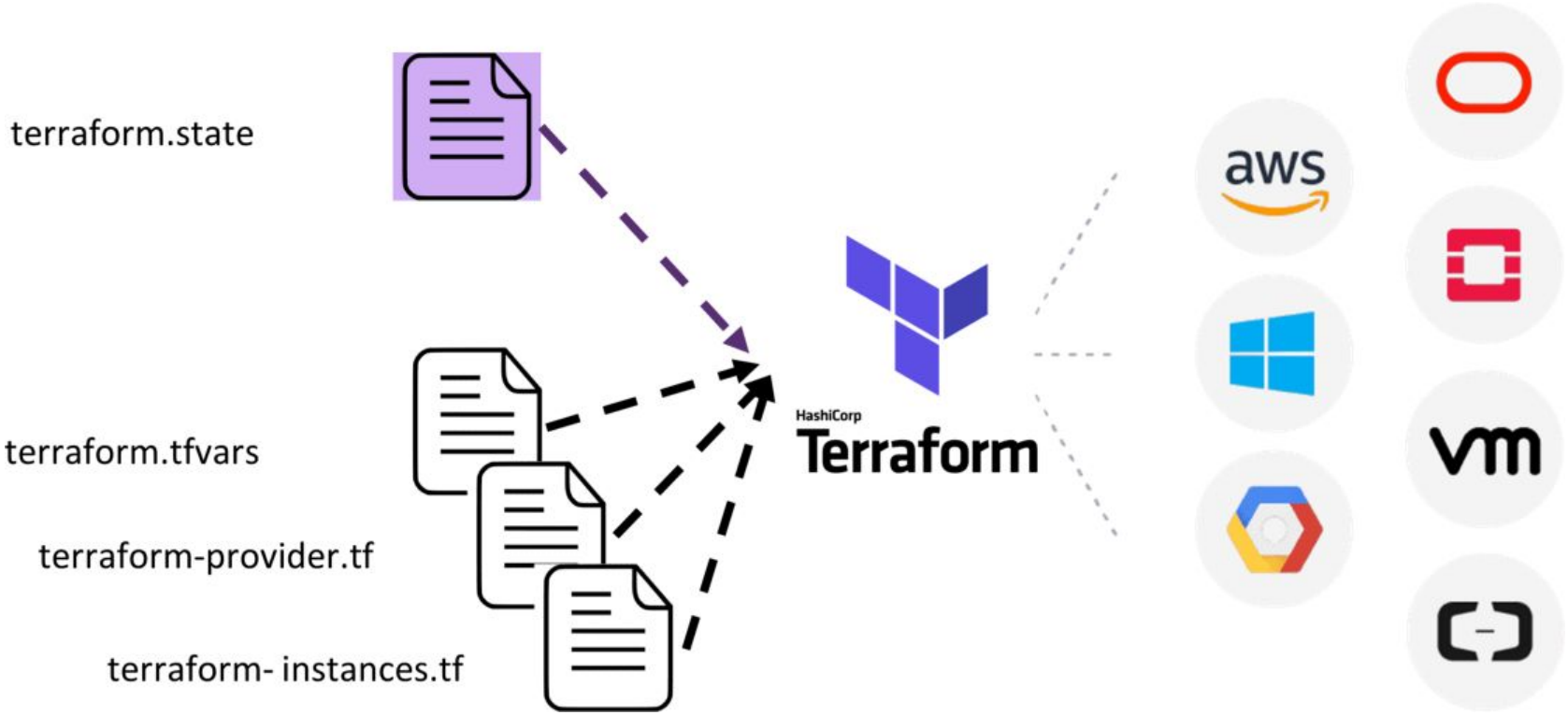


Architecture



SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Architecture



SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Terraform Components

Terraform Executable

Terraform

Providers API

Terraform File

Terraform

Statefile

Terraform config

file

Terraform Executable

Terraform Providers

Terraform Providers

IaaS, PaaS, and SaaS

Community and HashiCorp

- AWS, Azure, GCP, and Oracle

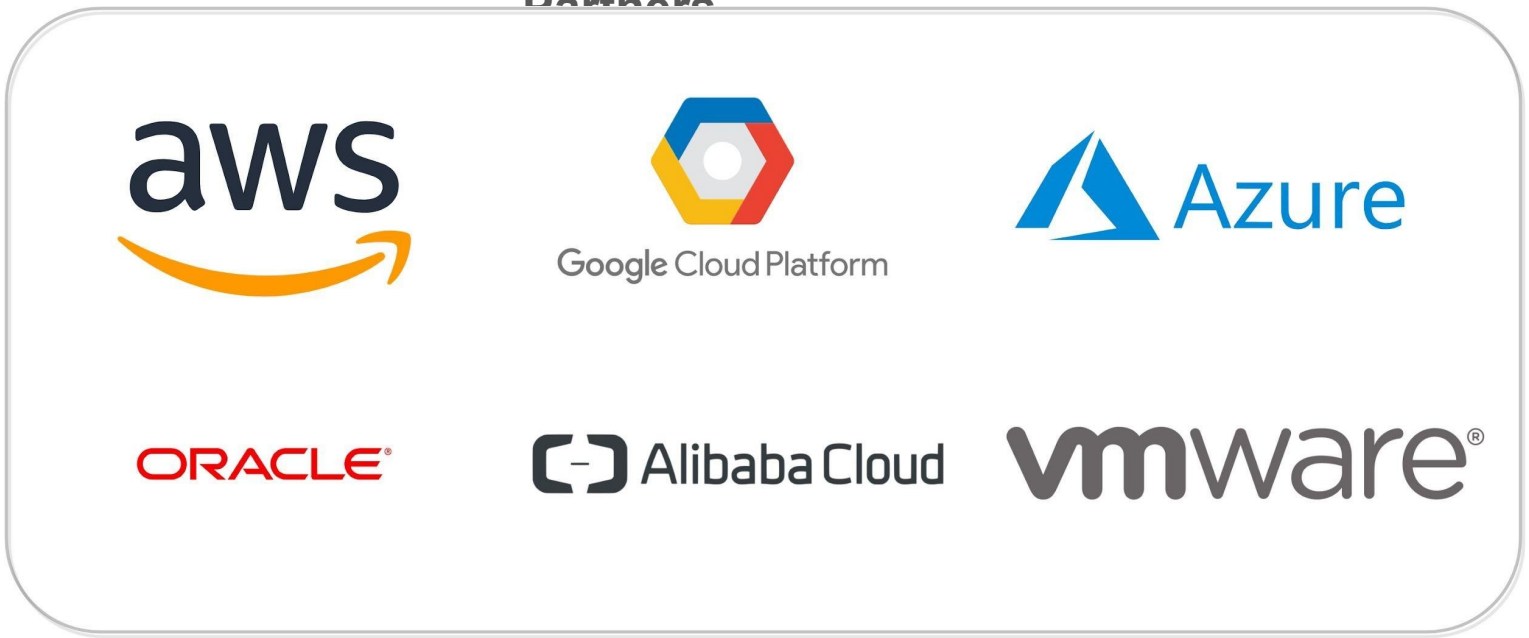
Open source using APIs
Resources and data sources

Multiple instances

Terraform: Providers (Plugins)

1125+ infrastructure providers

Major Cloud Partners



SRM INSTITUTE OF SCIENCE & TECHNOLOGY (Deemed to be University u/s 3 of UGC Act, 1956)

Terraform: Providers

Can be integrated with any API using providers framework

- Note: Terraform Docs → Extending Terraform → Writing Custom Providers

- GitLab
- GitHub
- BitBucket

- OpenFaaS
- **OpenAPI**
- Generic Rest API
- Stateful

- Template
- Random
- Null
- External (escape hatch)
- Archive

- DNS
- Palo Alto Networks
- F5 BIG-IP

- NewRelic
- Datadog
- PagerDuty

- Docker
- Kubernetes
- Nomad
- Consul
- Vault
- Terraform :)

- Digital Ocean
- Fastly
- OpenStack
- Heroku

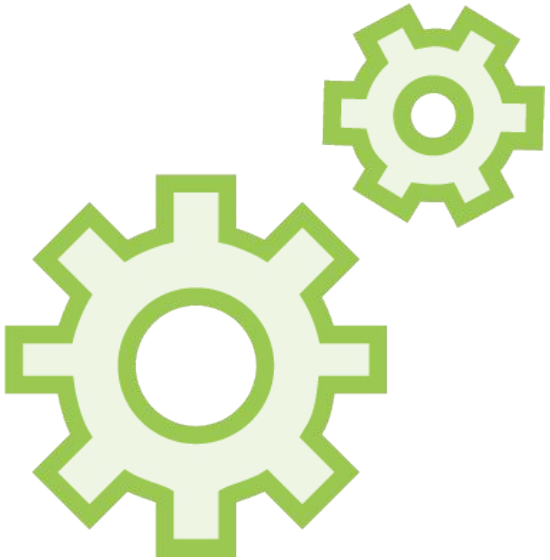
Provider Example

```
provider “azurerm” {
  subscription_id = “subscription-id”
  client_id =
  “principal-used-for-access”
  client_secret =
  “password-of-principal”  tenant_id =
  “tenant-id”
  alias = “arm-1”
}
```

Terraform Code

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Terraform Syntax



HashiCorp configuration language

Why not JSON?

Human readable and editable

Interpolation

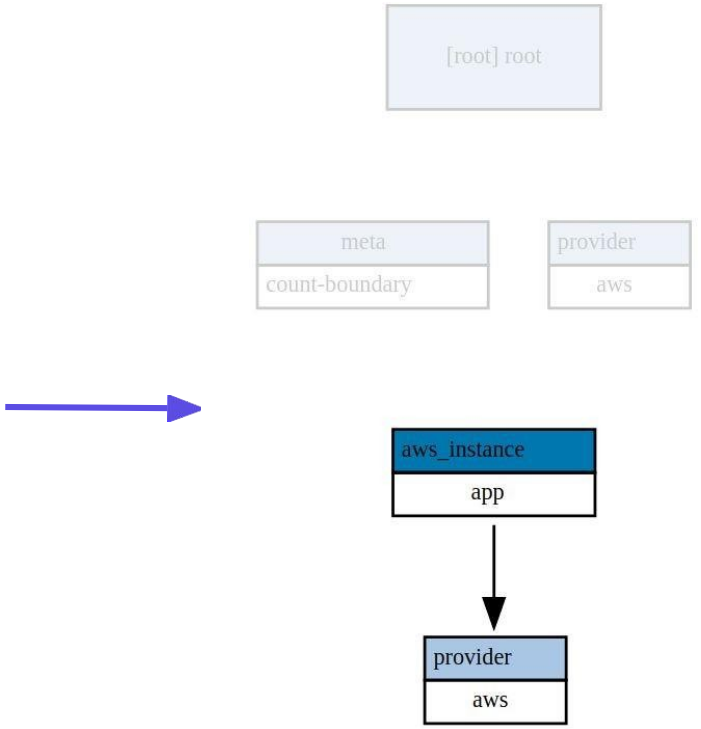
Conditional, functions, templates

Terraform: Example (Simple resource)

Type

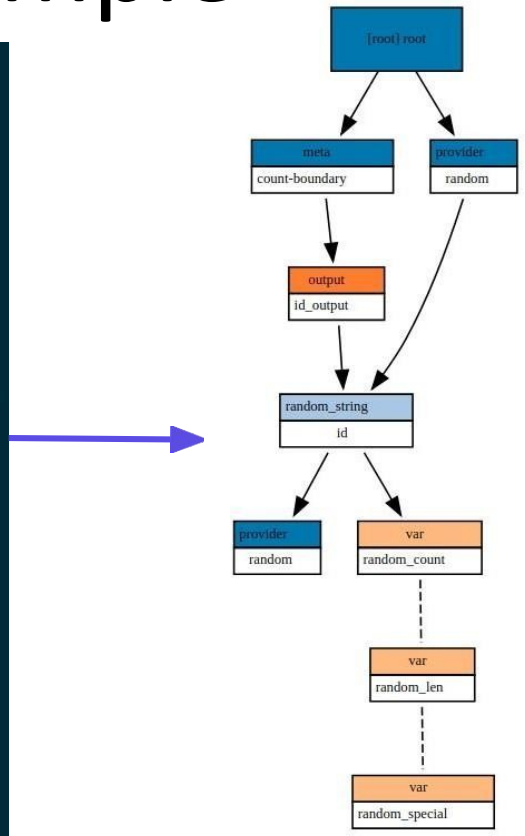
Name

```
resource "aws_instance" "app" {
  ami           = "ami-ab11cd22ee"
  instance_type = "t2.micro"
}
```



Terraform: Example (Simple

```
# file: main.tf
resource "random_string" "id" {
  count            = "${var.random_count}"
  special          = "${var.random_special}"
  length           = "${var.random_len}"
  override_special = "#"
  min_special      = 1
}
# file: outputs.tf
output "id_output" {
  value     = "${formatlist("secret:%s",random_string.id.*.result)}"
  sensitive = false
}
# file: variables.tf
variable "random_count" {
  default = 1
}
variable "random_len" {
  default = 32
}
variable "random_special" {
  default = true
}
```



```
variable "aws_access_key" {}

variable "aws_secret_key" {}


provider "aws" {

  access_key = "access_key"
  secret_key = "secret_key"
  region = "us-east-1"

}
```

- Variables

- **Provider**

```
resource "aws_instance"
"ex"{
  ami = "ami-c58c1dd3"
  instance_type =
  "t2.micro"
}

output "aws_public_ip" {

  value =
  "${aws_instance.ex.public_dns}
  "

}
```

- Resource
- **Output**

Code Example

```
provider “azurerm” {
  subscription_id = “subscription-id”
  client_id =
  “principal-used-for-access”
  client_secret =
  “password-of-principal”  tenant_id =
  “tenant-id”
  alias = “arm-1”
}
resource
  “azurerm_resource_group”{  name
  = “resource-group-name”
  location = “East US”
  provider = “azurerm.arm-1”
```

Terraform Syntax

```
#Create a variable

variable var_name {

 key = value #type, default,
 description

}

#Use a variable

${var.name} #get string

${var.map[“key”]} #get map element

${var.list[idx]} #get list element
```

Terraform Syntax

```
#Create provider

provider provider_name {

 key = value #depends on resource, use alias as
 needed

}

#Create data object
data data_type data_name

{}  #Use data object


${data_type.data_name.attribute(args)}
```

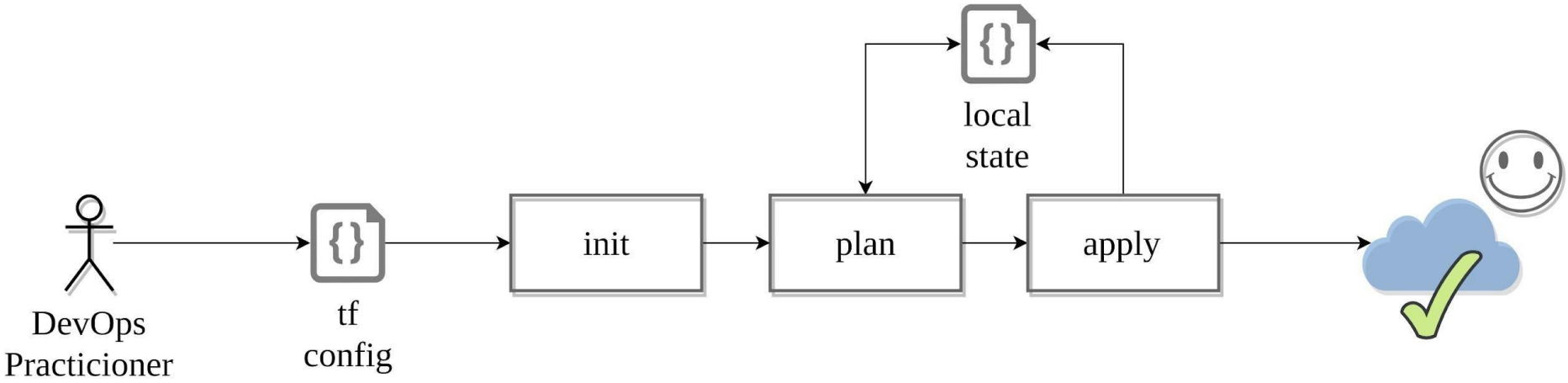
Terraform Syntax

```
#Create resource

resource resource_type resource_name {

 key = value #depends on resource

}

#Reference resource

${resource_type.resource_name.attribute(args)
}
```

Terraform Workflow

Workflow: Adoption stages

Single contributor



SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Terraform Core: Init

1. This command will never delete your existing configuration or state.
2. Checkpoint → https://checkpoint.hashicorp.com/
3. .terraformrc → enable plugin_cache_dir, disable checkpoint
4. Parsing configurations, syntax check
5. Checking for provisioners/providers (by precedence, only once)→ ".", terraform_bin_dir, terraform.d/plugins/linux_amd64 .terraform/plugins/linux_amd64
6. File lock.json contains sha-512 plugin hashes (.terraform)
7. Loading backend config (if it's available, local instead) Backend Initialization: Storage for terraform state file.

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Terraform Core: Plan + Apply

1. Starting Plugins: Provisioners/Providers
2. Building graph
   a. Terraform core traverses each vertex and requests each provider using parallelism
3. Providers syntax check: resource validation
4. If backend == <nil>, use local
5. If “-out file.plan” provided - save to file - the file is not encrypted
6. Terraform Core calculates the difference between the last-known state and the current state
7. Presents this difference as the output of the terraform plan operation to user
   in their terminal

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

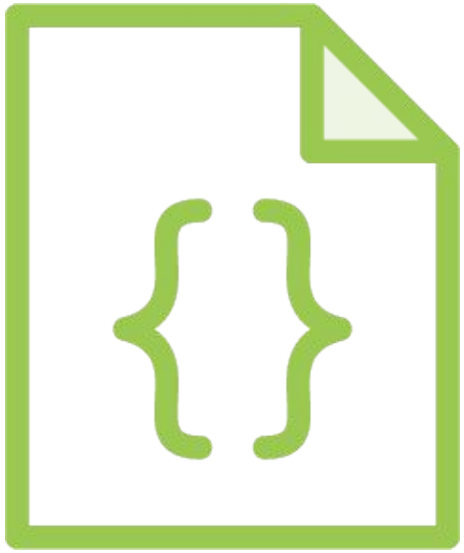
Terraform Core: Destroy

1. Measure twice, cut once
2. Consider -target flag
3. Avoid run on production
4. No “Retain” flag - Remove resource from state file instead
5. terraform destroy tries to evaluate outputs that can refer to non existing

   resources #18026
6. prevent_destroy should let you succeed #3874
7. You can’t destroy a single resource with count in the list

Terraform state

Terraform State



JSON format (Do not touch!)

Resources mappings and metadata

Locking
Local / remote

Environments

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Terraform state file

1. Backup your state files + use Versioning and Encryption
2. Do Not edit manually!
3. Main Keys: cat terraform.tfstate.backup | jq 'keys'
   a. "lineage" - Unique ID, persists after initialization
   b. "modules" - Main section
   c. "serial" - Increment number
   d. "terraform_version" - Implicit constraint
   e. "version" - state format version
4. Use “terraform state” command
   a. mv - to move/rename modules
   b. rm - to safely remove resource from the state. (destroy/retain like)
   c. pull - to observe current remote state
   d. list & show - to write/debug modules

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Terraform State

- Terraform keeps the remote state of the infrastructure
- It stores it in a file called terraform.tfstate
- There is also a backup of the previous state in terraform.tfstate.backup
- When you execute terraform apply, a new terraform.tfstate and

Terraform State

- You can keep the terraform.tfstate in version control
- e.g. git
- It gives you a history of your terraform.tfstate file (which is just a big JSON file)
- It allows you to collaborate with other team members
- Unfortunately you can get conflicts when 2 people work at the same time
- Local state works well in the beginning, but when you project becomes bigger, you might want to store your state remote

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

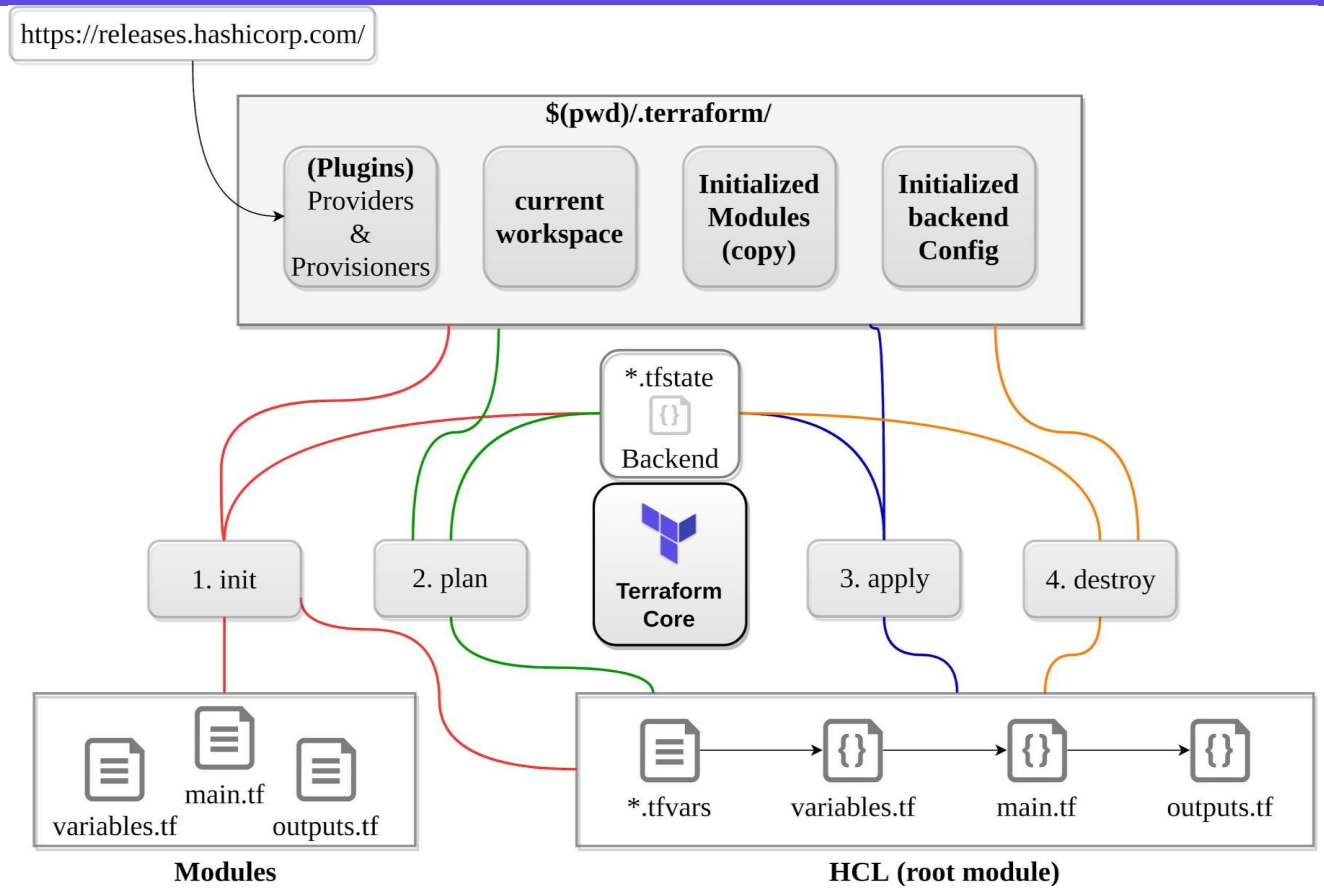
Terraform State

The terraform state can be saved remote, using the backend functionality in terraform.

The default is a local backend (the local terraform state file)
Other backbend's include:

- **s3** (with a looking mechanism using dynamoDB)
- **consul** (with locking)
- **terraform enterprise** (the commercial solution)

Simple workflow



UpdatingYour Configuration with More Resources

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Adding a New Provider to Your Configuration

Terraform Command Overview

Command	Description
terraform apply	Applies state
destroy	Destroys all terraform managed state (use with caution)
fmt	Rewrite terraform configuration files to a canonical format and style
get	Download and update modules
graph	Create a visual representation of a configuration or execution plan
import [options] ADDRESS ID	Import will try and find the infrastructure resource identified with ID and import the state into terraform.tfstate with resource id ADDRESS

SRM

Terraform Command Overview

Command	Description
output [options] [NAME]	Output any of your resources. Using NAME will only output a specific resource
plan	terraform plan, show the changes to be made to the infrastructure
push	Push changes to Atlas, Hashicorp's Enterprise tool that can automatically run terraform from a centralized server
refresh	Refresh the remote state. Can identify differences between state file and remote state
remote	Configure remote state storage
show	Show human readable output from a state or a plan

Terraform Command

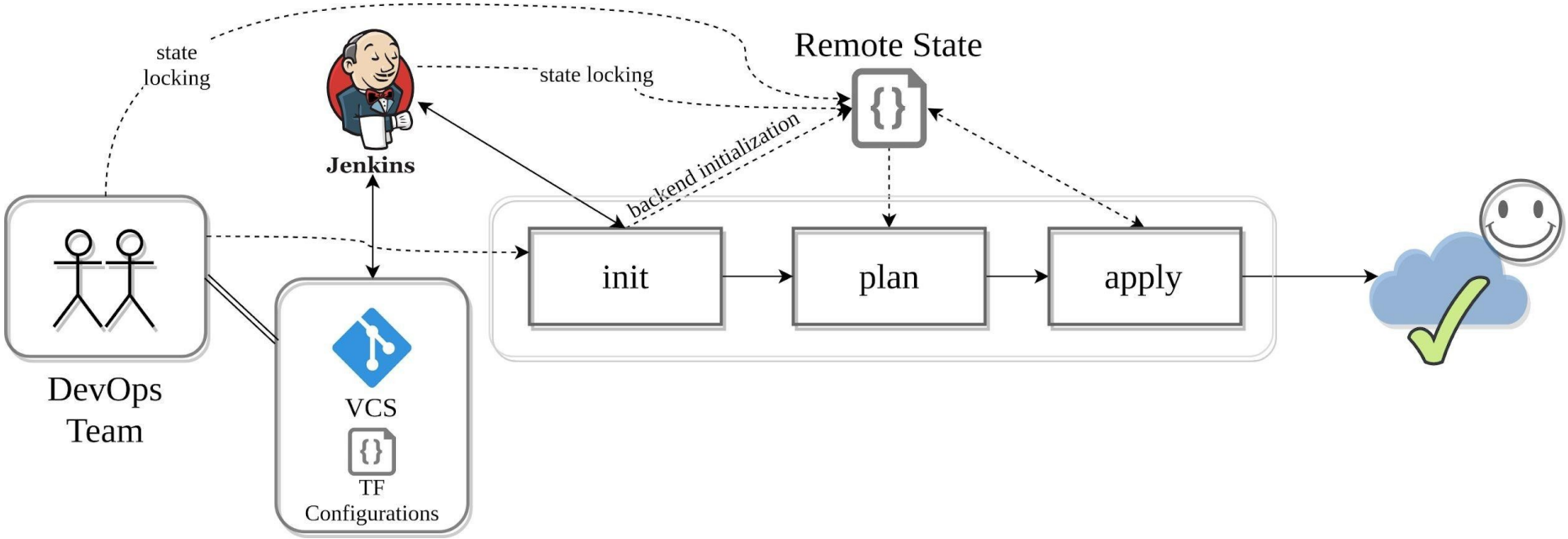
Command	Description
state	Use this command for advanced state management, e.g. Rename a resource with terraform state mv aws_instance.example aws_instance.production
taint	Manually mark a resource as tainted, meaning it will be destructed and recreated at the next apply
validate	validate your terraform syntax
untaint	undo a taint

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Terraform Advance Workflow

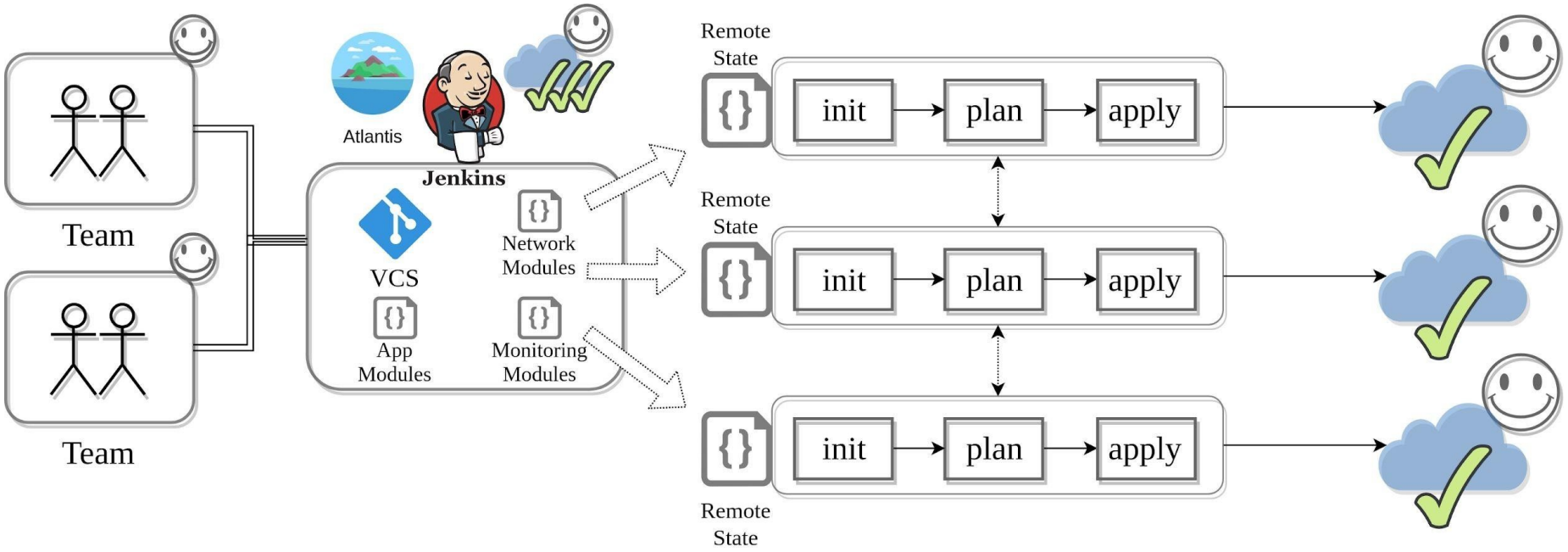
Workflow: Adoption stages

Team Collaboratio



Workflow: Adoption stages

Multiple Teams



SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Demo

Examine the Terraform file

Deploy the configuration

Review the results

Play along!

- AWS account
- Demo files

Dem
o

Examine the Terraform file

Deploy theconfiguration

Review theresults Play along!

- AWS account
- Azure subscription
- DNS domain
- Terraform software (terraform.io)
- Demo files

Demo

Examine the Terraform file Deploy the configuration

Review the results

Play along!

- AWS account
- Terraform software (terraform.io)
- Demo files

Demo

Examine the Terraform file Deploy the configuration

Review the results

Play along!

- AWS account
- Terraform software (terraform.io)
- Demo files

- Ansible

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Why Ansible?

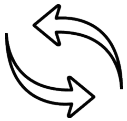
Simple

Human readable automation
No special coding skills needed

Tasks executed in order

Usable by every team

Get productive quickly



Powerful

App deployment
Configuration management

Workflow orchestration Network

automation **Orchestrate the**

app lifecycle

Agentless

Agentless architecture

Uses OpenSSH & WinRM
No agents to exploit or update

Get started immediately

More efficient & more secure

With Ansible you can automate:

CROSS PLATFORM – Linux, Windows, UNIX

Agentless support for all major OS variants, physical, virtual, cloud and network

HUMAN READABLE – YAML

Perfectly describe and document every aspect of your application environment

PERFECT DESCRIPTION OF APPLICATION

Every change can be made by playbooks, ensuring everyone is on the same page

VERSION CONTROLLED

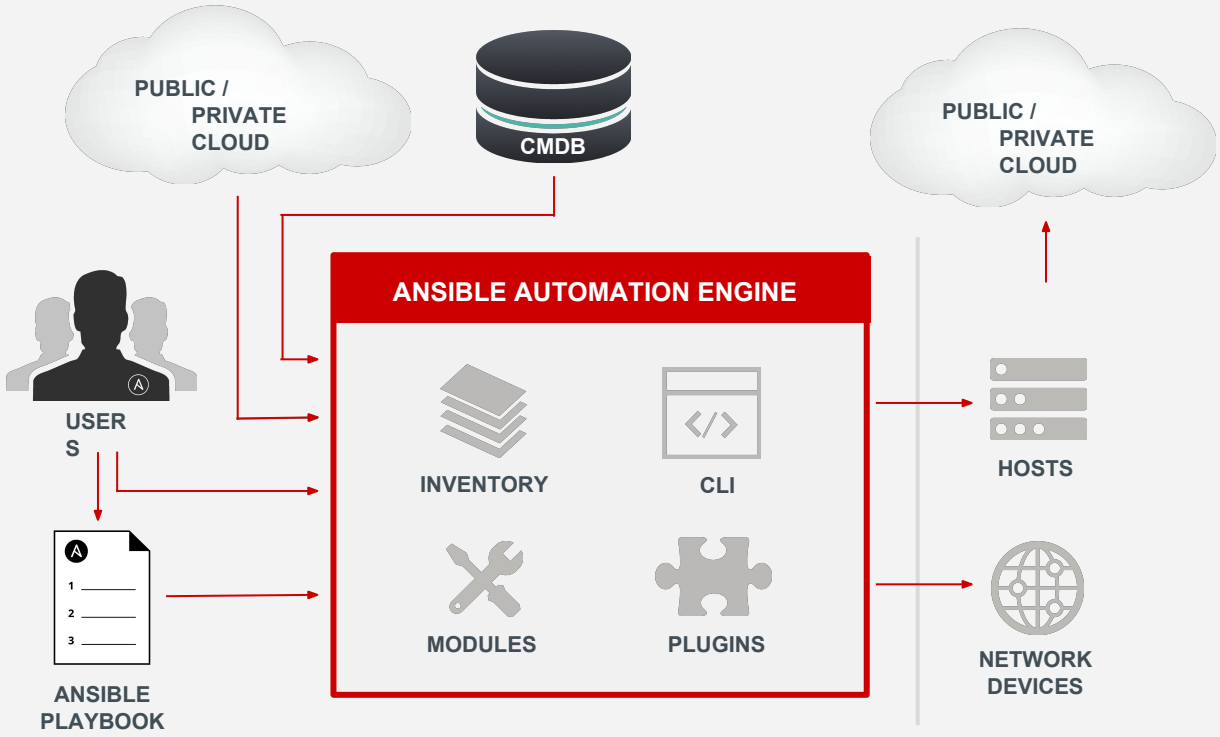
Playbooks are plain-text. Treat them like code in your existing version control.

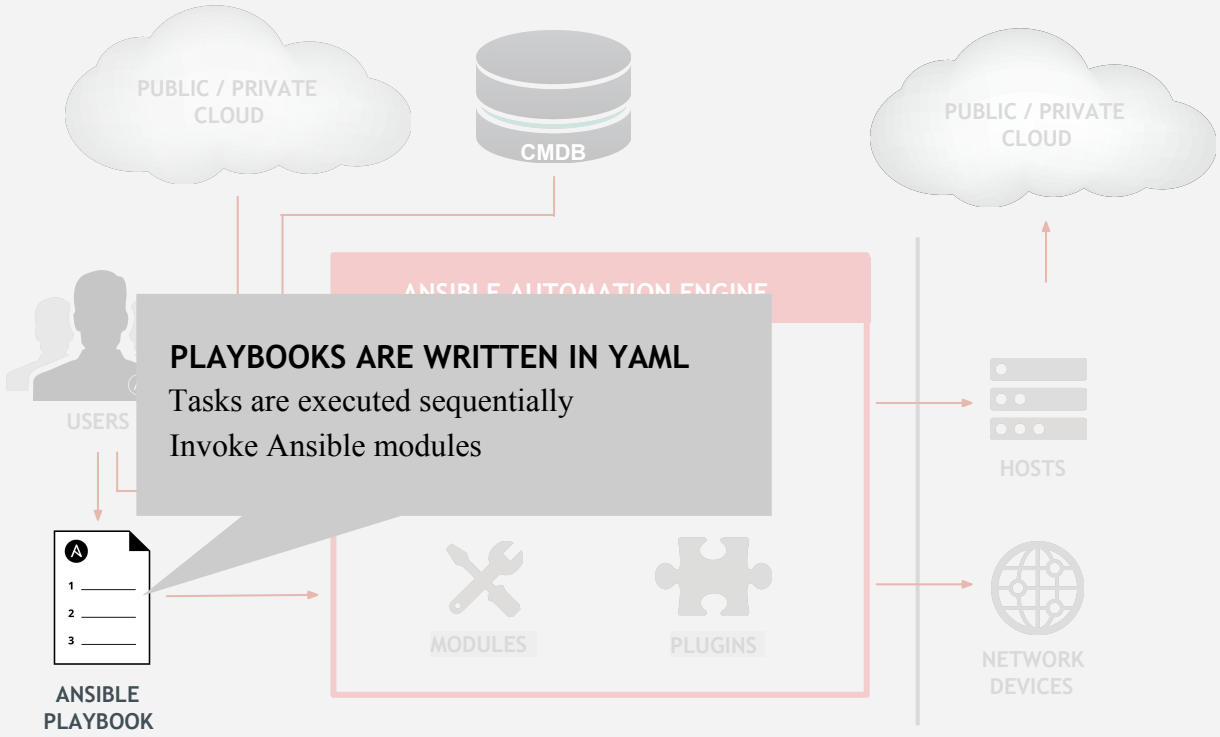
DYNAMIC INVENTORIES

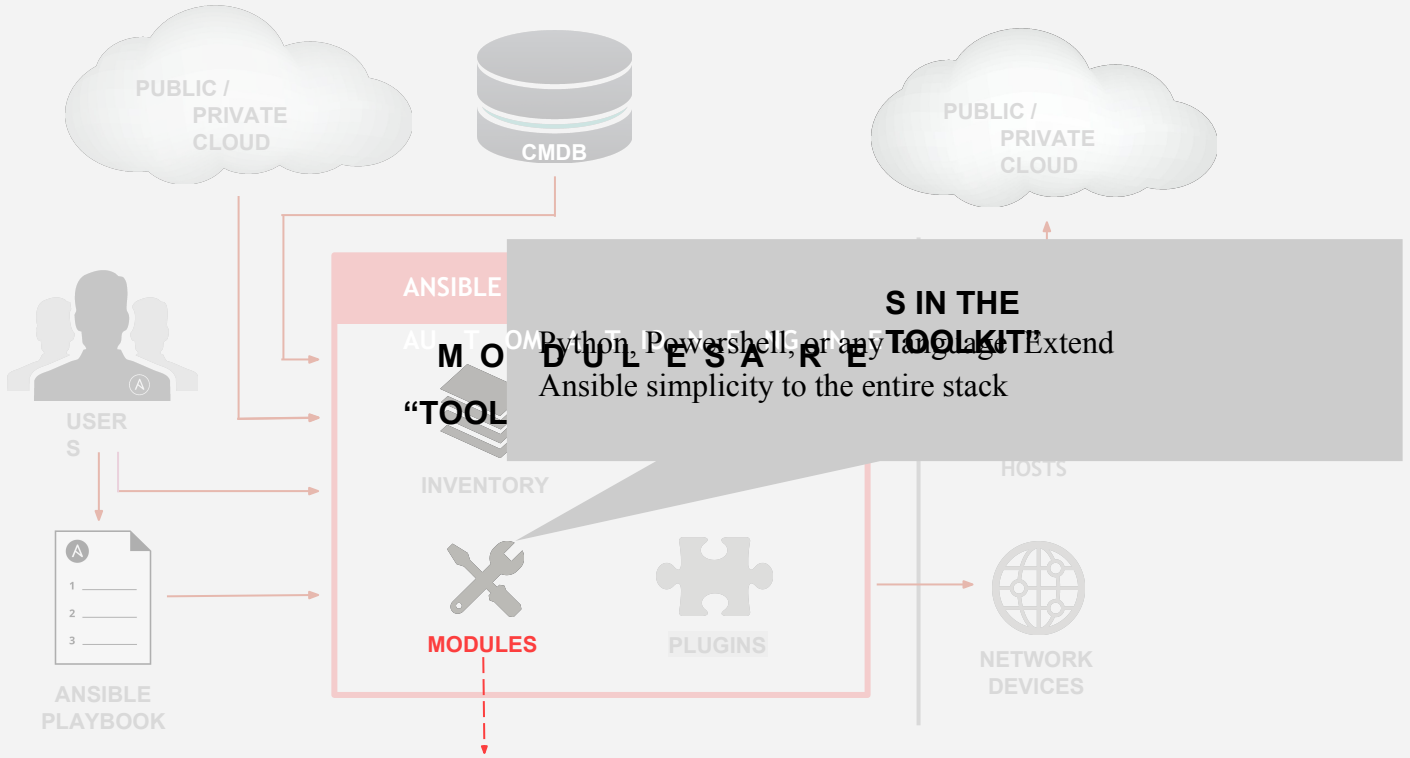
Capture all the servers 100% of the time, regardless of infrastructure, location, etc.

ORCHESTRATION THAT PLAYS WELL WITH OTHERS – HP SA, Puppet, Jenkins, RHNSS, etc.

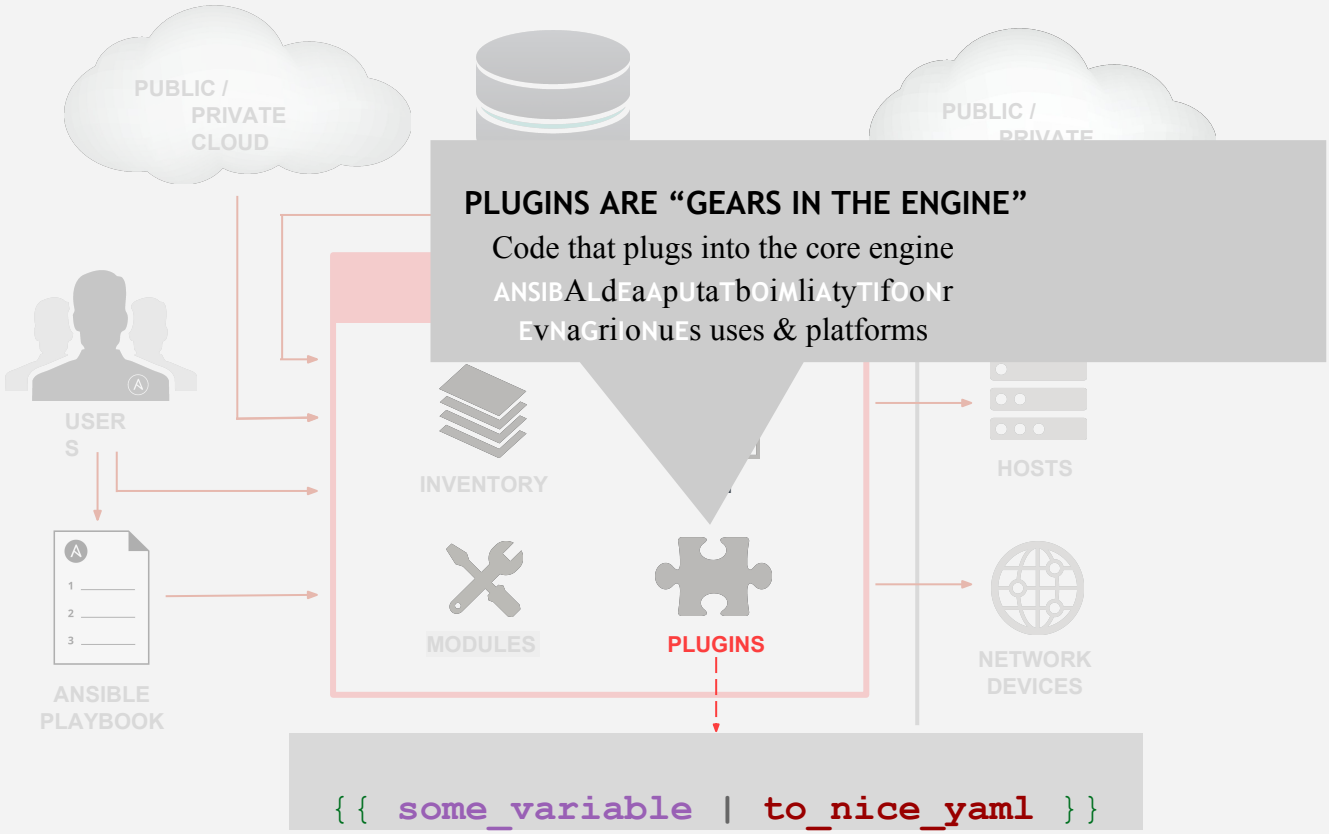
Homogenize existing environments by leveraging current toolsets and update mechanisms.


PUBLIC / PRIVATE CLOUD
CMDB
PUBLIC / PRIVATE CLOUD
ANSIBLE AUTOMATION ENGINE
USER S
INVENTORY
CLI
HOSTS
MODULES
PLUGINS
NETWORK DEVICES
ANSIBLE PLAYBOOK


PUBLIC / PRIVATE CLOUD
CMDB
PUBLIC / PRIVATE CLOUD
USERS
HOSTS
MODULES
PLUGINS
NETWORK DEVICES
ANSIBLE PLAYBOOK
PLAYBOOKS ARE WRITTEN IN YAML
Tasks are executed sequentially
Invoke Ansible modules



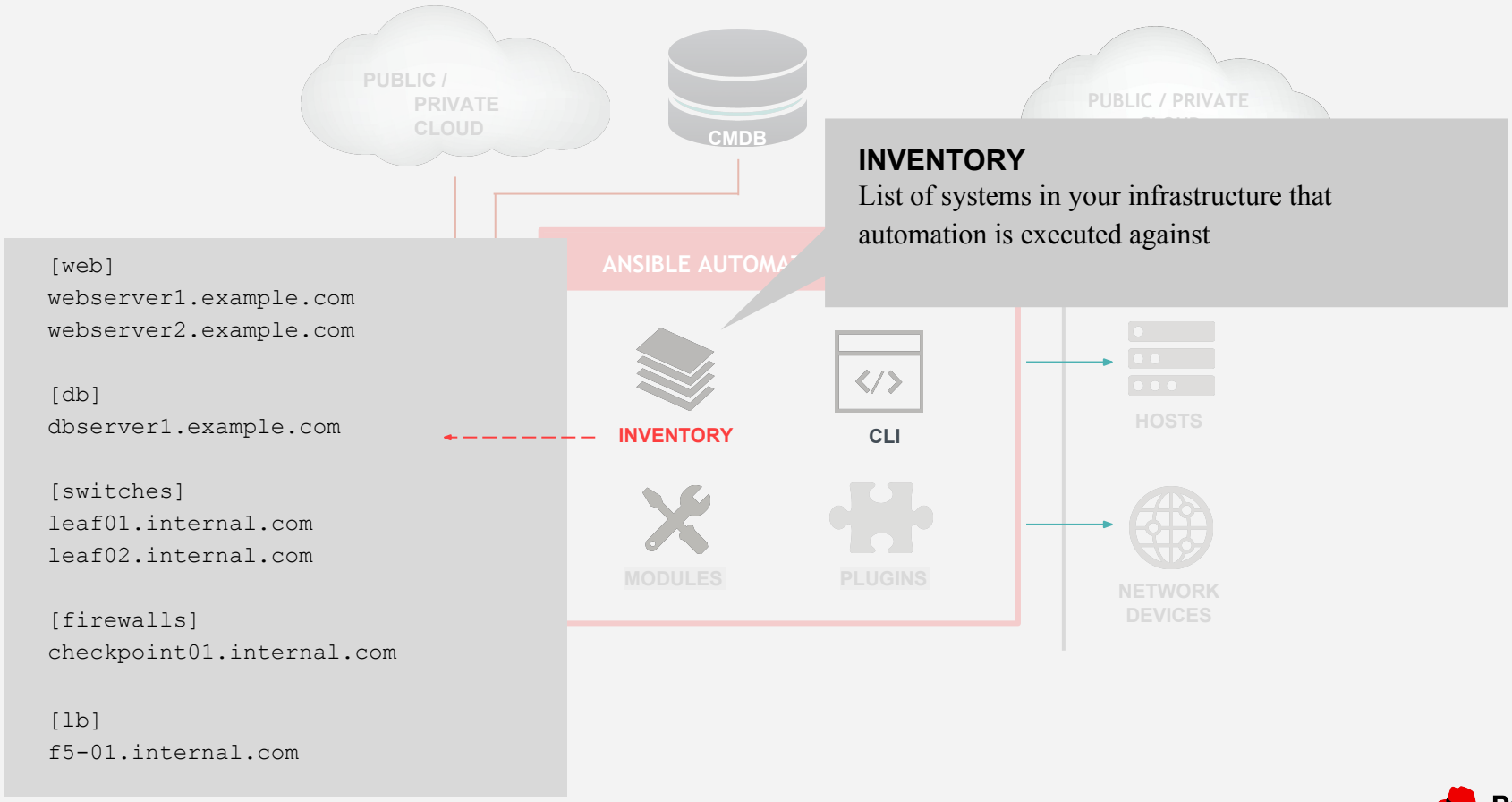

```
- name: latest index.html file is present
  template:
    src: files/index.html
    dest: /var/www/html/
```

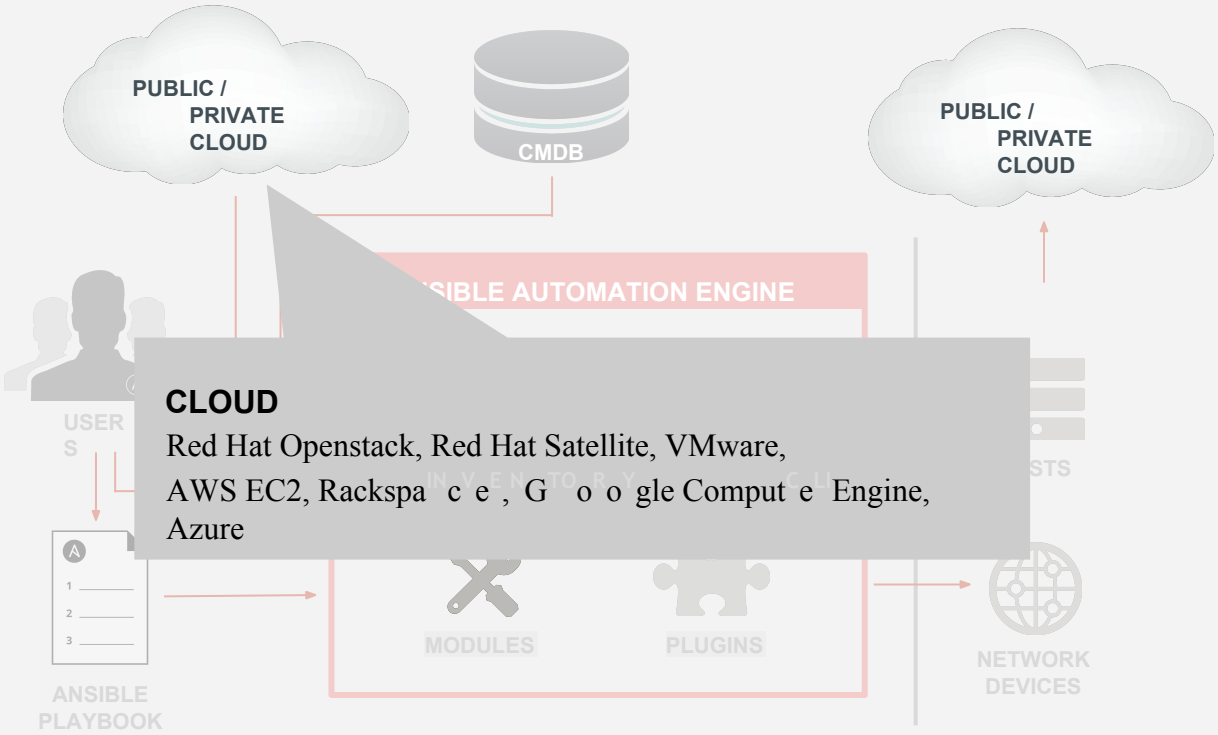


PLUGINS ARE "GEARS IN THE ENGINE"

Code that plugs into the core engine

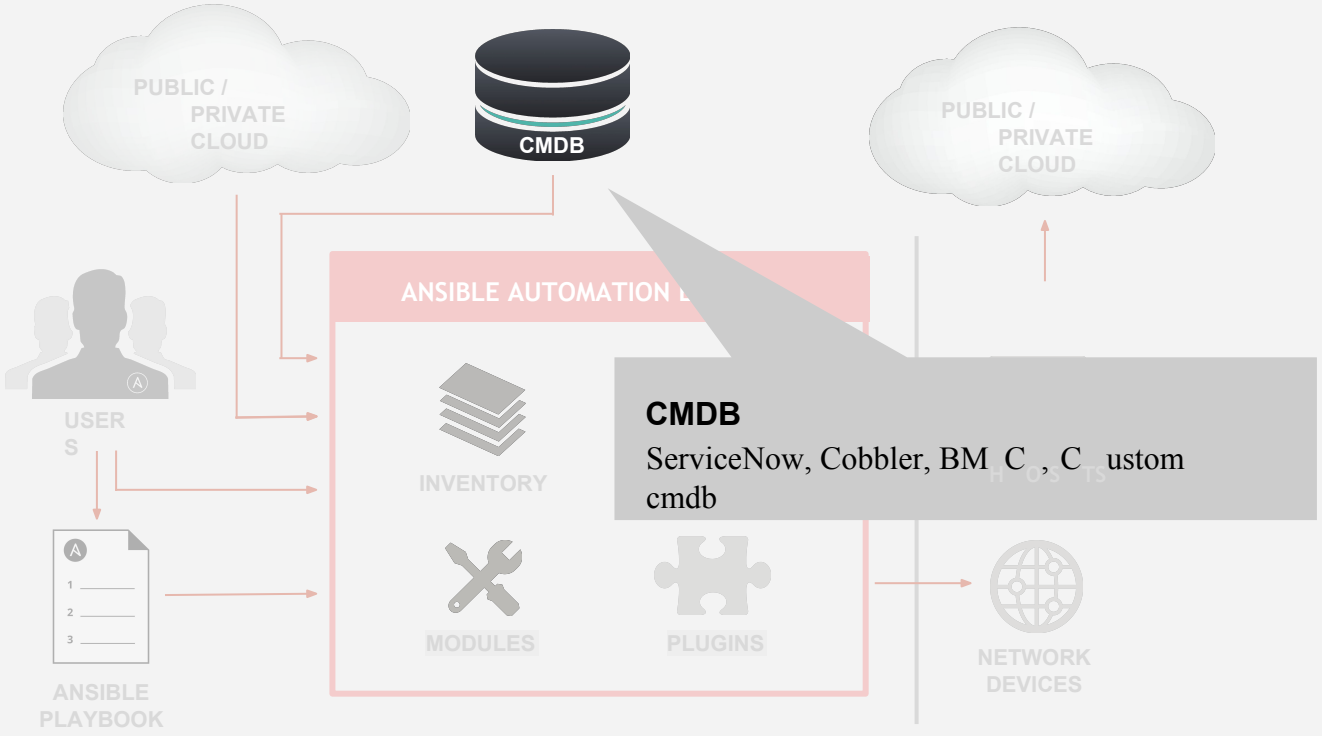
Adaptability for various uses & platforms

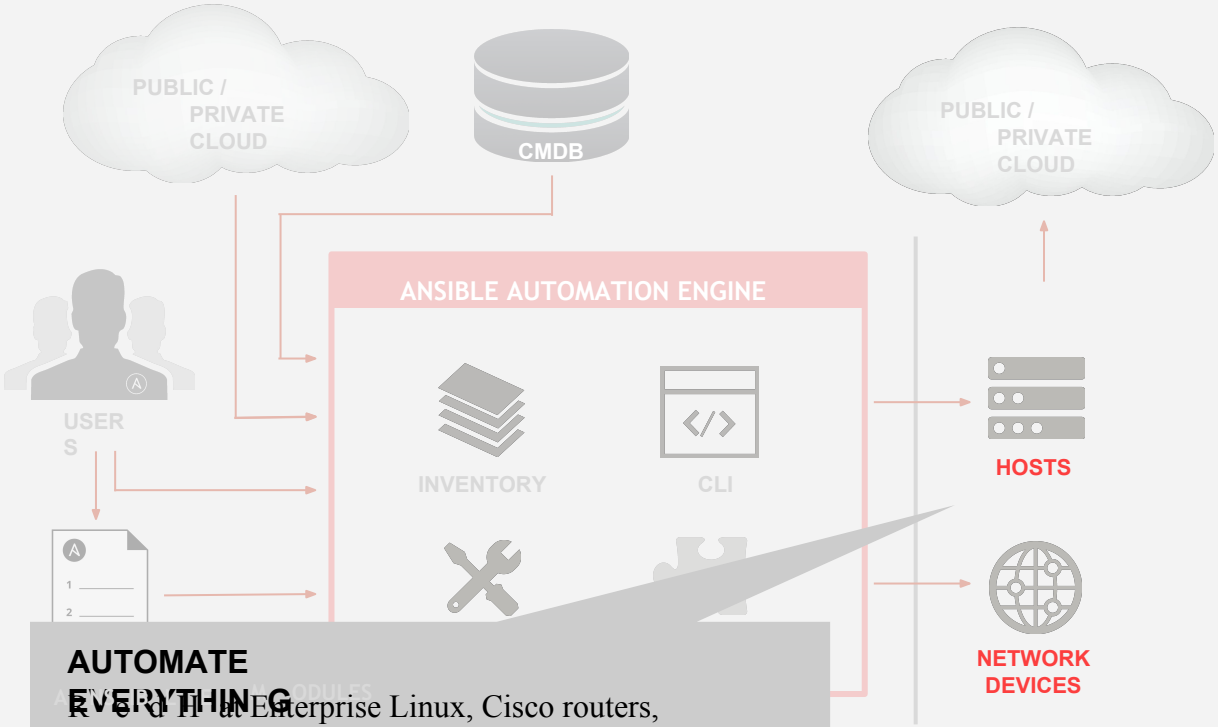

PUBLIC / PRIVATE CLOUD
CMDB
PUBLIC / PRIVATE
INVENTORY
List of systems in your infrastructure that automation is executed against
[web]
webserver1.example.com
webserver2.example.com
[db]
dbserver1.example.com
[switches]
leaf01.internal.com
leaf02.internal.com
[firewalls]
checkpoint01.internal.com
[lb]
f5-01.internal.com
ANSIBLE AUTOMA
INVENTORY
CLI
HOSTS
MODULES
PLUGINS
NETWORK DEVICES



CLOUD

Red Hat Openstack, Red Hat Satellite, VMware,
AWS EC2, Rackspace, Google Compute Engine,
Azure


PUBLIC / PRIVATE CLOUD
CMDB
PUBLIC / PRIVATE CLOUD
ANSIBLE AUTOMATION
USERS
INVENTORY
CMDB
ServiceNow, Cobbler, BMC, Custom cmdb
MODULES
PLUGINS
NETWORK DEVICES
ANSIBLE PLAYBOOK


PUBLIC / PRIVATE CLOUD
CMDB
PUBLIC / PRIVATE CLOUD
ANSIBLE AUTOMATION ENGINE
USER S
INVENTORY
CLI
HOSTS
NETWORK DEVICES
AUTOMATE EVERYTHING
Red Hat Enterprise Linux, Cisco routers, switches, Juniper routers, Windows hosts, Checkpoint firewalls, NetApp storage, F5 load balancers and more

Using Ansible

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Ad-hoc commands

```
# check all my inventory hosts are ready to be
# managed by Ansible
$ ansible all -m ping

# run the uptime command on all hosts in the
# web group
$ ansible web -m command -a “uptime”

# collect and display the discovered for the
# localhost
$ ansible localhost -m setup
```

Inventory

An inventory is a file containing:

- Hosts
- Groups
- Inventory-specific data (variables)
- Static or dynamic sources

Ansible Playbooks

```
---
- name: install and start apache
  hosts: web
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root

  tasks:
  - name: install httpd
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: start httpd
    service: name=httpd state=started
```

```
---
- name: install and start apache
  hosts: web
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root

  tasks:
  - name: install httpd
      yum: pkg=httpd state=latest
  - name: write the apache config file
      template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: start httpd
      service: name=httpd state=started
```

```
---
- name: install and start apache
  hosts: web
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root

  tasks:
  - name: install httpd
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: start httpd
    service: name=httpd state=started
```

```
---
- name: install and start apache
  hosts: web
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root

  tasks:
  - name: install httpd
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: start httpd
    service: name=httpd state=started
```

```
---
- name: install and start apache
  hosts: web
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root

  tasks:
  - name: install httpd
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: start httpd
    service: name=httpd state=started
```

```
---
- name: install and start apache
  hosts: web
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root

  tasks:
  - name: install httpd
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  - name: start httpd
    service: name=httpd state=started
```

```
tasks:
  - name: add cache
    dir  file:
      path: /opt/cache
      state: directory

  - name: install nginx
    yum:
      name: nginx
      state: latest
    notify: restart nginx

handlers:
  - name: restart nginx
    service:
      name: nginx
      state: restarted
```

Variables

Ansible can work with metadata from various sources and manage their context in the form of variables.

- Command line parameters
- Plays and tasks
- Files
- Inventory
- Discovered facts
- Roles

Tips/Best Practices

Simplicity

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Simplicity

```
- hosts: web
  tasks:
  - yum:
      name: httpd
      state: latest

  - service:
      name: httpd
      state: started
      enabled: yes
```

Simplicity

```
- hosts: web
  name: install and start
  apache  tasks:
    - name: install apache
      packages  yum:
        name: httpd
        state: latest

    - name: start apache
      service  service:
        name: httpd
        state: started
        enabled: yes
```

Naming example

Inventory

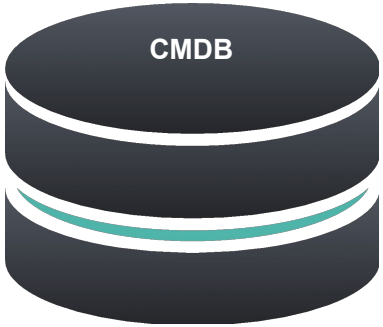
```
10.1.2.75
10.1.5.45
10.1.4.5
10.1.0.40

w14301.example.com
w17802.example.com
w19203.example.com
w19304.example.com
```

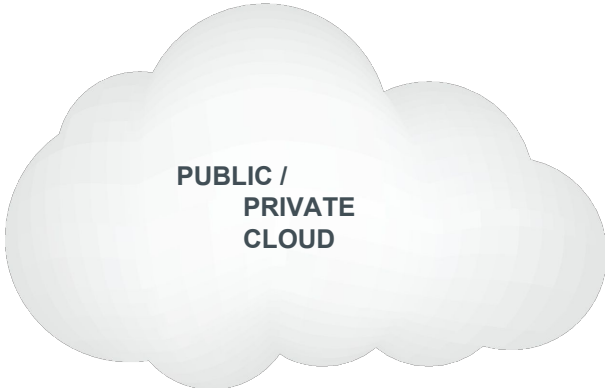
Inventory

```
db1 ansible_host=10.1.2.75
db2 ansible_host=10.1.5.45
db3 ansible_host=10.1.4.5
db4 ansible_host=10.1.0.40

web1 ansible_host=w14301.example.com
web2 ansible_host=w17802.example.com
web3 ansible_host=w19203.example.com
web4 ansible_host=w19203.example.com
```

Dynamic Inventories

- Stay in sync automatically
- Reduce human error





YAML Syntax

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

YAML and Syntax

```
- name: install telegraf
  yum: name=telegraf-{{ telegraf_version }} state=present
update_cache=yes  disable_gpg_check=yes enablerepo=telegraf
  notify: restart telegraf

- name: configure telegraf
  template: src=telegraf.conf.j2 dest=/etc/telegraf/telegraf.conf

- name: start telegraf
  service: name=telegraf state=started enabled=yes
```

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

YAML and Syntax

```
- name: install telegraf
  yum: >
      name=telegraf-{{ telegraf_version
      }}  state=present
      update_cache=yes
      disable_gpg_check=yes
      enablerepo=telegraf
  notify: restart telegraf

- name: configure telegraf
  template: src=telegraf.conf.j2
  dest=/etc/telegraf/telegraf.conf

- name: start telegraf
  service: name=telegraf state=started enabled=yes
```

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

YAML and Syntax

```
- name: install telegraf
  yum:
    name: telegraf-{{ telegraf_version
    }}  state: present
    update_cache: yes
    disable_gpg_check: yes
    enablerepo: telegraf
  notify: restart telegraf

- name: configure telegraf
  template:
    src: telegraf.conf.j2
    dest: /etc/telegraf/telegraf.conf
  notify: restart telegraf

- name: start telegraf
  service:
    name: telegraf
    state: started
    enabled: yes
```

ansible-playbook playbook.yml --syntax-check

Roles

Roles

- Think about the full life-cycle of a service, microservice or container — not a whole stack or environment
- Keep provisioning separate from configuration and app deployment
- Roles are not classes or object or libraries – those are programming constructs
- Keep roles loosely-coupled — limit hard dependencies on other roles or external variables

Variable Precedence

The order in which the same variable from different sources will override each other.

Variable Precedence

1. Extra vars
2. Include params
3. Role (and include_role) params
4. Set_facts / registered vars
5. Include_vars
6. Task vars (only for the task)
7. Block vars (only for tasks in the block)
8. Role vars
9. Play vars_files
10. Play vars_prompt
11. Play vars
12. Host facts / Cached set_facts
13. Playbook host_vars
14. Inventory host_vars
15. Inventory file/script host vars
16. Playbook group_vars
17. Inventory group_vars
18. Playbook group_vars/all
19. Inventory group_vars/all
20. Inventory file or script group vars
21. Role defaults
22. Command line values (*e.g.*, -u user)

Things to Avoid

Things to Avoid

- Using command modules
  - Things like shell, raw, command etc.
- Complex tasks...at first
  - Start small
- Not using source control
  - But no really...

Ansible Content Collections

Collections Q and A

What are they?

- Collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins. You can install and use collections through Ansible Galaxy and Automation Hub

How do I get them?

- ansible-galaxy collection install namespace.collection -p /path Where can I get them?
- Today
  - Galaxy
  - Automation Hub

Collection Directory Structure

- **docs**/: local documentation for the collection
- **galaxy.yml**: source data for the MANIFEST.json that will be part of the collection package
- **playbooks**/: playbook snippets
  - **tasks**/: holds 'task list files' for include_tasks/import_tasks usage
- **plugins**/: all ansible plugins and modules go here, each in its own subdir
  - **modules**/: ansible modules
  - **lookups**/: lookup plugins
  - **filters**/: Jinja2 filter plugins
  - **connection**/: connection plugins required if not using default
- **roles**/: directory for ansible roles
- **tests**/: tests for the collection's content

SRM

Collections: Let's Go!

1. Init collection: `ansible-galaxy collection init foo.bar`
2. Sanity testing: `ansible-test sanity`
3. Unit tests: `ansible-test units`
4. Integration tests: `ansible-test integration`
5. Build the collection: `ansible-galaxy collection build`
6. Publish the collection: `ansible-galaxy collection publish`
7. Install the collection: `ansible-galaxy collection install foo.bar`

Red Hat

Resource Link Index

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable
https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#using-variables
https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html
https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html
https://docs.ansible.com/ansible/latest/user_guide/intro_getting_started.html#getting-started
https://docs.ansible.com/ansible/latest/user_guide/intro_adhoc.html
https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html
https://docs.ansible.com/ansible/latest/index.html
https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html
https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.htm
l https://docs.ansible.com/ansible-lint/
https://github.com/ansible/ansible
https://github.com/ansible/ansible-lint
https://ansible.github.io/workshops/
https://www.ansible.com/resources/ebooks/get-started-with-red-hat-ansible-towe
r https://docs.ansible.com/ansible/latest/user_guide/collections_using.html
https://docs.ansible.com/ansible/latest/dev_guide/developing_collections.html