HANA EXPERT SERIES

HANA Heroes Expert Session

Title: Automating SAP Deployments with Terraform and Ansible

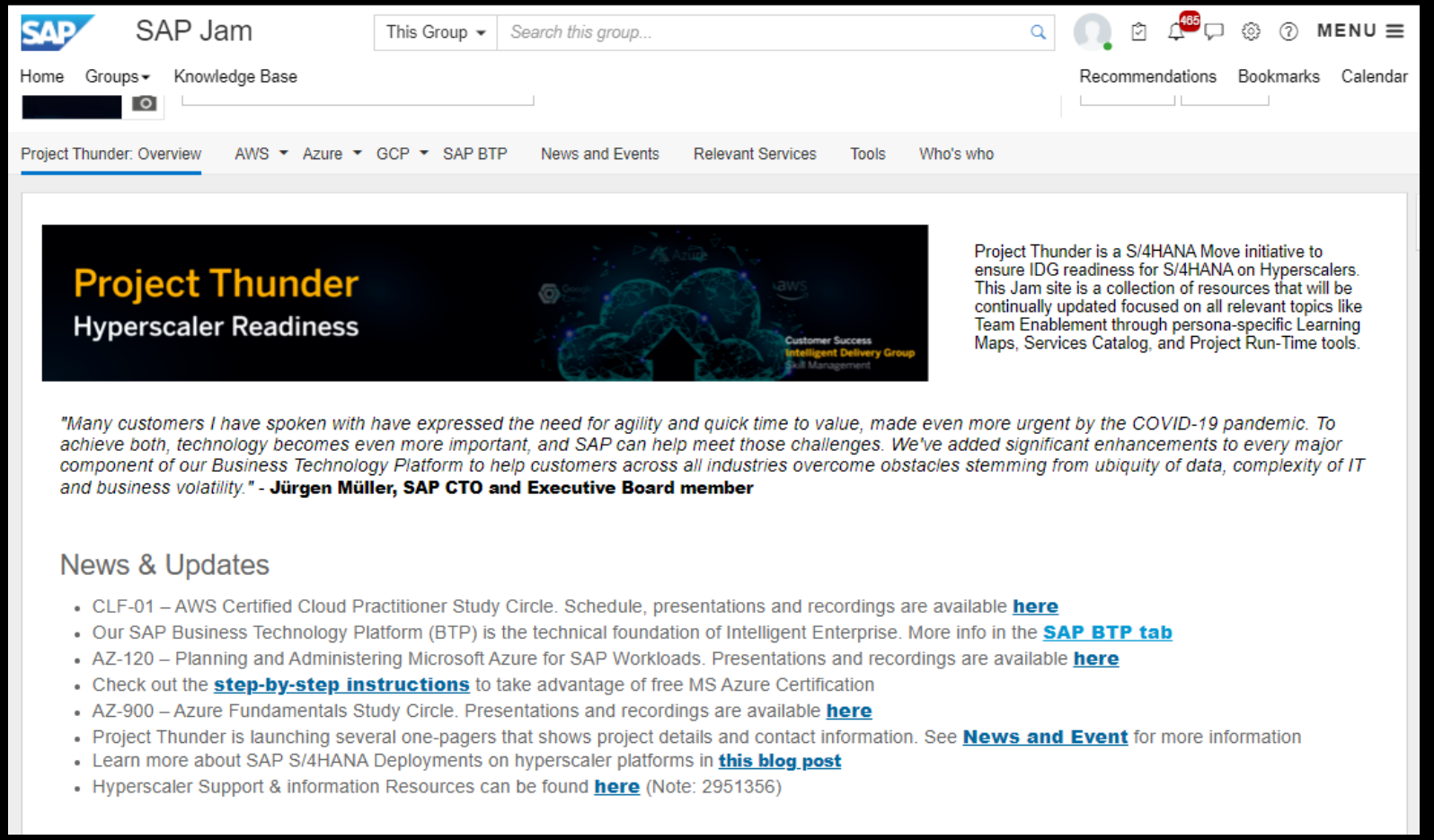
Presenter: Darpan Patel

Proudly presented by **SAP HANA Heroes**

Kick Off Automated SAP S/4HANA Build

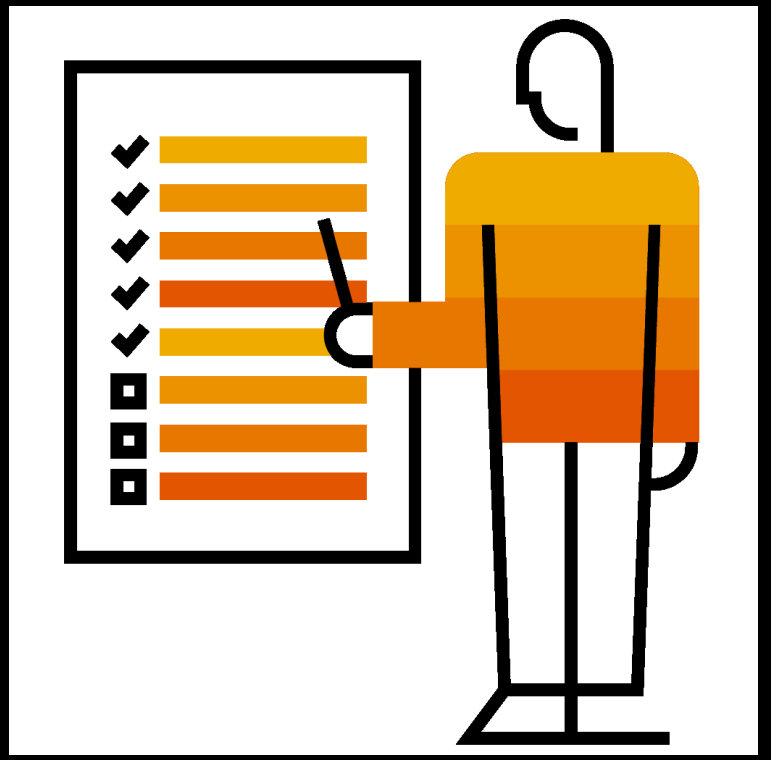
Project Thunder

- Join here if you are not a member so you don't miss updates regarding this topic and many more!
- Blog Post on this topic will be released beginning of Q2



Agenda

- Why learn these tools?
- Getting Started with Terraform
- Getting Started with Ansible
- Merging Terraform and Ansible



Why Terraform and Ansible?

SAP Implementations

- Implementation Issues:
  - Technical Implementations can become very lengthy
  - They are a bottleneck for the project
  - Involve a very manual process
    - Both Provisioning servers and applications
  - Outdated in today's quick paced and automated world
- How do we make implementations more efficient?
  - Automate
    - Provisioning Servers
    - Resources that attach onto those servers
    - The networks they reside in
    - OS configurations
    - Application Deployments

HashiCorp Terraform

- Infrastructure as Code
- Use to Deploy the Hosts for SAP application servers
- Database server hosts
- Storage
- Networks, VPC’s, Subnets, routers
- Security Groups, NACL’s

RED HAT® ANSIBLE® Automation

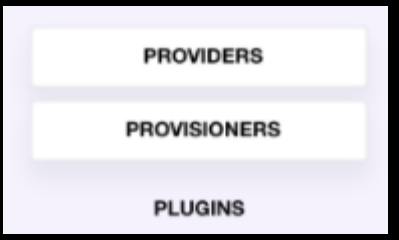
- Deploy applications
- Install Packages for OS
- Configure your file system
- Mount Network locations
- Install SAP HANA
- Any task SWPM can do
- System Copy, System Rename, Distributed System, Standard System

Getting Started with Terraform

Terraform Implementation Considerations

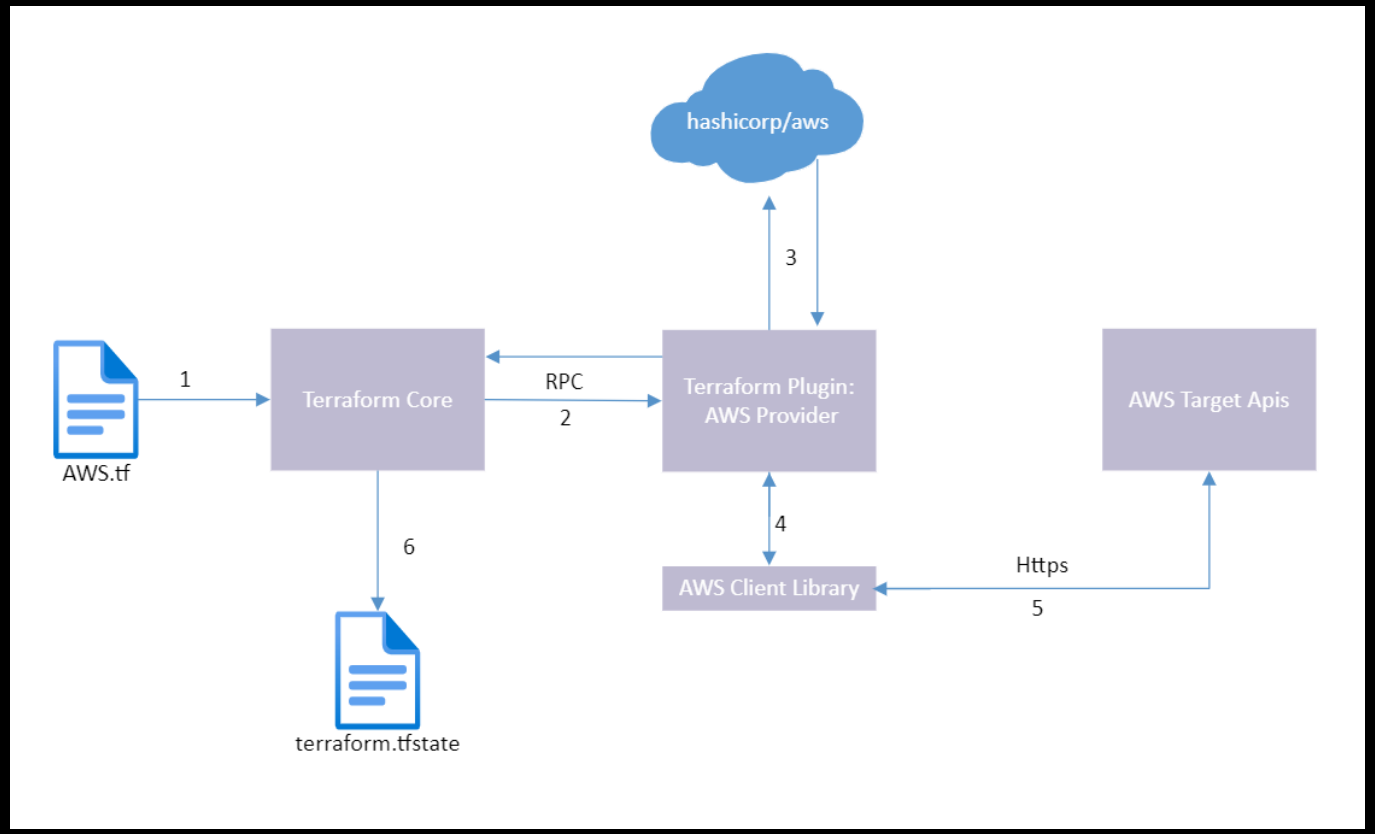
- Terraform is logically split into two main parts:
  - Terraform Core
  - Terraform Plugins
- Terraform is Agentless
- Terraform can be installed on any platform
- Installation Options
  - Manual
  - OS Package Manager
- Terraform Core and Plugins are written in the GO Language
- Terraform uses CRUD (create, read, update, and delete) API's to communicate with providers

- Statically-compiled binary written in GO
- Terraform core is the command line tool
- Entry point for anyone using terraform
- Resource State Management
- Communicates with plugins



- Terraform plugins are executable binaries written in GO
- Plugins contain Providers and Provisioners
- Providers provide a service such as AWS

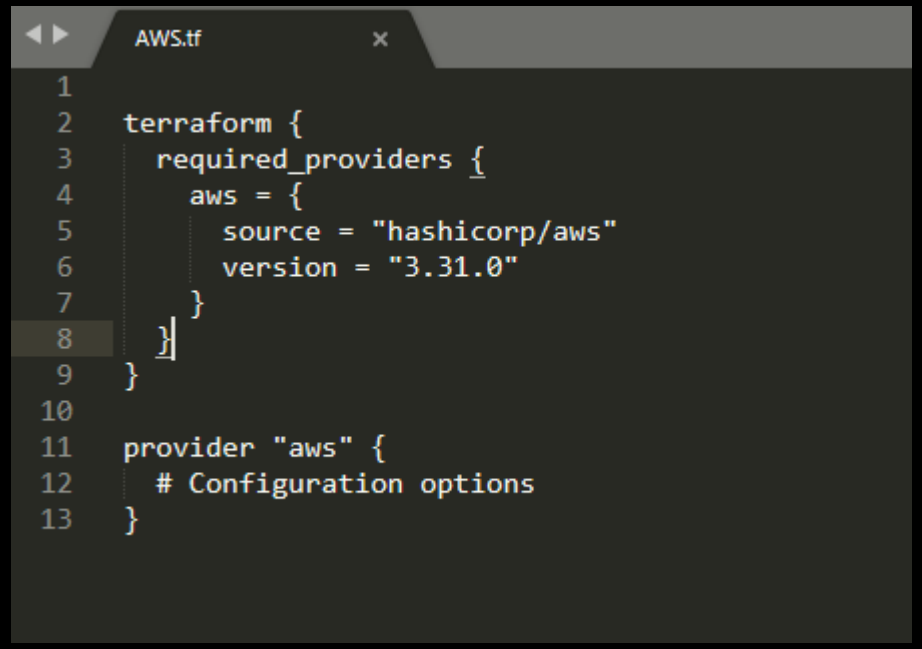
Terraform Execution Flow



- Executing a Terraform Script to provision in AWS:
  - 1 – Execute terraform against AWS.tf
  - 2 – Core makes Remote Procedure Call to Plugin to download AWS Provider and provide resources to provision
  - 3 – Plugin downloads AWS Libraries- ADD where the request goes
  - 4 – AWS Client library translates requests to API requests
  - 5 – API requests sent to AWS
  - 6 – Once provisioning is complete, state data is logged in terraform.tfstate

Terraform Providers

AWS.tf Terraform File:



- Terraform Providers:
  - To install a provider you must include provider configuration code into your terraform files
  - A list of available providers can be found here on the terraform website
  - Each Provider comes with its own set of documentation describing its resource types and their arguments
  - Terraform currently has 70 providers in their registry
  - You can create your own provider

Terraform Resources

Resource Template:

```
resource "<PROVIDER>_<TYPE>" "<NAME>" {
  [CONFIG …]
}
```

AWS.tf – Create an EC2 Instance:

```
resource "aws_instance" "example" {
    ami = "ami-0c09927662c939f41"
    instance_type = "t2.micro"
    tags { name = "TESTVM"}
}
```

- Resources:
  - Resources are the most important element in the Terraform language
  - They represent some type of infrastructure object
    - virtual networks, compute instances, compute devices or DNS records.
  - Terraform is platform dependent
  - Each provider has different requirements for parameters
  - AWS requires – ami ID, and Instance_type

Terraform Variables

AWS.tf Terraform File with Variables :

```
15
16  resource "aws_instance" "example" {
17      ami = var.amiID
18      instance_type = var.instanceType
19      tags { name = var.tags}
20  }
21
22
```

Variables.tf File :

```
1   variable "amiID"{
2       type = string
3       default = "ami-0c09927662c939f41"
4   }
5
6   variable "instanceType"{
7       type = string
8       default = "t2.micro"
9   }
10
11  variable "Tags"{
12      type = string
13      default = "TEST_VM"
14  }
```

- Variables:
  - Variables can be used to make your terraform scripts re-usable
  - Avoid having to hardcode in your parameters by using variables
  - Easier to Maintain – Instead of maintaining script file, you just need to maintain variables file
- Create Variables File:
  - Create a file in the same directory as your provisioning script and title it variables

Terraform Best Practices for File Structure

Example File Structure :

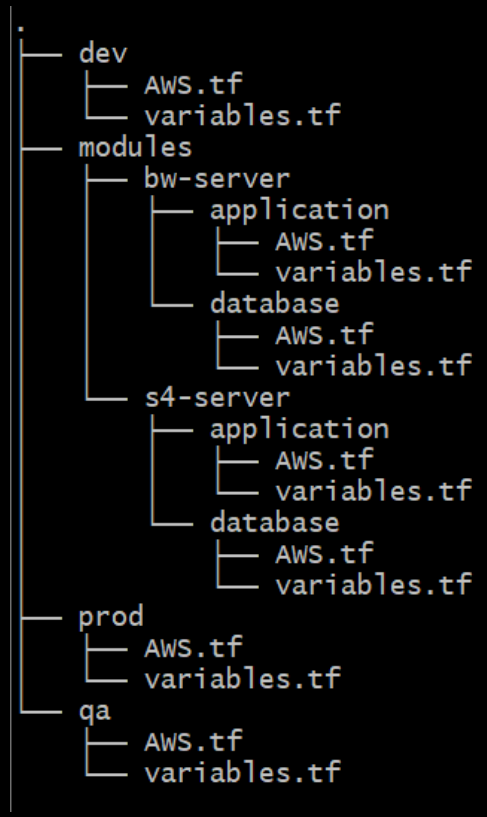
```
darpan-test:/terraform # tree -l
.
├── dev
│   ├── AWS.tf
│   └── variables.tf
├── prod
│   ├── AWS.tf
│   └── variables.tf
└── qa
    ├── AWS.tf
    └── variables.tf

3 directories, 6 files
darpan-test:/terraform #
```

- File Structure:
  - Separate out clusters of Servers
  - An example would be breaking out your environments to include DEV, QA, and PROD
  - Each environment has its own execution script – AWS.tf
  - Can also create groups such as Active Directory Servers, DNS servers, Printer Servers, etc..
- Drawbacks to this:
  - Every time you upgrade or modify one AWS.tf, you will need to modify the others

Terraform Modules

Example File Structure with Modules :



- Modules
  - Don’t need to update provisioning scripts in multiple locations
  - All resource provisioning scripts will be put into the module
  - Useful when managing multiple environments with different types of servers
  - Modules can be shared to the community here
- Use Case
  - You want to build out another application server for your S/4 Dev Environment
  - Update /dev/AWS.tf to include the module /modules/s4-server/application
  - To execute – cd into /dev and run: *terraform apply*

Getting Started with Ansible

Ansible Implementation Considerations

- Types of Resources in Ansible
  - Control Node
  - Managed Node
- Ansible is Agentless
- Installation Options
  - Install using OS package Manager
    - RHEL
    - SUSE
  - Install with Python Package Manager pip
- Control Node uses ssh to communicate with Managed Nodes
- Ansible Modules/Libraries are written in python

Control Node

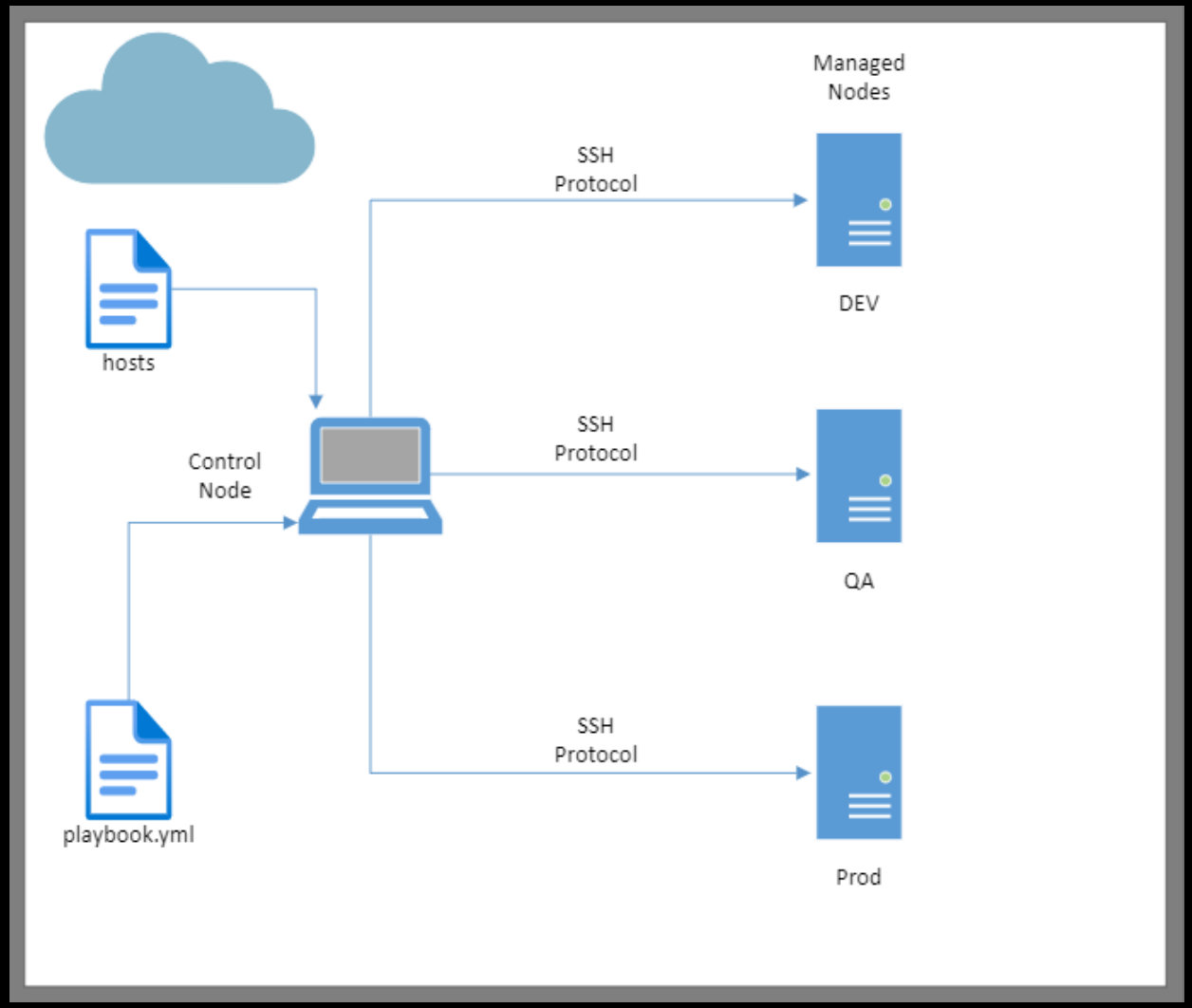
- Cannot be a Windows System
- Can be a laptop, shared desktop, or server
- RHEL, SUSE, macOS, Ubuntu, other linux operating systems
- Control Node should be located on the same network as Managed Nodes
- Must have python installed, python 3 is recommended

Managed Node

- Can be any operating system, including Windows
- Must have python installed, python 3 is recommended

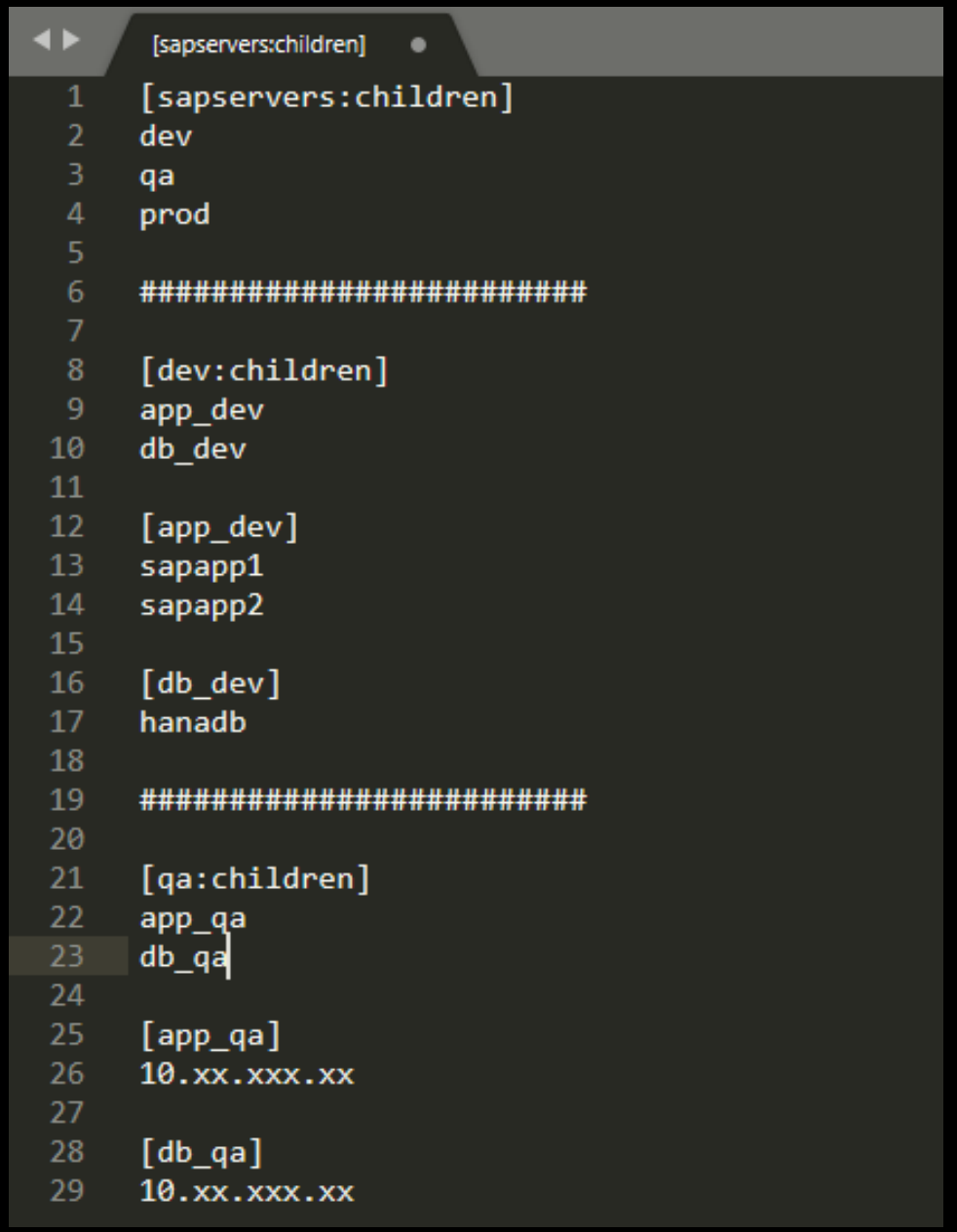
Ansible Execution Flow

- Authentication
  - SSH Keys or Passwords
- Hosts File
  - Contains the list of Managed Nodes
- Playbook
  - Contain Tasks to execute against the hosts in Hosts file



Ansible Hosts File

- Hosts File
  - Host file can be written in YAML or INI format
  - Managed nodes can use FQDN, Short Name, or IP Addresses
  - Group out your Servers for segregation of environments
  - Make groups of groups using :children suffix to make your execution calls more robust
  - Execute Ansible Playbooks against a server, a group of servers, and groups of groups of servers
  - More Information on Hosts files here



Ansible Playbook Files

- Playbook Files
  - Playbooks are written in YAML format
  - You must specify a host or a group of hosts to run the play against
  - Define your task by providing a name, module and parameters for each task
  - Playbooks define the execution flow of your Ansible tasks
- Execute Playbook command:
  - Ansible-playbook <playbook_name>
- Playbook Drawbacks
  - Can get messy when you have too many tasks

```yaml
- hosts: dev
  tasks:
  - name: Install SAP Hana Prerequisite Packages for SUSE Linux 15
    zypper:
      name: libgcc_s1
      state: present

  - name: Install SAP Hana Prerequisite Packages for SUSE Linux 15
    zypper:
      name: libstdc++6
      state: present

  - name: Install SAP Hana Prerequisite Packages for SUSE Linux 15
    zypper:
      name: libatomic1
      state: present

```

Ansible Roles

- Roles
  - Provide a clean way to organize your Ansible scripts and not overpopulating your playbooks
  - Treat roles like functions in programming
    - Should perform one task or a group of similar tasks
  - Roles are reusable and can be shared to your team or the Ansible Galaxy Community
- Role Creation
  - In the same directory your playbook can be found create a folder called roles and change directory into it
  - Run - ansible-galaxy init <Role_Name>
  - Folder structure for role is created

New look Playbook:

```
- hosts: dev
  roles:
    - { role: sap-configure }
```

Role task file:

```
---
# tasks file for sap-configure

- name: Install SAP Hana Prerequisite Packages for SUSE Linux 15
  zypper:
    name: libgcc_s1
    state: present

- name: Install SAP Hana Prerequisite Packages for SUSE Linux 15
  zypper:
    name: libstdc++6
    state: present

- name: Install SAP Hana Prerequisite Packages for SUSE Linux 15
  zypper:
    name: libatomic1
    state: present
```

Ansible Variable Files

- Variable File
  - Use variables to manage differences between systems
  - Prevents you from having to hard code in your playbook and role files
  - Create variables with YAML syntax, lists, or dictionaries
  - Define variables in playbooks, hosts file, roles, at the command line, or in separate vars files
  - Make all variables in vars file useable in your playbooks and roles
- Variable Drawbacks
  - Sensitive variables need to be secured as they are visible to users

Playbook File calling vars file:

```
- hosts: dev
  vars_files:
    - /etc/ansible/vars/ansible_vars.yml
  vars:
    ansible_ssh_private_key_file: "{{ private_key }}"
  roles:
    - { role: sap-configure }
```

Vars File:

```
#Secret Variables for Playbook
{
  "private_key": "/etc/ansible/dpid_rsa"
}
```

Ansible Vault

- Vault
  - Encrypt and Decrypt your sensitive files with Ansible Vault
  - Before files are encrypted you must provide a password
  - When executing playbooks, if an ansible vault encrypted file exists, you will be prompted for the vault password
  - Files are decrypted during runtime only for Ansible to use in execution
  - Create multiple vault passwords for encrypting different sets of files
- Vault Encrypt Command
  - ansible-vault encrypt <file_path1> <file_path2> ….

Vars file before encryption:

```
#Secret Variables for Playbook
{
  "private_key": "/etc/ansible/dpid_rsa"
}
```

Vars file after encryption:

```
$ANSIBLE_VAULT;1.1;AES256
31333532396633343931643130613139616666636363333339643465663864346336363656
39363131396365636633616437633463353236346138323 20a383361626261333535636132 3
32636435363930396135303236346661663163656536376337303435386137393061323862 3
35353631343662616 40a6234306639376362613433363764386232666337353636633831663
64363133393930636331633638636630616264323464636162326161343466623636313634 6
38623062333566363236636562653161353936633930333661343165666337383830326264 3
61643965333337326266326533383766633235636639343937316262656131333563343339 6
39326562643761663262663630616361383265613134303837666438333962323033323134 3
61343233613762323437323034656639326363623930316130653462623733656436653938 3
33353565653731353734303739306437666434323362396332653837353230613537336635 3
64323163306431633131353038643735656165613633333630653637363733313732386232 6
31643064323864356638326337393935326233316561363235313332643863346231646131 3
61313266626566646632333662623966343062386463376263323563636264393461303333
66353233343530643966333132383835656238396338383366316535343239313862323738 3
38383664616431613930623763663131326335333037313563326364663438636335531316
39306434653635656536646333643931353435633336643362653038353338663861396636 6
34643638383830663433396661636534623665306362653435393936653537643930
```

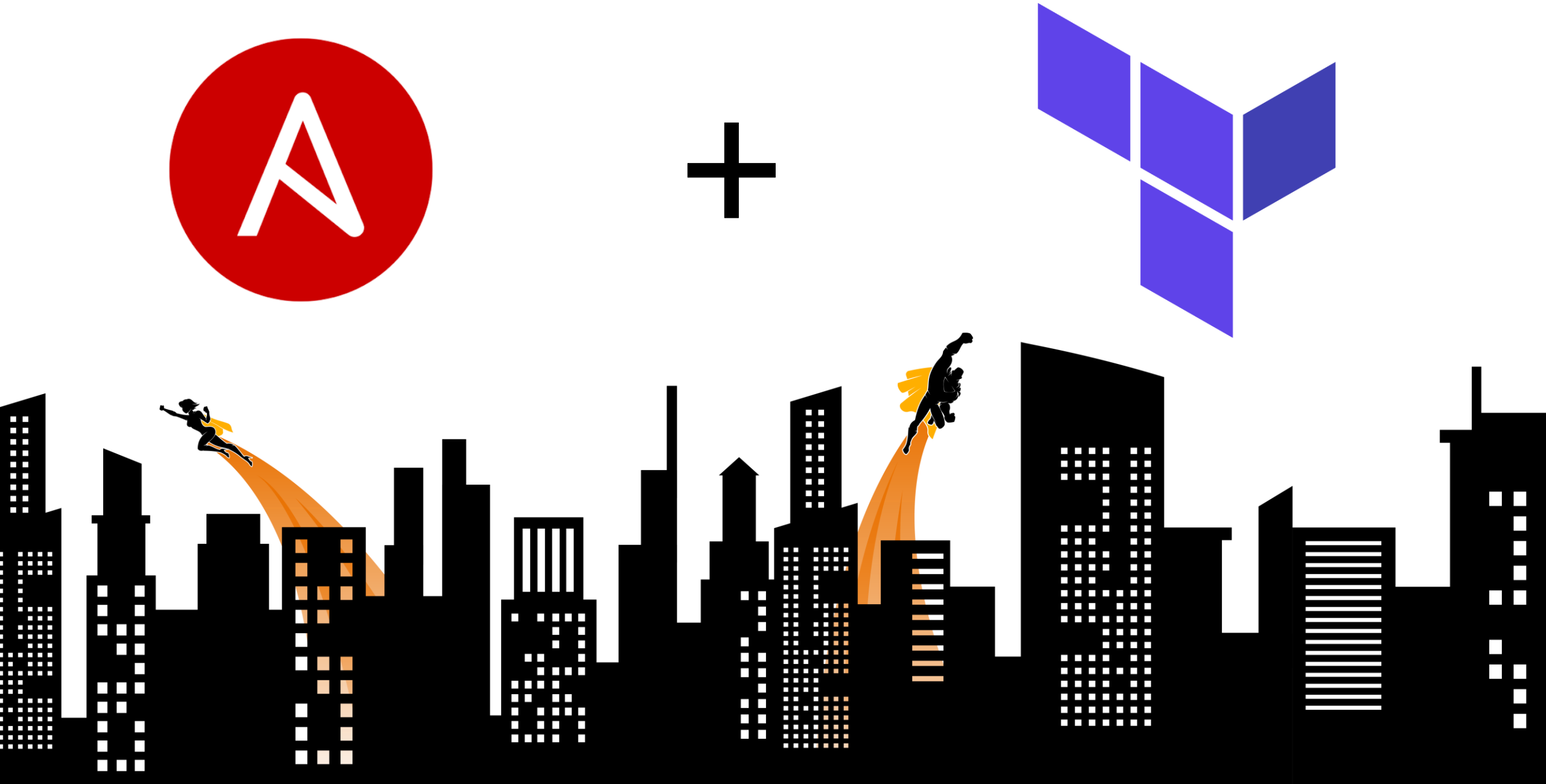
Ansible File Structure

- File Structure
  - Ansible can be run from anywhere
  - Bare minimum you need a playbook and hosts file
  - Group Vars gives you the ability to associate variables with a specific group of hosts
  - Host Vars gives you the ability to associate variables to specific hosts

Example of Ansible File Structure:

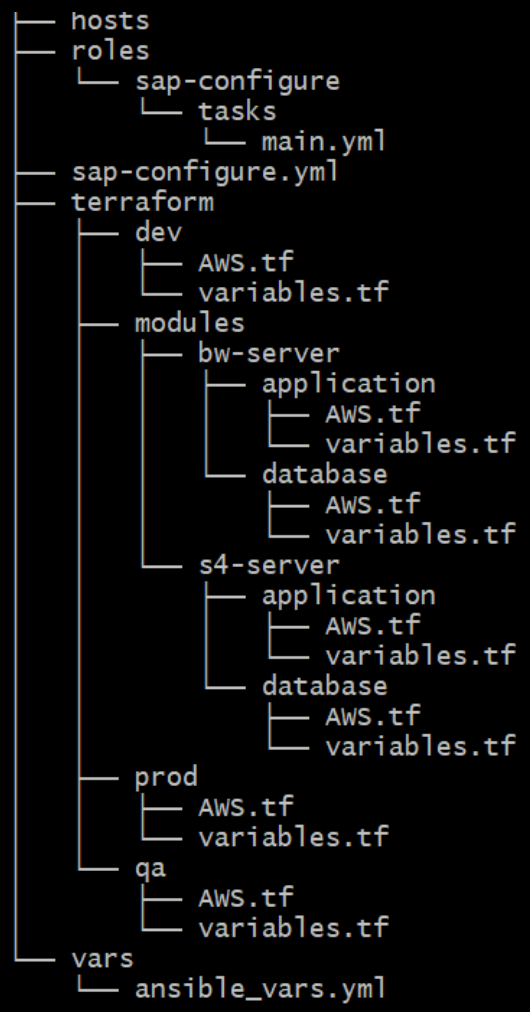
```
darpan-test:/ansible # tree -l
.
├── group_vars
│   ├── dev.yml
│   ├── prod.yml
│   ├── qa.yml
│   └── sapservers.yml
├── host_vars
│   ├── 10.47.106.205.yml
│   ├── hanadb.yml
│   └── sapapp1.yml
├── hosts
├── roles
│   └── sap-configure
│       └── tasks
│           └── main.yml
├── sap-configure.yml
└── vars
    └── ansible_vars.yml

6 directories, 11 files
darpan-test:/ansible # pwd
/ansible
```



Use Ansible to Call Terraform

Combining Terraform and Ansible File Structures:



sap-configure.yml calls Terraform:

```yaml
#############################################################
- hosts: localhost
  tasks:
    - name: init terraform
      shell: terraform init
      args:
        chdir: "/etc/ansible/terraform/dev"

    - name: apply terraform script
      terraform:
        project_path: "/etc/ansible/terraform/dev"
        state: present
#############################################################
- hosts: dev
  vars_files:
    - /ansible/vars/ansible_vars.yml
  vars:
    ansible_ssh_private_key_file: "{{ private_key }}"
  roles:
    - { role: sap-configure }
```

How Ansible Performs SAP Installations

Hdblcm In Batch Mode and SWPM Unattended:

- Config File needs to be maintained with parameters used for HANA and S/4 Installations
- You can write a playbook that picks up this file and inputs it into an hdblcm command or sapinst command
- No need to reinvent the wheel – download the following ansible roles
  - SAP HANA
  - SAP S/4HANA
- For SWPM, follow SAP Note - 2230669 to generate config files

Config File for SWPM:

```
# Standard system with AS ABAP only: ASCS instance number. Leave empty for default.
NW_CI_Instance.ascsInstanceNumber = {{ sap_s4hana_deployment_ascs_instance_nr }}

# Standard system with AS ABAP only: Virtual host name for the ASCS instance. Leave empty for default.
NW_CI_Instance.ascsVirtualHostname = {{ ansible_hostname }}

# Instance number of the primary application server instance. Leave empty for default.
NW_CI_Instance.ciInstanceNumber = {{ sap_s4hana_deployment_pas_instance_nr }}
```