Terraform and Ansible Deep Dive

Spectrum Scale German User Meeting 2022
Cologne, Germany – October 20th, 2022

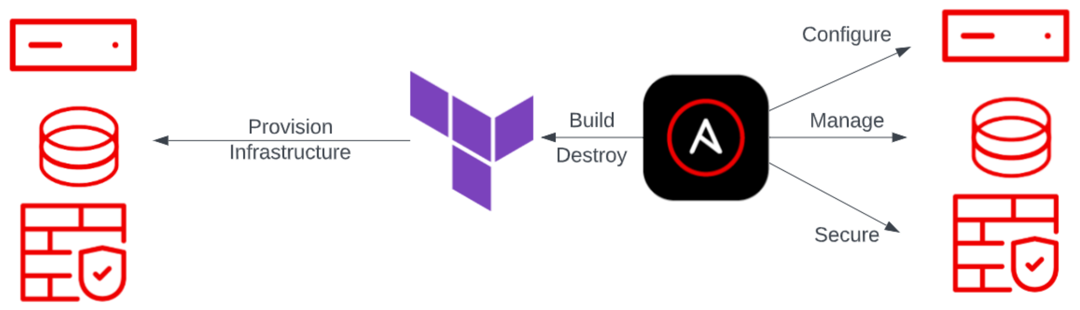
Achim Christ (IBM)
Muthu Muthiah (IBM)

Disclaimer

Agenda

- Benefits & Use Cases
- Getting started with Ansible
- Getting started with Spectrum Scale Ansible roles
- Getting started with Terraform and Spectrum Scale templates
- Sample scenarios in more detail

Benefits & Use Cases



[source]

- Manage storage as “just another service”
  → Without vendor lock-in

- Rapidly spin up test / demo / enablement clusters
- Dynamically scale-up / scale-down clusters
  → Reproducibility

- Keep (multiple) environments consistent
  → Prevent “configuration drift”

- Disaster Recovery / Cyber Resiliency
  → Revert to well-defined state (with confidence)

Key Performance Metrics

- End-to-end deployment in 10 minutes or less

- Best practices encoded into process
  - Availability Zone → Failure Group → Quorum
  - Protocol node tuning
  - …

- Advantages of Infrastructure as Code
  - ✓ Clearly defined state – ‘Single Source of Truth’
  - ✓ Combine with version control – ‘GitOps’
  - ✓ Ability to version / roll-back changes

  …

# nodes	Type	Duration
1	QEMU / KVM	~ 9 mins.
3	QEMU / KVM	~ 10 mins.
3 *(+ CES)*	QEMU / KVM	~ 13 mins.
20	AWS	~ 40 mins.
40	AWS	~ 60 mins.
64	AWS	~ 50 - 90 mins.

What is Red Hat Ansible?

- Automation framework (language)

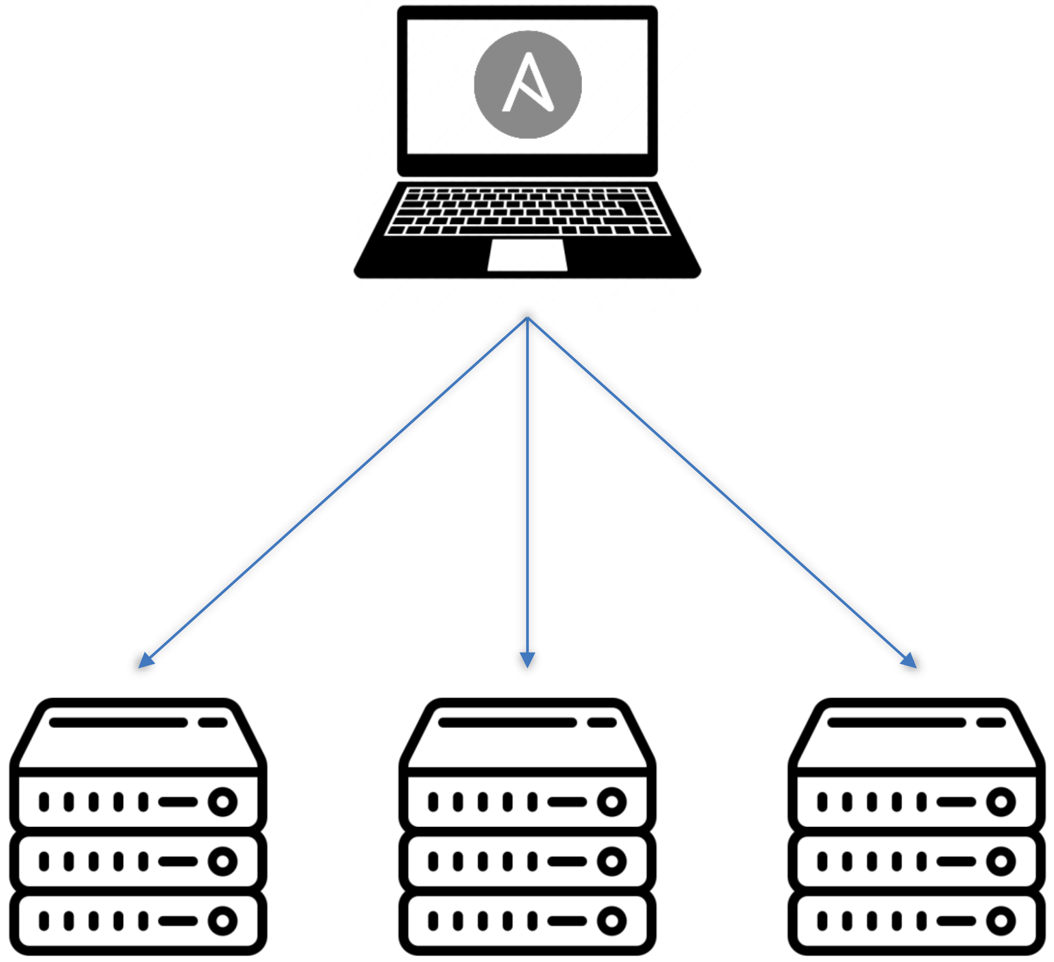
- Open source (GPL 3.0 licensed): GitHub Starred 54.9k
- Collection of CLI commands, optional GUI (AWX)
- Basis for Red Hat Ansible Automation Platform

- Initial release 2012
- Written in Python
- Ansible Inc. acquired by Red Hat in 2015
- Widely used across IBM portfolio [Galaxy]

Ansible Terminology

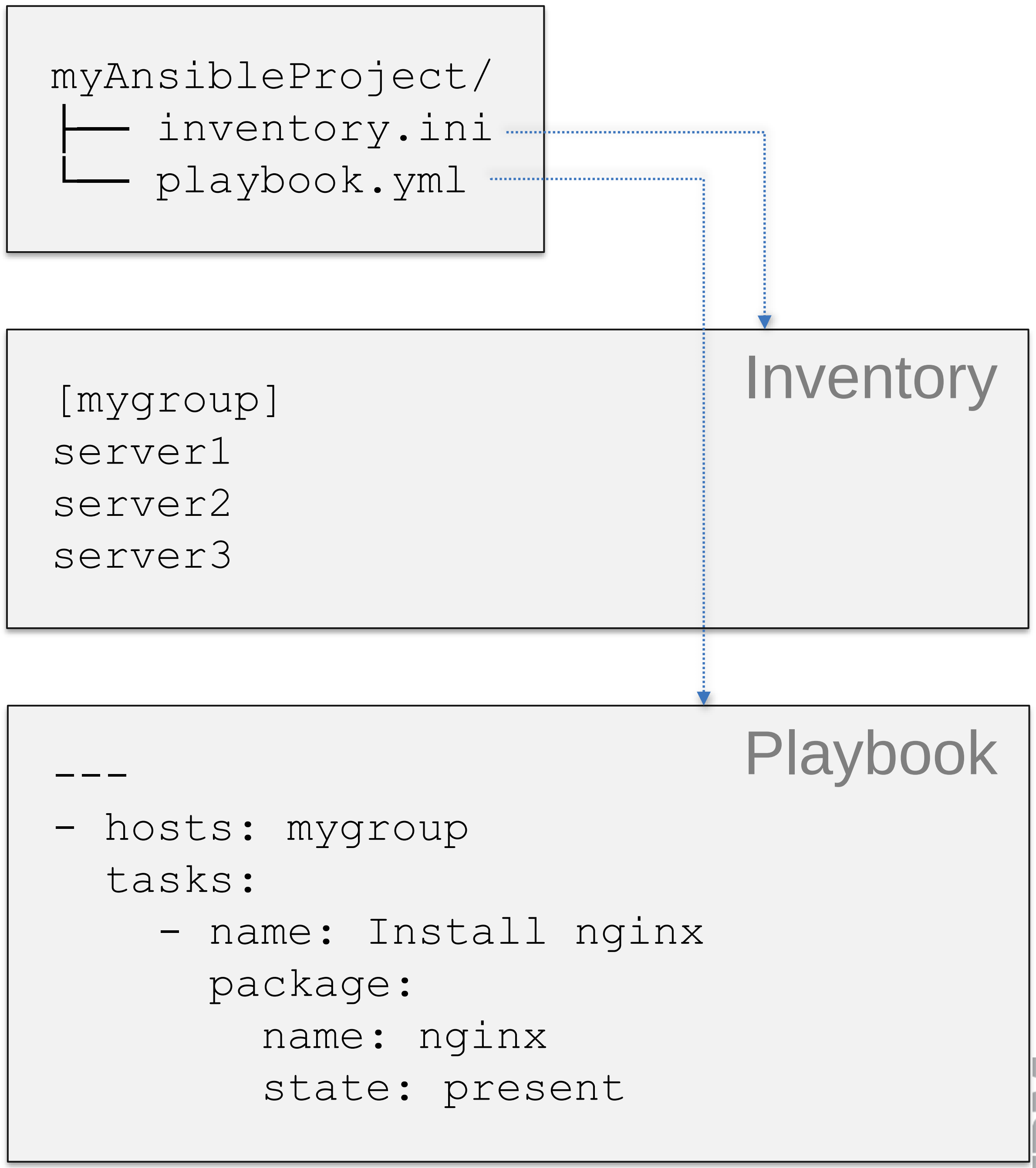
```
# ssh root@server1
```

- Control node
- Managed host (target node)
- Playbook
- Inventory



Ansible Terminology

- Control node
- Managed host (target node)
- Playbook
- Inventory



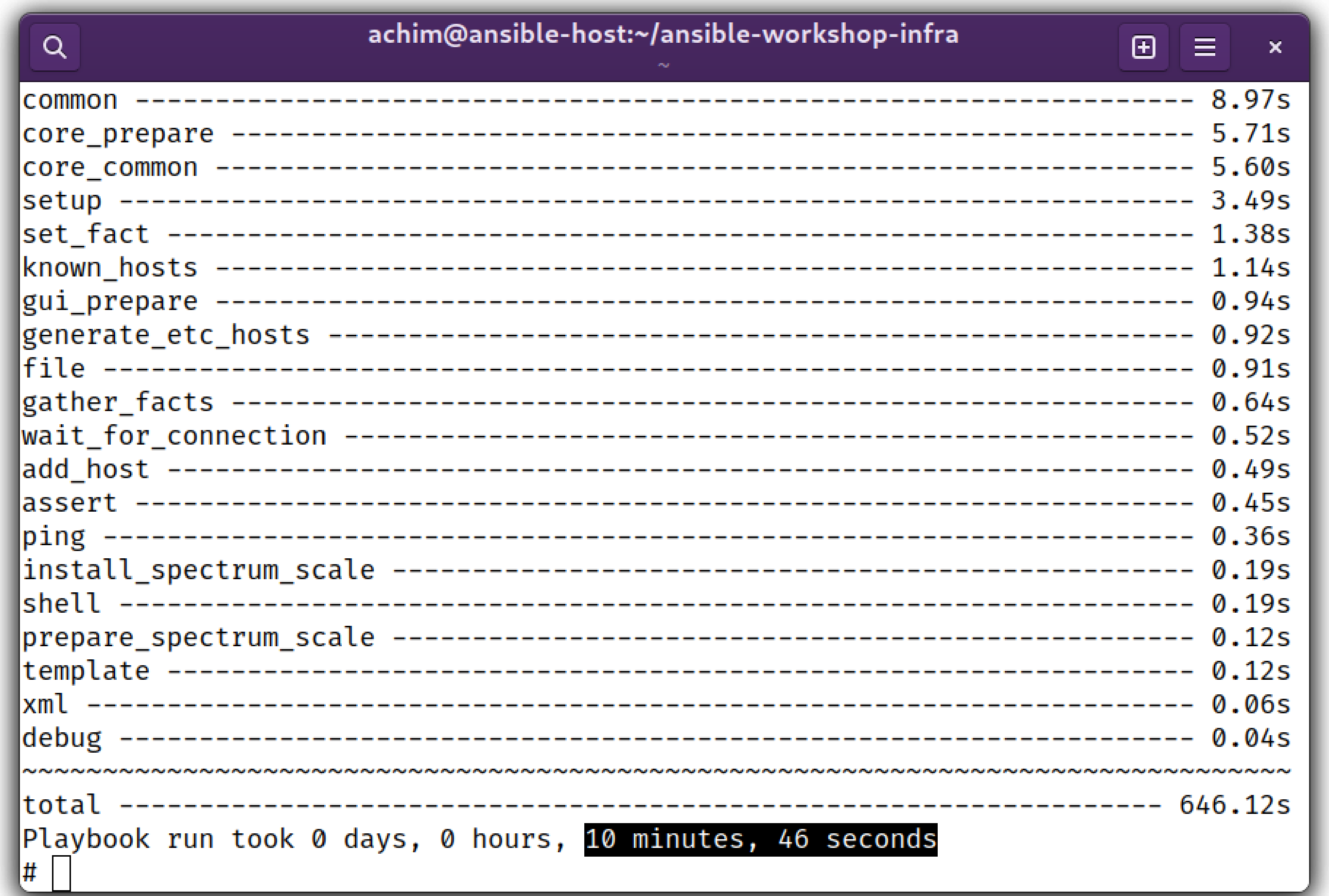
```
myAnsibleProject/
├── inventory.ini
└── playbook.yml
```

```
[mygroup]
server1
server2
server3
```

```
---
- hosts: mygroup
  tasks:
    - name: Install nginx
      package:
        name: nginx
        state: present
```

Getting Started with Ansible

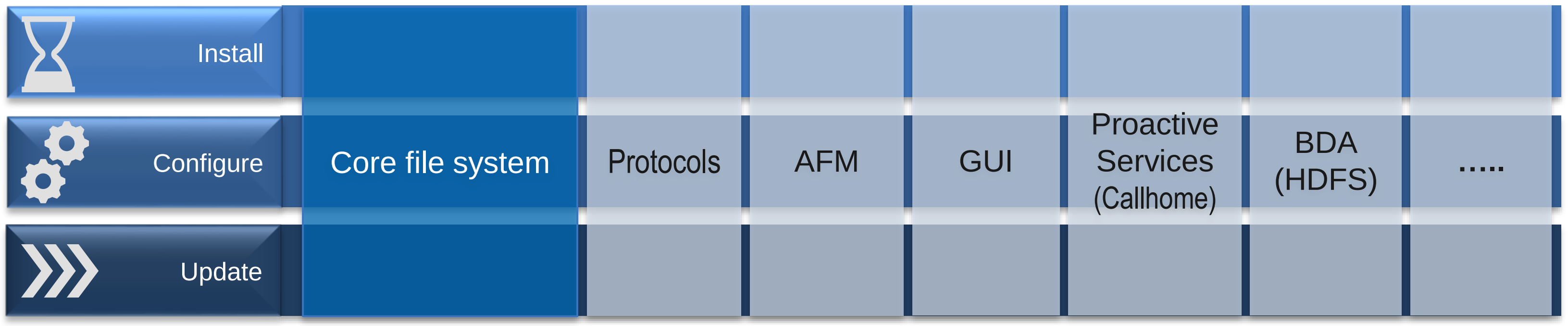
- `pip install ansible` [link]
- `ssh-copy-id root@server1`
- `vim inventory.ini`
- `vim playbook.yml`
- `ansible-playbook \` [link]
  ```
  -i inventory.ini \
  playbook.yml
  ```



```
common ------------------------------------------------------------------- 8.97s
core_prepare ------------------------------------------------------------- 5.71s
core_common -------------------------------------------------------------- 5.60s
setup -------------------------------------------------------------------- 3.49s
set_fact ----------------------------------------------------------------- 1.38s
known_hosts -------------------------------------------------------------- 1.14s
gui_prepare -------------------------------------------------------------- 0.94s
generate_etc_hosts ------------------------------------------------------- 0.92s
file --------------------------------------------------------------------- 0.91s
gather_facts ------------------------------------------------------------- 0.64s
wait_for_connection ------------------------------------------------------ 0.52s
add_host ----------------------------------------------------------------- 0.49s
assert ------------------------------------------------------------------- 0.45s
ping --------------------------------------------------------------------- 0.36s
install_spectrum_scale --------------------------------------------------- 0.19s
shell -------------------------------------------------------------------- 0.19s
prepare_spectrum_scale --------------------------------------------------- 0.12s
template ----------------------------------------------------------------- 0.12s
xml ---------------------------------------------------------------------- 0.06s
debug -------------------------------------------------------------------- 0.04s
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
total ------------------------------------------------------------------ 646.12s
Playbook run took 0 days, 0 hours, 10 minutes, 46 seconds
# 
```

→ https://github.com/acch/ansible-boilerplate

Spectrum Scale ↔ Ansible Structure



→ https://github.com/ibm/ibm-spectrum-scale-install-infra/tree/main

```
roles/
├── callhome_*
├── core_*
├── ece_*
├── fal_*
├── gui_*
├── hdfs_*
├── nfs_*
├── obj_*
├── perfmon_*
└── smb_*
```

〉

```
roles/
├── core_prepare
├── core_install
├── core_configure
├── core_verify
└── core_upgrade
```

Getting started with Spectrum Scale Ansible roles

- `git clone \` [link]
  ```
  -b main \
  https://github.com/ibm/ibm-spectrum-scale-install-infra.git \
  collections/ansible_collections/ibm/spectrum_scale
  ```
- `vim playbook.yml`

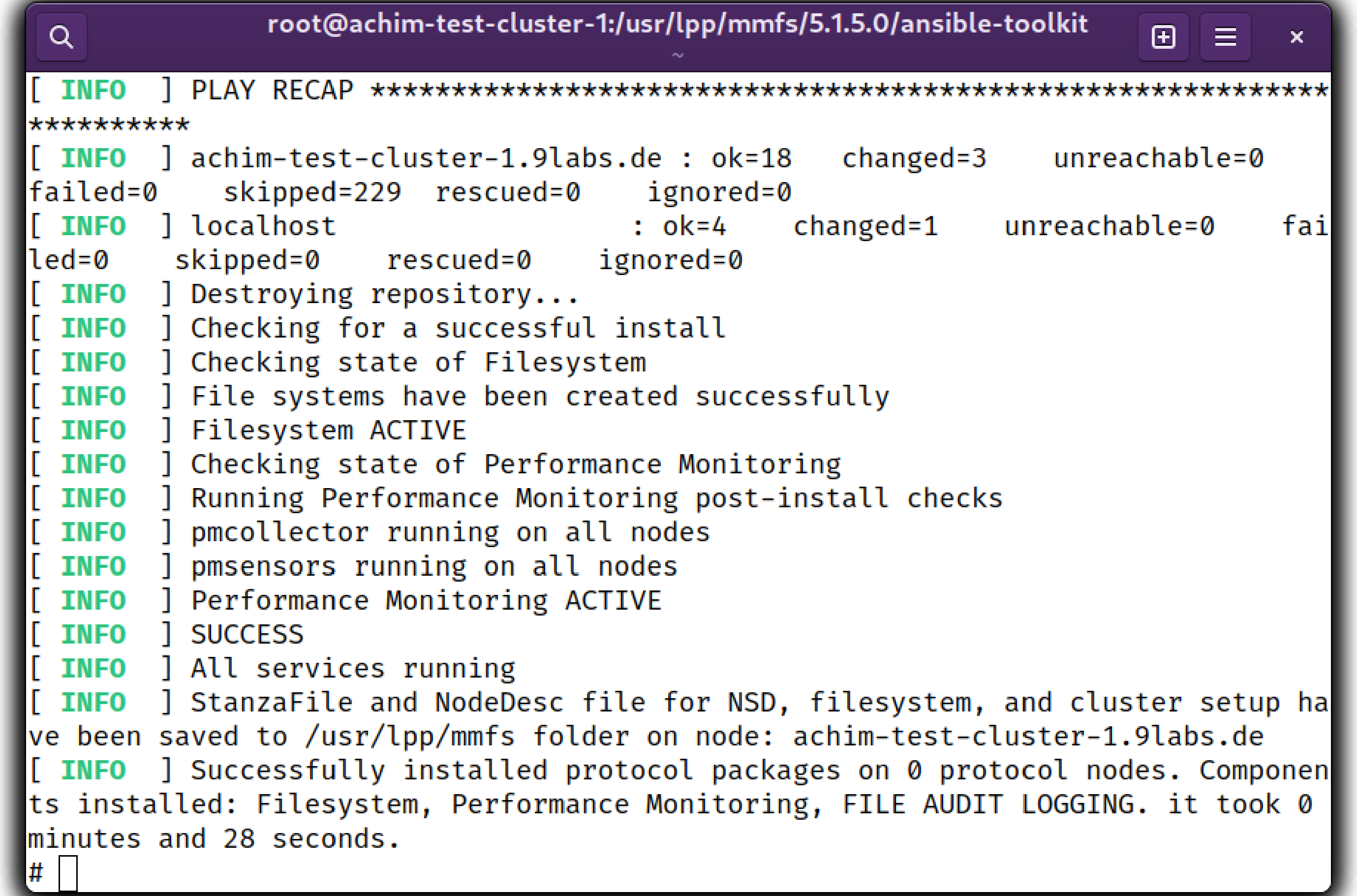
- Future:
  `ansible-galaxy collection install ibm.spectrum_scale`

```yaml
---
- hosts: all

  roles:
    - ibm.spectrum_scale.core_prepare
    - ibm.spectrum_scale.core_install
    - ibm.spectrum_scale.core_configure
    - ibm.spectrum_scale.core_verify

  vars:
    scale_install_repository_url: >-
      http://webserver.local/gpfs_rpms/
    scale_cluster_clustername: >-
      mycluster.local
    scale_storage:
      filesystem: gpfs01
      disks:
        - device: /dev/vdb
          servers: localhost
```

→ https://github.com/ibm/ibm-spectrum-scale-install-infra/tree/main#readme

Batteries Included: Installation Toolkit

```
# /usr/lpp/mmfs/*/ansible-toolkit/spectrumscale -h
usage: spectrumscale [-h] [-v] [--version]
{setup,node,config,nsd,filesystem,callhome,
fileauditlogging,enable,disable,install,deploy,
upgrade}

positional arguments:
    setup                …
    node                 …
    config               …
    nsd                  …
    filesystem           …
    callhome             …
    fileauditlogging     …
    enable               …
    disable              …
    install              …
    deploy               …
    upgrade              …

optional arguments:
  -h, --help             …
  -v, --verbose          …
  --version              …

For usage of sub-commands, specify a command
followed by -h or --help
```



root@achim-test-cluster-1:/usr/lpp/mmfs/5.1.5.0/ansible-toolkit

```
[ INFO  ] PLAY RECAP ****************************************************************
**********
[ INFO  ] achim-test-cluster-1.9labs.de : ok=18   changed=3    unreachable=0
failed=0    skipped=229  rescued=0    ignored=0
[ INFO  ] localhost                    : ok=4    changed=1    unreachable=0    fai
led=0    skipped=0    rescued=0    ignored=0
[ INFO  ] Destroying repository...
[ INFO  ] Checking for a successful install
[ INFO  ] Checking state of Filesystem
[ INFO  ] File systems have been created successfully
[ INFO  ] Filesystem ACTIVE
[ INFO  ] Checking state of Performance Monitoring
[ INFO  ] Running Performance Monitoring post-install checks
[ INFO  ] pmcollector running on all nodes
[ INFO  ] pmsensors running on all nodes
[ INFO  ] Performance Monitoring ACTIVE
[ INFO  ] SUCCESS
[ INFO  ] All services running
[ INFO  ] StanzaFile and NodeDesc file for NSD, filesystem, and cluster setup ha
ve been saved to /usr/lpp/mmfs folder on node: achim-test-cluster-1.9labs.de
[ INFO  ] Successfully installed protocol packages on 0 protocol nodes. Componen
ts installed: Filesystem, Performance Monitoring, FILE AUDIT LOGGING. it took 0
minutes and 28 seconds.
#
```

What is HashiCorp Terraform?

- Infrastructure resource provisioning

- Open source (MPL 2.0 licensed): GitHub

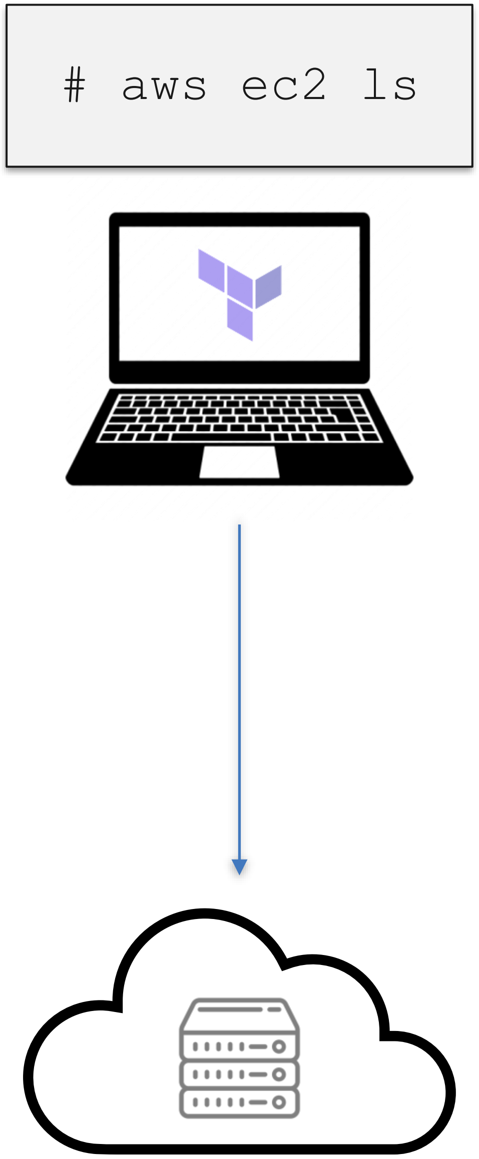
Starred 34.4k

- Terraform CLI
- Terraform Cloud
- Terraform Enterprise

- Initial release 2014
- Written in Go
- Extensive list of (cloud) infrastructure providers

Terraform Terminology

- Provider
- Resource
- Module
- Data Source
- Template
- State



```
# aws ec2 ls
```

Getting started with Terraform

- Install Terraform
  - `yum install -y yum-utils`
  - `yum-config-manager --add-repo` https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo
  - `yum -y install terraform`
- Install and configure AWS CLI
  - `curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip‟ && unzip awscliv2.zip sudo ./aws/install` [link]
  - `aws configure` [link]
- Define Terraform Configuration (template)
  - `vim template.tf`
- Execute Terraform
  - `terraform init` [link]
  - `terraform plan` [link]
  - `terraform apply` [link]
  - `terraform destroy` [link]

```
#   yum -y install terraform
Updating Subscription Management repositories.
Red Hat Enterprise Linux 8 for x86  37 kB/s | 2.4 kB     00:00
Red Hat Enterprise Linux 8 for x86  44 kB/s | 2.8 kB     00:00
Dependencies resolved.
================================================================
 Package          Architecture Version         Repository       Size
================================================================
Installing:
 terraform        x86_64       1.3.2-1         hashicorp        13 M

Transaction Summary
================================================================
Install  1 Package

Total download size: 13 M
Installed size: 58 M
```

```
# aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrX…EXAMPLEKEY
Default region name [None]: eu-central-1
Default output format [None]:
```

Getting started with Terraform

```
# cat provider.tf
```

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }

  required_version = ">= 1.0"
}

provider "aws" {
  region  = "ap-south-1"
}
```

```
# cat main.tf
```

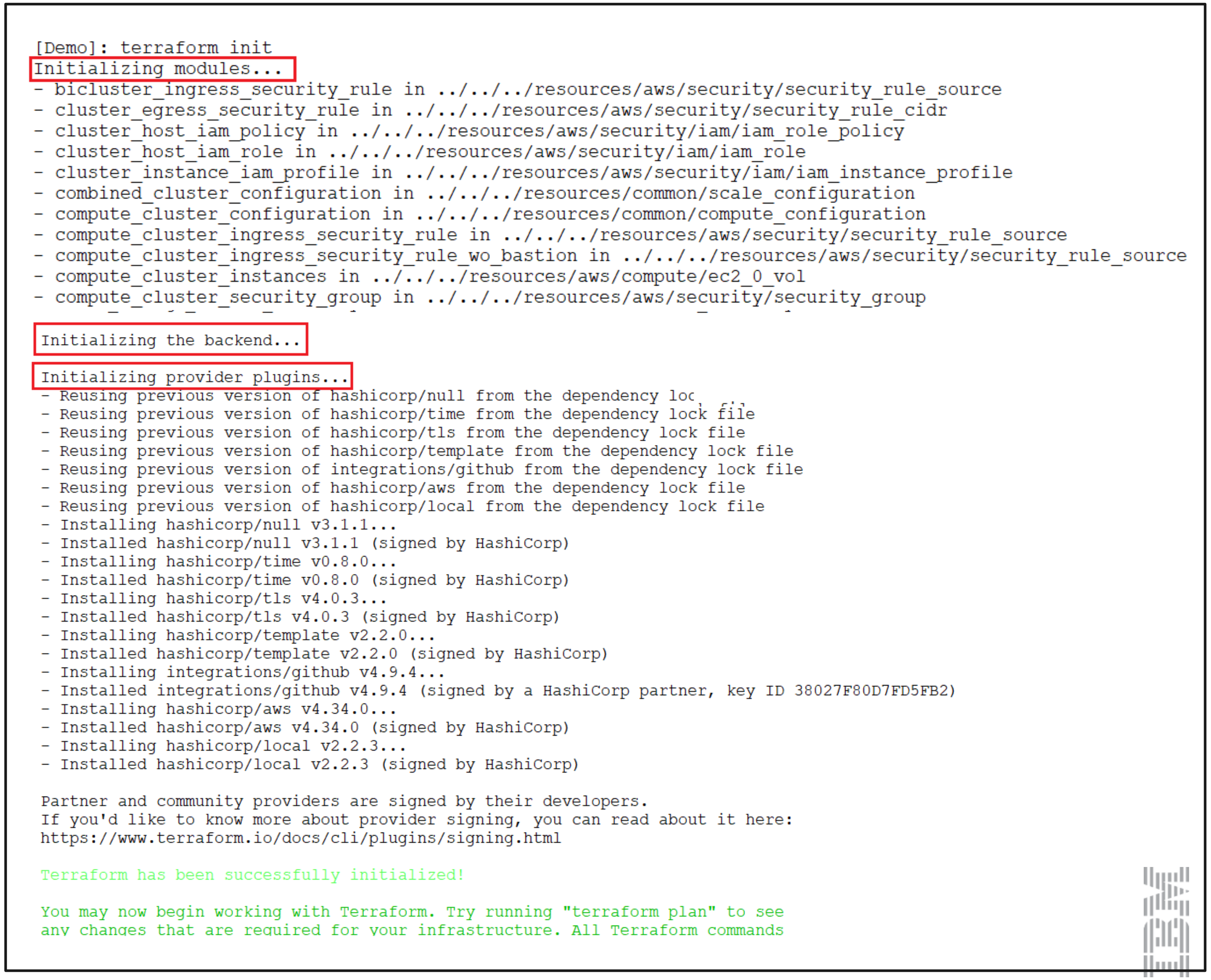
```
resource "aws_instance" "demo-instance" {

   ami                         = "ami-01216e7612243e0ef"
   instance_type               = "t2.micro"
   associate_public_ip_address = "true"
   vpc_security_group_ids      = ["sg-008e9776a30111111"]
   subnet_id                   = "subnet-0f724b2fda111111"

   tags = {
     Name = "demo-instance"
   }
}
```

Getting started with Terraform

```
# terraform init
```

The *terraform init* command performs the following:

- Initializes the directory containing Terraform configuration files (templates)
- **Initialize Modules**: All module blocks defined in the Terraform configuration files (templates) are retrieved and initialized
- **Initialize Backend**: Initializes where Terraform State files are stored (local file system/S3 Bucket etc.)
- **Initialize Provider Plugins**: All Terraform Plugins whose providers are referenced in the Terraform configuration files (templates) either directly or indirectly are retrieved and installed



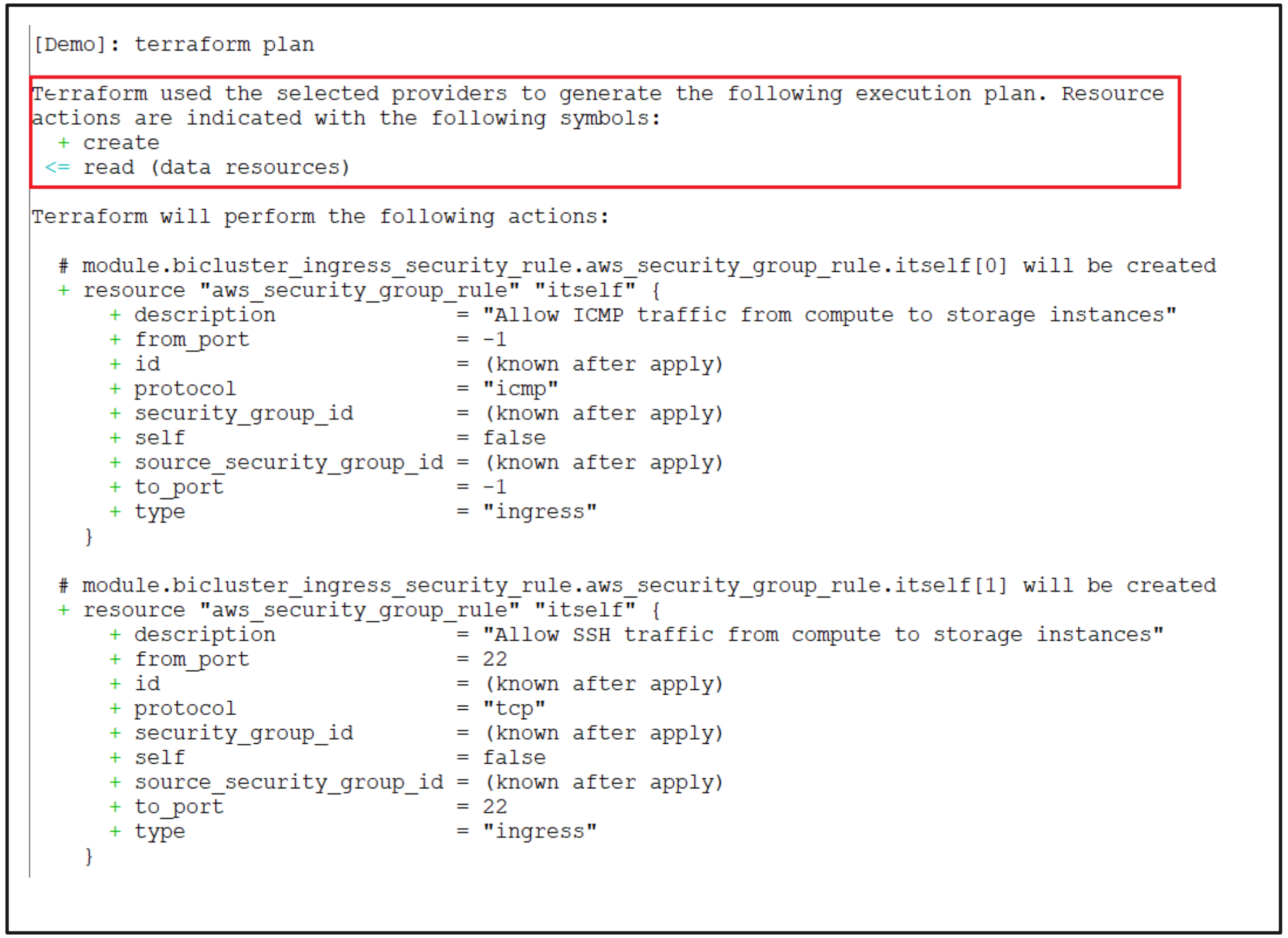
```
[Demo]: terraform init
Initializing modules...
- bicluster_ingress_security_rule in ../../../resources/aws/security/security_rule_source
- cluster_egress_security_rule in ../../../resources/aws/security/security_rule_cidr
- cluster_host_iam_policy in ../../../resources/aws/security/iam/iam_role_policy
- cluster_host_iam_role in ../../../resources/aws/security/iam/iam_role
- cluster_instance_iam_profile in ../../../resources/aws/security/iam/iam_instance_profile
- combined_cluster_configuration in ../../../resources/common/scale_configuration
- compute_cluster_configuration in ../../../resources/common/compute_configuration
- compute_cluster_ingress_security_rule in ../../../resources/aws/security/security_rule_source
- compute_cluster_ingress_security_rule_wo_bastion in ../../../resources/aws/security/security_rule_source
- compute_cluster_instances in ../../../resources/aws/compute/ec2_0_vol
- compute_cluster_security_group in ../../../resources/aws/security/security_group

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/null from the dependency loc
- Reusing previous version of hashicorp/time from the dependency lock file
- Reusing previous version of hashicorp/tls from the dependency lock file
- Reusing previous version of hashicorp/template from the dependency lock file
- Reusing previous version of integrations/github from the dependency lock file
- Reusing previous version of hashicorp/aws from the dependency lock file
- Reusing previous version of hashicorp/local from the dependency lock file
- Installing hashicorp/null v3.1.1...
- Installed hashicorp/null v3.1.1 (signed by HashiCorp)
- Installing hashicorp/time v0.8.0...
- Installed hashicorp/time v0.8.0 (signed by HashiCorp)
- Installing hashicorp/tls v4.0.3...
- Installed hashicorp/tls v4.0.3 (signed by HashiCorp)
- Installing hashicorp/template v2.2.0...
- Installed hashicorp/template v2.2.0 (signed by HashiCorp)
- Installing integrations/github v4.9.4...
- Installed integrations/github v4.9.4 (signed by a HashiCorp partner, key ID 38027F80D7FD5FB2)
- Installing hashicorp/aws v4.34.0...
- Installed hashicorp/aws v4.34.0 (signed by HashiCorp)
- Installing hashicorp/local v2.2.3...
- Installed hashicorp/local v2.2.3 (signed by HashiCorp)

Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
```

Getting started with Terraform

```
# terraform plan
```

The *terraform plan* command:

- Allows a preview the actions Terraform will take to modify the infrastructure
- The output uses the following format
  + For resources that will be newly created
  - For resources that will be deleted
  ~ For resources that will be modified
- It also provides the ability to save the plan for later input into the *terraform apply* command



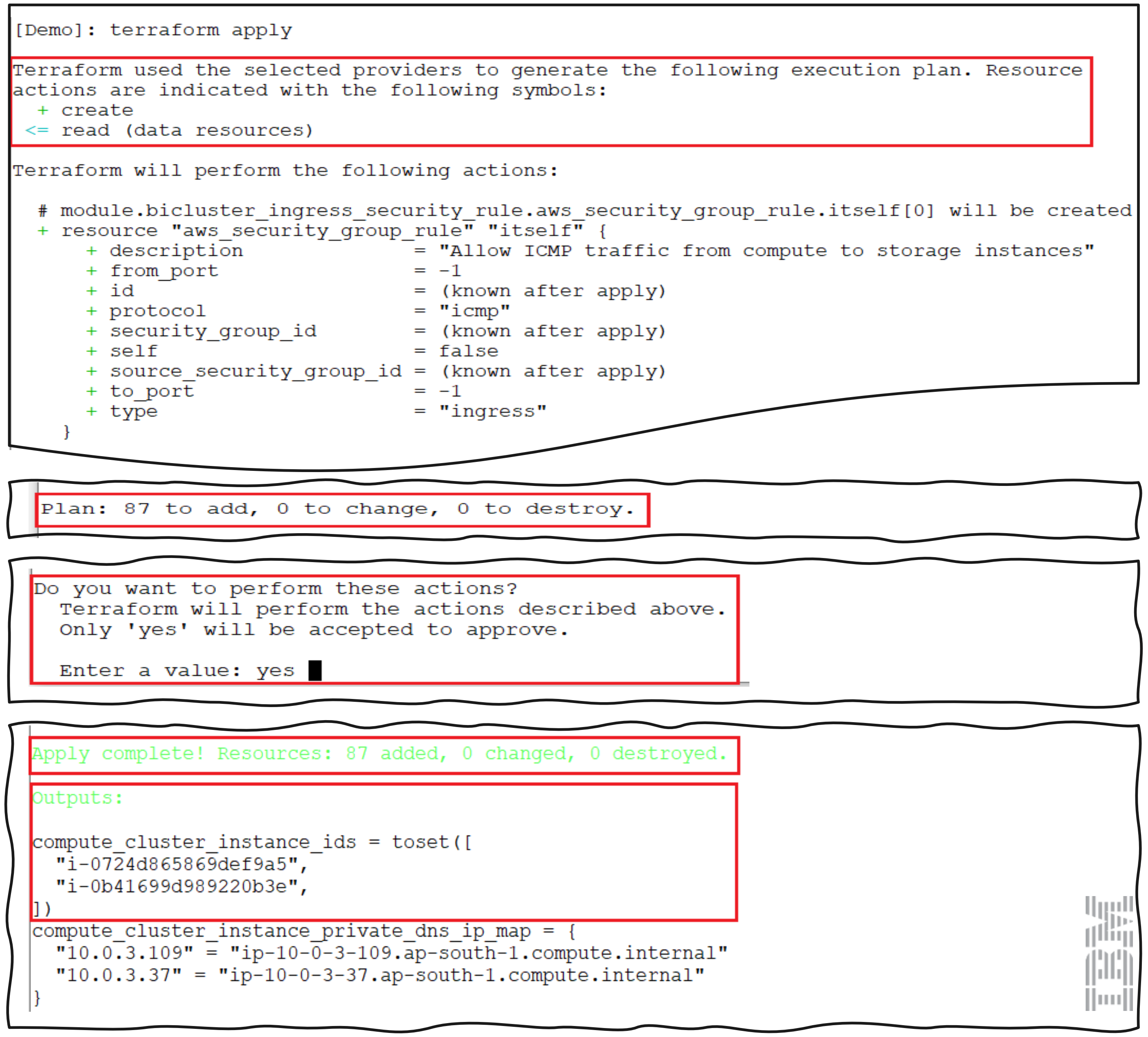
```
[Demo]: terraform plan

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the following symbols:
  + create
 <= read (data resources)

Terraform will perform the following actions:

  # module.bicluster_ingress_security_rule.aws_security_group_rule.itself[0] will be created
  + resource "aws_security_group_rule" "itself" {
      + description              = "Allow ICMP traffic from compute to storage instances"
      + from_port                = -1
      + id                       = (known after apply)
      + protocol                 = "icmp"
      + security_group_id        = (known after apply)
      + self                     = false
      + source_security_group_id = (known after apply)
      + to_port                  = -1
      + type                     = "ingress"
    }

  # module.bicluster_ingress_security_rule.aws_security_group_rule.itself[1] will be created
  + resource "aws_security_group_rule" "itself" {
      + description              = "Allow SSH traffic from compute to storage instances"
      + from_port                = 22
      + id                       = (known after apply)
      + protocol                 = "tcp"
      + security_group_id        = (known after apply)
      + self                     = false
      + source_security_group_id = (known after apply)
      + to_port                  = 22
      + type                     = "ingress"
    }
```

Getting started with Terraform

```
# terraform apply
```

The *terraform apply* command:

- Executes the changes defined by the Terraform configuration (template) to create, modify and destroy resources
- Lock the Terraform project state, so that no other instances of Terraform will be able to manipulate the resources at the same time
- Create a plan and prompt for approval
- Execute steps in the plan using the providers that were previously installed during *terraform init*
- Update project state file with the current state of all provisioned resources
- Unlock the state file
- Print out a report of all changes that were made
- Print out any output values defined in configuration



```
[Demo]: terraform apply

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the following symbols:
  + create
 <= read (data resources)

Terraform will perform the following actions:

  # module.bicluster_ingress_security_rule.aws_security_group_rule.itself[0] will be created
  + resource "aws_security_group_rule" "itself" {
      + description              = "Allow ICMP traffic from compute to storage instances"
      + from_port                = -1
      + id                       = (known after apply)
      + protocol                 = "icmp"
      + security_group_id        = (known after apply)
      + self                     = false
      + source_security_group_id = (known after apply)
      + to_port                  = -1
      + type                     = "ingress"
    }
```

```
Plan: 87 to add, 0 to change, 0 to destroy.
```

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

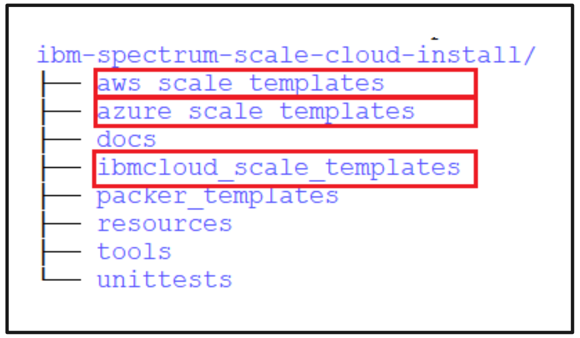
```
Apply complete! Resources: 87 added, 0 changed, 0 destroyed.

Outputs:

compute_cluster_instance_ids = toset([
  "i-0724d865869def9a5",
  "i-0b41699d989220b3e",
])
compute_cluster_instance_private_dns_ip_map = {
  "10.0.3.109" = "ip-10-0-3-109.ap-south-1.compute.internal"
  "10.0.3.37" = "ip-10-0-3-37.ap-south-1.compute.internal"
}
```

Spectrum Scale ↔ Terraform Structure

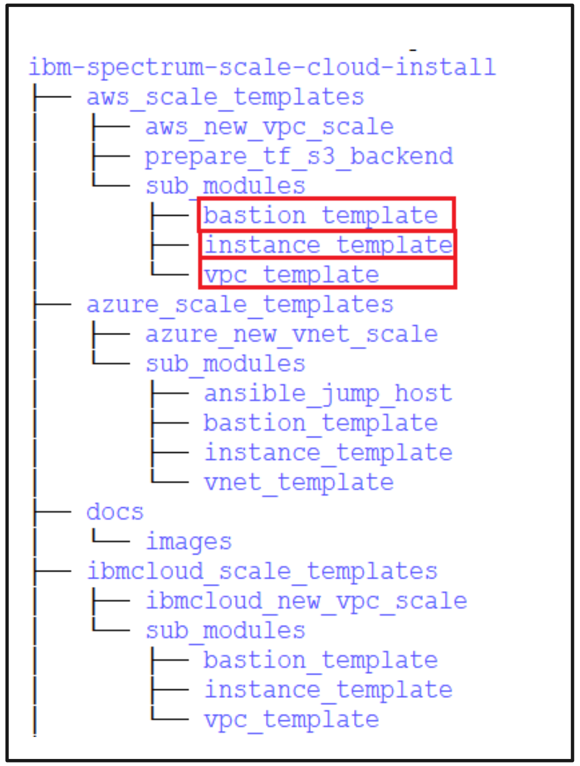
→ https://github.com/ibm/ibm-spectrum-scale-cloud-install

The Spectrum Scale Terraform Git Repository consists of:

- Templates to provision the required infrastructure on various public clouds
  - AWS
  - IBM Cloud
  - Azure (PoC Only)
  - GCP (PoC only available in master branch)
- Component Templates to provision (as configured)
  - Virtual Private Cloud (VPC)
  - Bastion Host
  - Storage and Compute Instances
- Templates to provision all other required Cloud resources
  - Networking configuration (VPC Private and Public Subnets as needed)
  - Storage (EBS Volumes for use as NSD)
  - Security configuration (Security Groups, Identity and Access Management configuration)



```
ibm-spectrum-scale-cloud-install/
├── aws_scale_templates
├── azure_scale_templates
├── docs
├── ibmcloud_scale_templates
├── packer_templates
├── resources
├── tools
└── unittests
```



```
ibm-spectrum-scale-cloud-install
├── aws_scale_templates
│   ├── aws_new_vpc_scale
│   ├── prepare_tf_s3_backend
│   └── sub_modules
│       ├── bastion_template
│       ├── instance_template
│       └── vpc_template
├── azure_scale_templates
│   ├── azure_new_vnet_scale
│   └── sub_modules
│       ├── ansible_jump_host
│       ├── bastion_template
│       ├── instance_template
│       └── vnet_template
├── docs
│   └── images
├── ibmcloud_scale_templates
│   ├── ibmcloud_new_vpc_scale
│   └── sub_modules
│       ├── bastion_template
│       ├── instance_template
│       └── vpc_template
```

Spectrum Scale Terraform ↔ Ansible Integration

- Terraform provisions all required resources on the cloud and stores the information into a state file
- Option to generate Ansible inventory from Terraform state
- Option to execute Ansible from Terraform
- Alternative: start Terraform from Ansible
  - `community.general.terraform` module [link]
  - Dynamic inventory plugin [link]

```
- name: Deploy Terraform Instance
  community.general.terraform:
    project_path: /path/to/tf_build
    state: present
  register: deployed_tf
```

```
[Demo]: generate_spectrum_scale_ansible_inventory.sh
Identified cluster type: combined
Total node count:  4
Total quorum count:  3
Writing cloud infrastructure details to:  scale_clusterdefinition.json
Completed writing cloud infrastructure details to:  scale_clusterdefinition.json
```

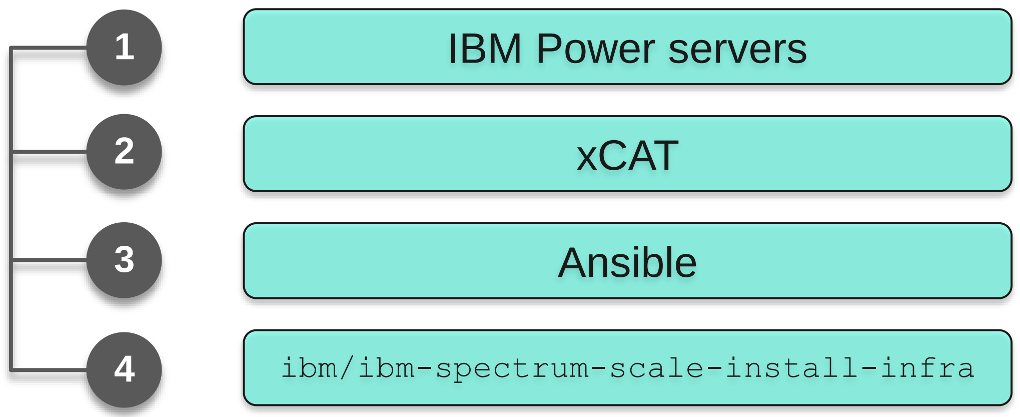
Sample Scenario I (on-prem)

- University & research customer @ Switzerland

- Goal
  - Efficiently deploy & manage large protocol clusters consistently
  - Refresh IBM Power server hardware

- Challenges
  - 12 protocol nodes (SMB & NFS) with AD integration
  - Complex network / firewall / hardening rules
  - Historically grown:
    slightly different configuration on each node
  - Migrate from Big Endian → Little Endian

- Solution
  - IBM Power 9 servers
  - Initialization & OS installation through xCAT
  - OS configuration (security, compliance) through Ansible
  - Spectrum Scale installation through Ansible
  - Spectrum Scale configuration through
    - `mmaddnode`
    - `tdbdump` / `tdbrestore`

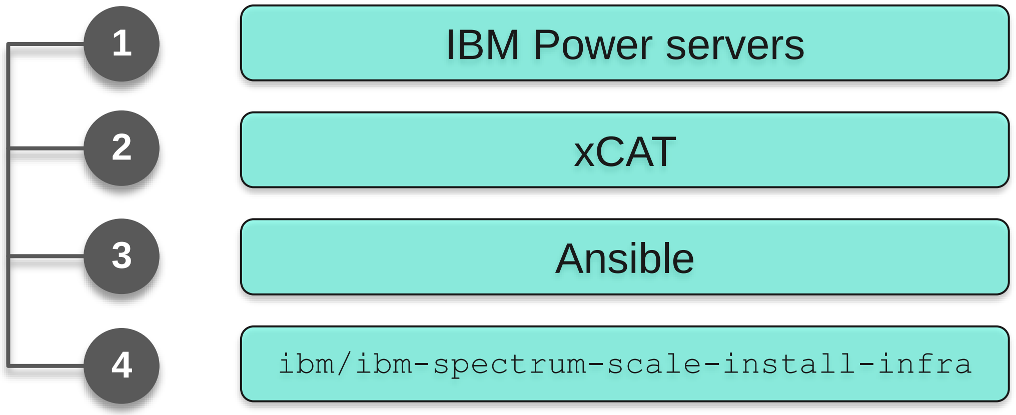


Sample Scenario I (on-prem)

- Use xCAT for provisioning RHEL on IBM Power
  - Use (node) groups
  - Kickstart template / custom partitioning (script)
  - Custom (post-)script to inject SSH key

- All post OS-installation configuration: Ansible playbook(s)
  - Custom hardening, compliance
  - Admin user configuration
  - Firewall configuration
  - `rhel-system-roles`

- Spectrum Scale package installation using Scale roles
- Subsequent node configuration:
  - `` `mmaddnode` `` | `` `mmsdrrestore` ``



```
- ibm.spectrum_scale.core_prepare
- ibm.spectrum_scale.core_install
- ibm.spectrum_scale.core_verify
- ibm.spectrum_scale.nfs_prepare
- ibm.spectrum_scale.nfs_install
- ibm.spectrum_scale.nfs_verify
- ibm.spectrum_scale.smb_prepare
- ibm.spectrum_scale.smb_install
- ibm.spectrum_scale.smb_verify
```

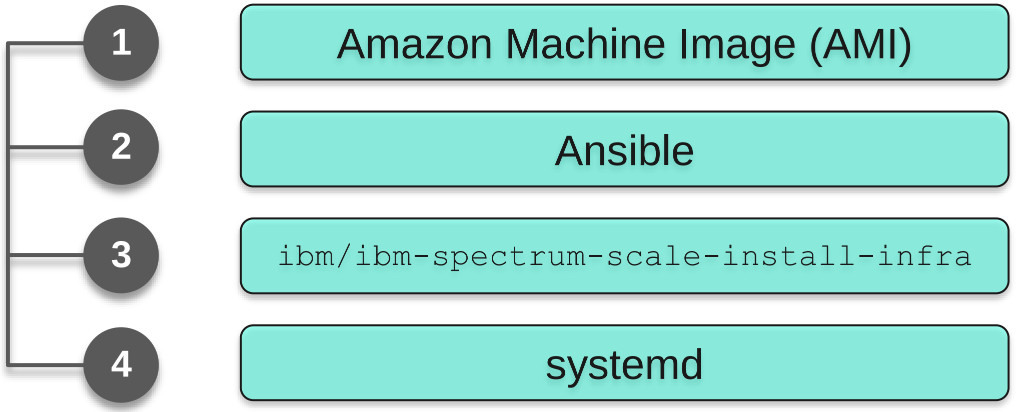
Sample Scenario II (public cloud)

- Public sector customer @ United Kingdom

- Goal
  - Dynamically spin up / down self-contained application instances on AWS Cloud
  - Terminate instances EOB to save cost

- Challenges
  - Nodes join & leave clusters "at will"
  - Spectrum Scale doesn't react well to dynamically changing IPs

- Solution
  - Two groups of EC2 instances:
    - Permanent (Quorum, NSD Servers, GUI)
    - Temporary (NSD Clients)
  - OS installation & configuration via custom AMIs
  - Spectrum Scale installation through Ansible
  - Spectrum Scale configuration through
    - systemd service units
    - `mmdelnode` / `mmaddnode`


1
Amazon Machine Image (AMI)
2
Ansible
3
ibm/ibm-spectrum-scale-install-infra
4
systemd

Sample Scenario II (public cloud)

- Use systemd service unit
  to run `mmdelnode` upon shutdown
  - Considered `mmsdrrestore`
  - Considered `mmchnode --spec-file`

- “Stale” nodes remain after crash
  - Need to reconcile cluster config (housekeeping)

- Firewall configuration
  - `firewalld` module [link]
  - Firewall role (Linux System Roles)
    - `rhel-system-roles` [link]
    - Don’t forget `permanent: yes`

- Optimization: pre-built kernel extension → pre-built AMI

1. Amazon Machine Image (AMI)
2. Ansible
3. `ibm/ibm-spectrum-scale-install-infra`
4. systemd

```
[Unit]
Description=...

[Service]
Type=oneshot
RemainAfterExit=true
ExecStop=/usr/local/bin/script.sh

[Install]
WantedBy=multi-user.target
```

Summary & Lessons Learned

- Each automation use case is different

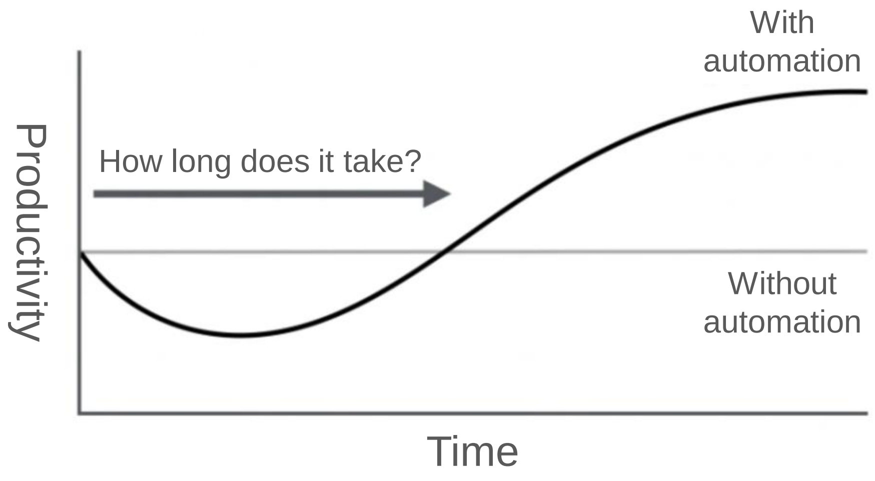
- DevOps / Infrastructure as Code is a process, not a tool
- Tools make certain things easy (sweet spot)
- “*If your only tool is a hammer then every problem looks like a nail*”
- IBM evaluated different options, purposely decided on best fit

- IBM provides integration points for Spectrum Scale … at various levels …

- Automation requires investment

- Automation is “all or nothing” (kind of)
  – Requires commitment

- Automation is never “done”



Thank you for using
IBM Spectrum Scale!