Configuration Management with Ansible

Key Takeaways

TECHWORLD WITH NANA

Introduction to Ansible

TECHWORLD WITH NANA

Ansible is a tool to automate IT tasks

Configure systems, deploy software or orchestrate more advanced IT tasks

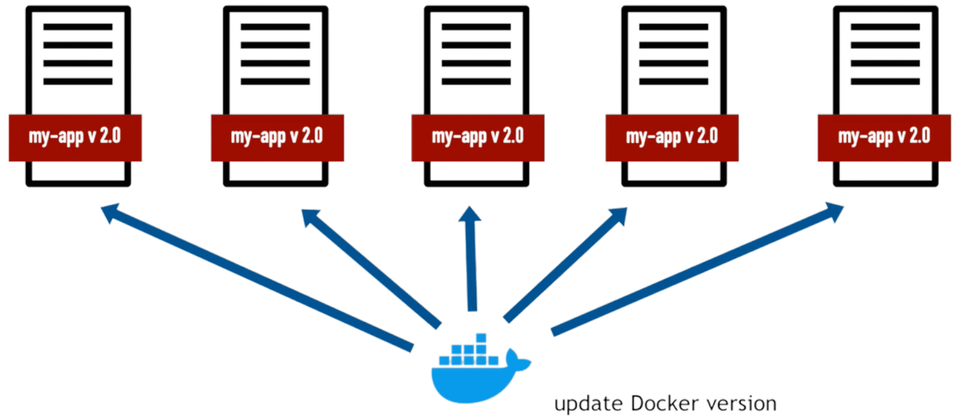
Use Cases for Ansible

For **repetitive tasks** like:

- Updates, Backups, Create Users & Assign Permissions, System Reboots, ...

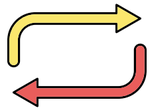
When you need the **same configuration on many servers**:

- Update all at the same time with Ansible



Ansible advantages

- Instead of SSH into all remote server, execute tasks from your own machine



- Configuration/Installation/Deployment steps in a single file

- Re-use same file multiple times for different environments

- More reliable and less error prone

- Supporting all infrastructure from OS to cloud providers

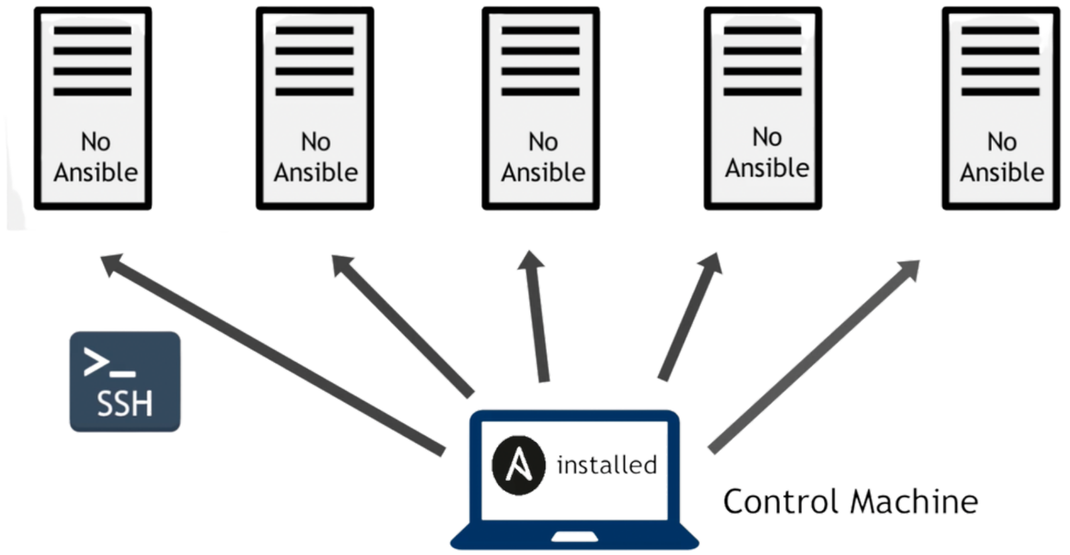
How Ansible works

Ansible uses YAML

- YAML is used, because it's easier for humans to read and write
- So, no need for learning a specific language for Ansible

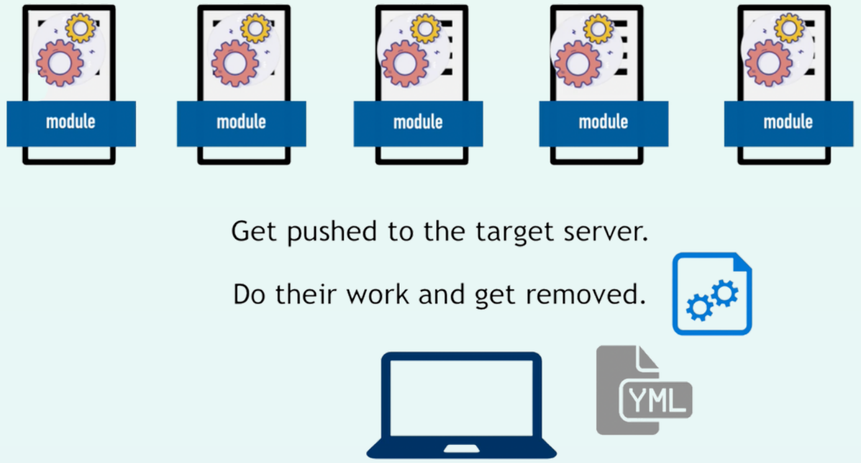
Ansible is agentless

- Ansible connects to remote servers using simple SSH, **no special agent required**



Ansible Modules

- A module is a **reusable, standalone script** that Ansible runs on your behalf
- Modules are **granular, performing a specific task**



Module Index

- All modules
- Cloud modules
- Clustering modules
- Commands modules
- Crypto modules
- Database modules
- Files modules
- Identity modules
- Inventory modules
- Messaging modules
- Monitoring modules
- Net Tools modules
- Network modules
- Notification modules
- Packaging modules

- You can develop own modules or **use existing ones**, for example "jenkins_job" module:

Creating a job:

Delete a job:

```yaml
# Create a jenkins job using basic authentication
- jenkins_job:
    config: "{{ lookup('file', 'templates/test.xml') }}"
    name: test
    password: admin
    url: http://localhost:8080
    user: admin

# Delete a jenkins job using the token
- jenkins_job:
    name: test
    token: asdfasfasfasdfasdfadfasfasdfasdfc
    state: absent
    url: http://localhost:8080
    user: admin
```

Ansible Playbooks - 1

- A Playbook basically **groups multiple modules together**, which gets executed in order from top to bottom
- So with a Playbook, you can orchestrate steps of any manual ordered process

A task consists of:

```
tasks:
  - name: Rename table foo to bar
    postgresql_table:
      table: foo
      rename: bar

  - name: Set owner to someuser
    postgresql_table:
      name: foo
      owner: someuser

  - name: Truncate table foo
    postgresql_table:
      name: foo
      truncate: yes
```

Module name

Arguments

Description of task

Where and who executes tasks?

```
- hosts: databases
  remote_user: root

  tasks:
    - name: Rename table foo to bar
      postgresql_table:
        table: foo
        rename: bar

    - name: Set owner to someuser
      postgresql_table:
        name: foo
        owner: someuser

    - name: Truncate table foo
      postgresql_table:
```

HOSTS:

- Defines where these tasks should get executed

REMOTE_USER:

- Defines with which user the tasks should be executed

Ansible Playbooks - 2

```yaml
hosts: webservers
remote_user: root

tasks:
  - name: create directory for nginx
    file:
      path: /path/to/nginx/dir
      state: directory

  - name: install nginx latest version
    yum:
      name: nginx
      state: latest

  - name: start nginx
    service:
      name: nginx
      state: started

- hosts: databases
  remote_user: root

  tasks:
    - name: Rename table foo to bar
      postgresql_table:
        table: foo
        rename: bar

    - name: Set owner to someuser
      postgresql_table:
        name: foo
        owner: someuser

    - name: Truncate table foo
      postgresql_table:
        name: foo
        truncate: yes
```

Play for Webservers

Play for Databases

Playbook consists of

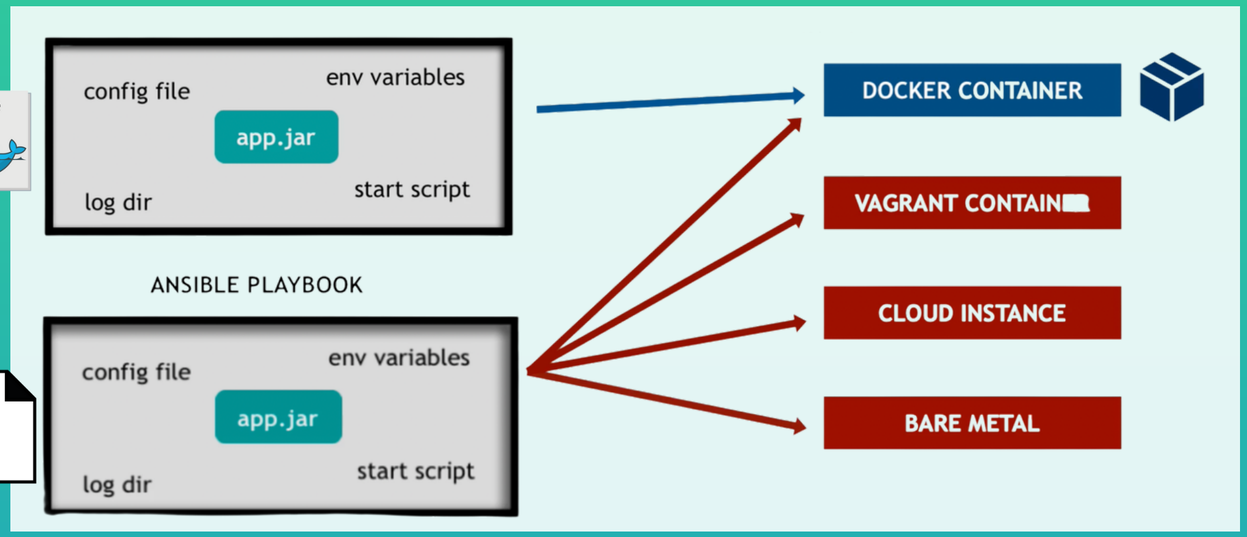
- 1 or more "plays" in an ordered list
- Each "play" executes part of the overall goal of the "playbook"
- A play runs 1 or more tasks
- Each task calls an Ansible module

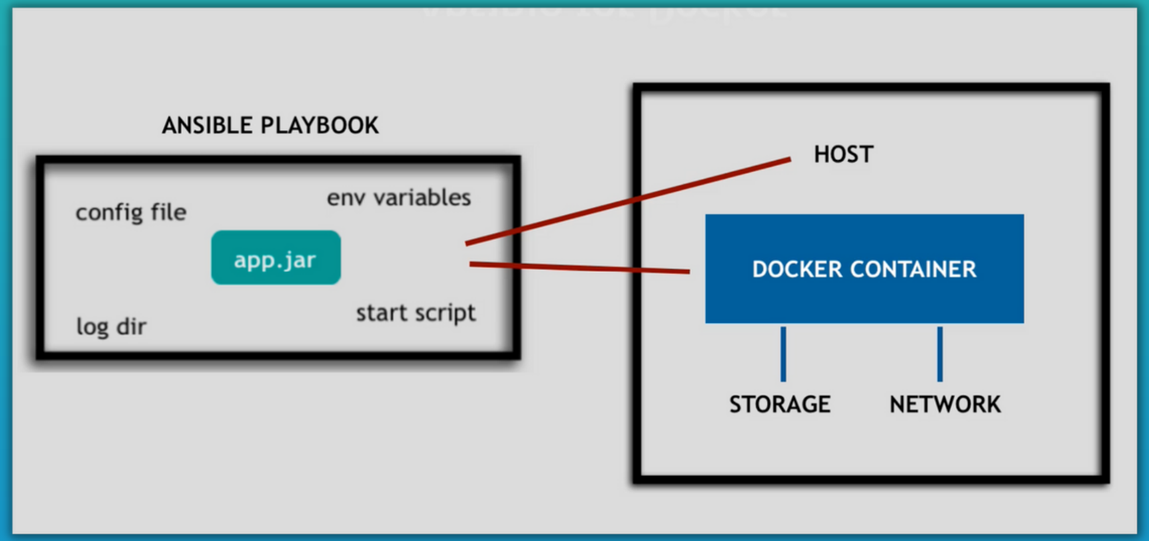
Playbook describes

- **How** and in **which order**
- At what **time** and **where** (on which machines)
- **What** (the modules) should be executed

Ansible for Docker Usage

- With Ansible you can create alternative to Dockerfile, which is **more powerful**
- You can define same type of configuration, but instead of creating only a Docker container, Ansible allows you to **reproduce the application across many environments**





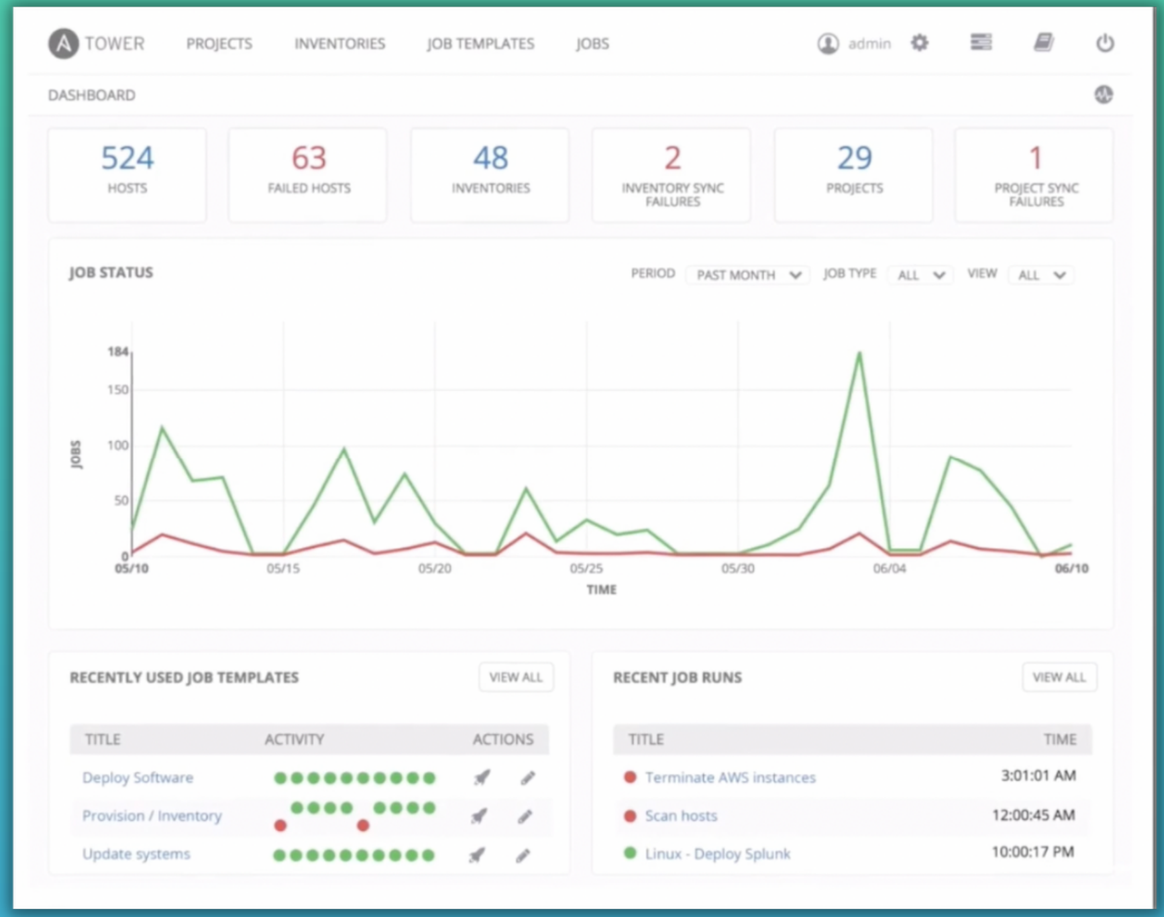
- **Manage both** container and its host

TECHWORLD WITH NANA

Ansible Tower

- A **web-based** solution that makes Ansible more easy to use

- Centrally store automation tasks
- Across teams
- Configure permissions
- Manage inventory



TECHWORLD WITH NANA

Comparable Tools

Ansible

- Simple YAML
- Agentless

Puppet and Chef

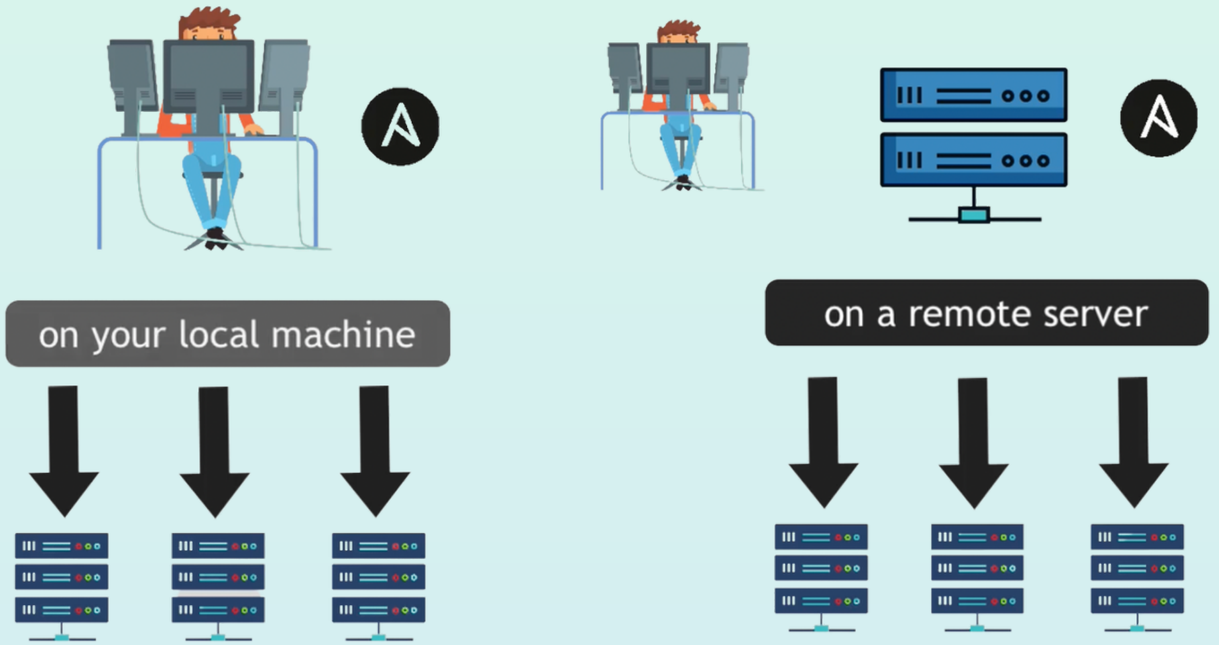
- Ruby more difficult to learn
- Installation needed
- Need for managing updates on target servers

Deep Dive into Ansible

TECHWORLD WITH NANA

Ansible Installation

- You can install Ansible either on your **local machine** or on a **remote server:**
- The machine that runs Ansible is called the **"Control Node"**
- Control Node manages the target servers
- Windows is not supported for the control node



Prerequisites: Ansible needs Python to run

Ansible Inventory and how to connect

Information in inventory file

File containing data about the Ansible **remote hosts** and **how to connect** to them:

1) Host IP-address or Host DNS-name, 2) SSH Private Key

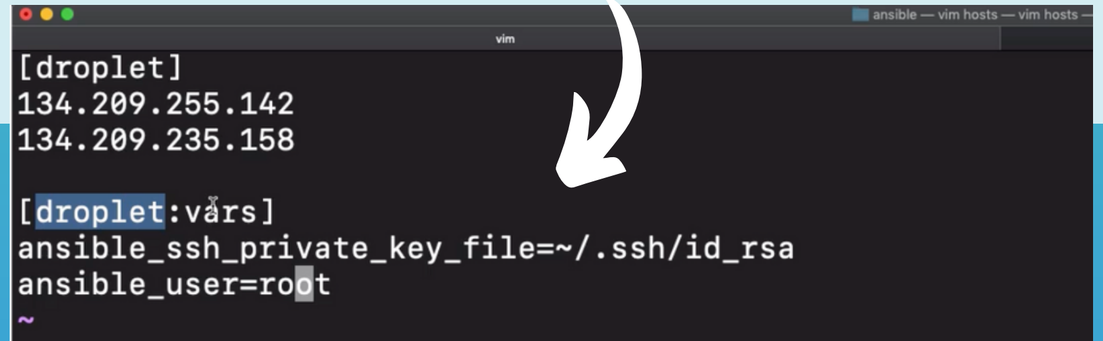
3) SSH User

Inventory parameters

- There are a number of variables you can use, to control how Ansible interacts with remote hosts (see Ansible docs)

Grouping hosts

- To address multiple servers, we **group them** based on their functionality or geo location etc.
- SSH key and user can be defined for whole group



You can create groups that track:

- WHERE - a datacenter/region, e.g. east, west
- WHAT - e.g. database servers, web servers etc.
- WHEN - which stage, e.g. dev, test, prod environment

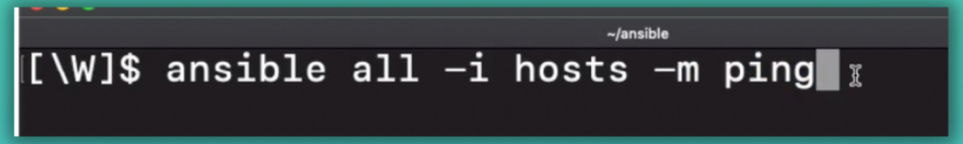
TECHWORLD WITH NANA

Ad-hoc commands

- Ad-hoc commands are not stored for future uses
- A fast way to **interact with the managed hosts**

```
$ ansible [pattern] -m [module] -a "[module options]"
```

Example:



[**pattern**] = targeting hosts and groups

- "all" = default group, which contains every host

[module] = which module you want to execute

Ansible Playbooks

Summary

- A Playbook can have multiple Plays
- A Play is a group of ordered tasks
- Plays and tasks runs in order from top to bottom

Infrastructure as Code

- Ansible configuration files are treated like code
- Create Ansible project and save in source control, like Git
- Git repo is your "single source of truth"

Example Play

- Play for Webserver
- Tasks for Play would be to:
  a. Install nginx
  b. Start nginx

"Gather Facts" Module

- Is automatically called by Playbooks
- To **gather useful variables about remote hosts**, that you can use in Playbooks
- So Ansible provides many *facts* about the system automatically

Using Ansible Modules

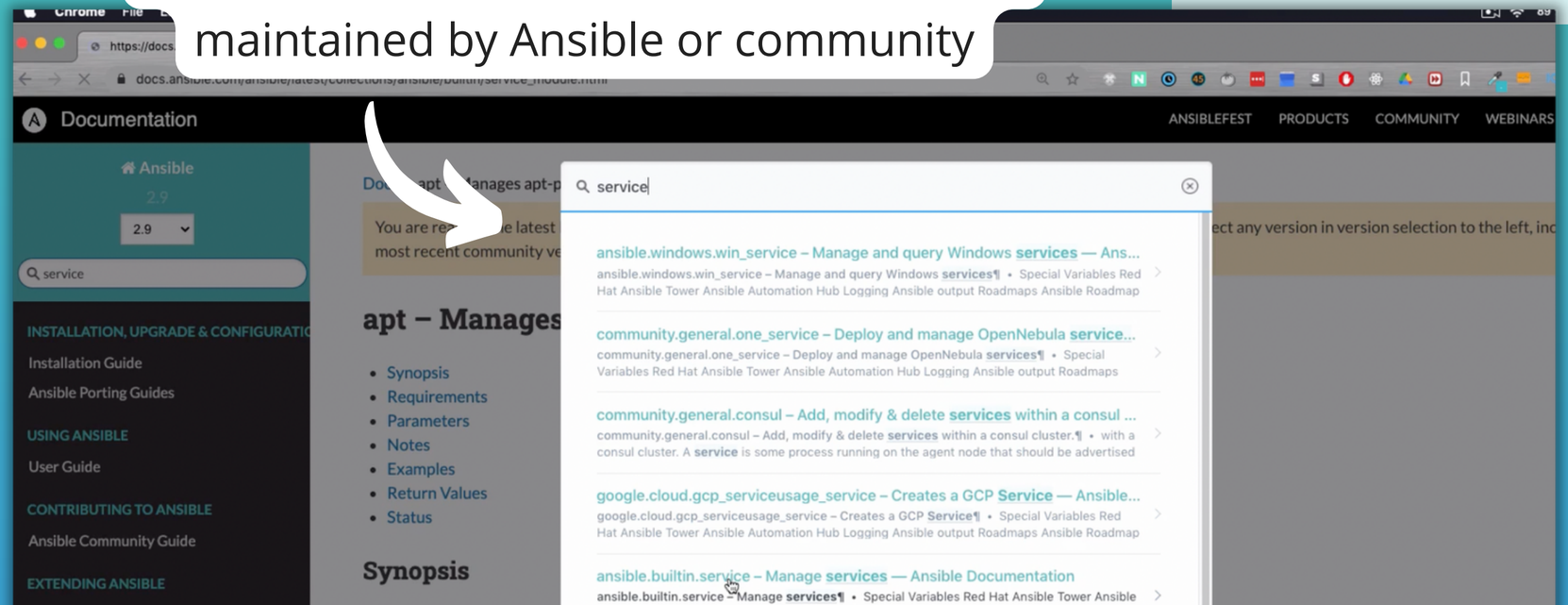
Summary

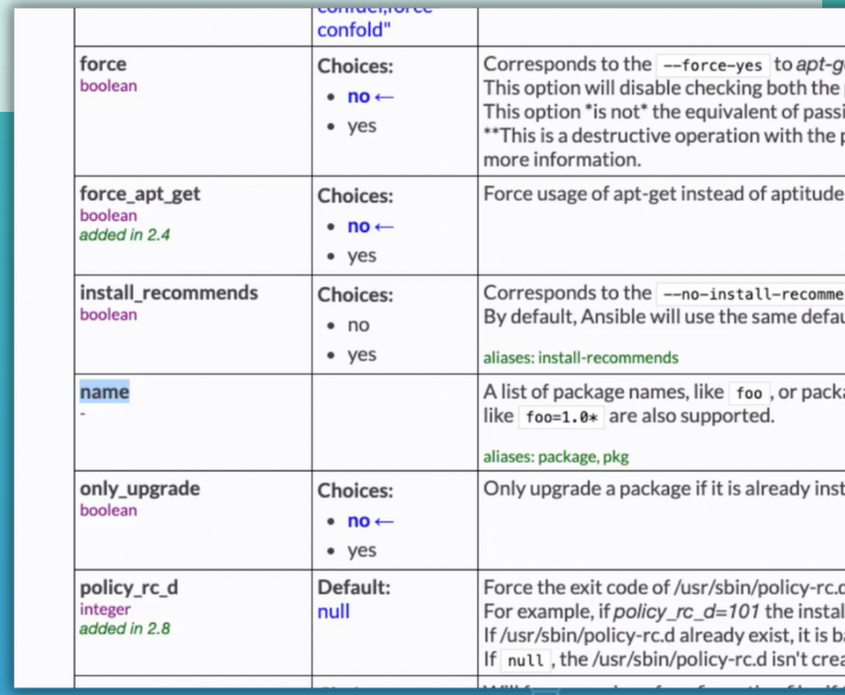
- Main building blocks of Ansible Playbooks
- Also referred to as "task plugins"
- Ansible executes a module usually on remote server
- And collects return values

Module Documentation

- You shouldn't learn modules by heart
- Instead **reference the module documentation**, which describes exactly how to use the module, which parameters you can configure with what values

You can **search for any module**, whether maintained by Ansible or community





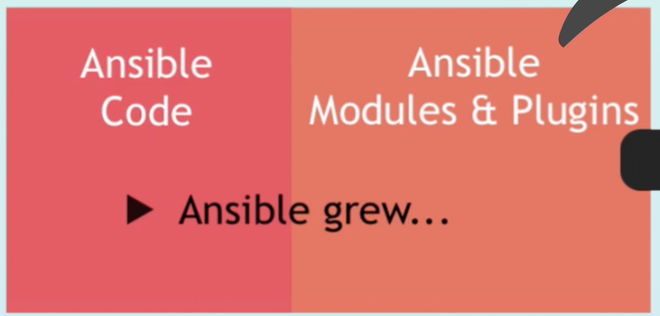
force boolean	Choices: • no ← • yes	Corresponds to the `--force-yes` to *apt-g* This option will disable checking both the This option *is not* the equivalent of passi **This is a destructive operation with the p more information.
force_apt_get boolean *added in 2.4*	Choices: • no ← • yes	Force usage of apt-get instead of aptitude
install_recommends boolean	Choices: • no • yes	Corresponds to the `--no-install-recomme` By default, Ansible will use the same defau aliases: install-recommends
name -		A list of package names, like `foo` , or packa like `foo=1.0*` are also supported. aliases: package, pkg
only_upgrade boolean	Choices: • no ← • yes	Only upgrade a package if it is already inst
policy_rc_d integer *added in 2.8*	Default: null	Force the exit code of /usr/sbin/policy-rc.d For example, if *policy_rc_d=101* the instal If /usr/sbin/policy-rc.d already exist, it is b If `null` , the /usr/sbin/policy-rc.d isn't crea

Ansible Changes

Ansible 2.9 and earlier

Classic Ansible

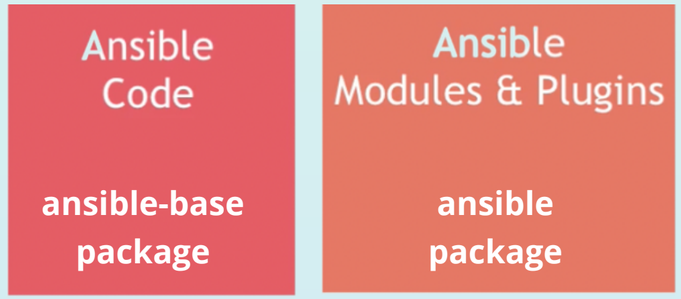
- All modules were included
- **Single repository** *ansible/ansible*
- Single package called ansible



MODULARIZE!

Ansible 2.10 and later

- ansible/ansible (ansible-base) repository contains the core Ansible programs
- **Modules and Plugins moved into various "collections"**

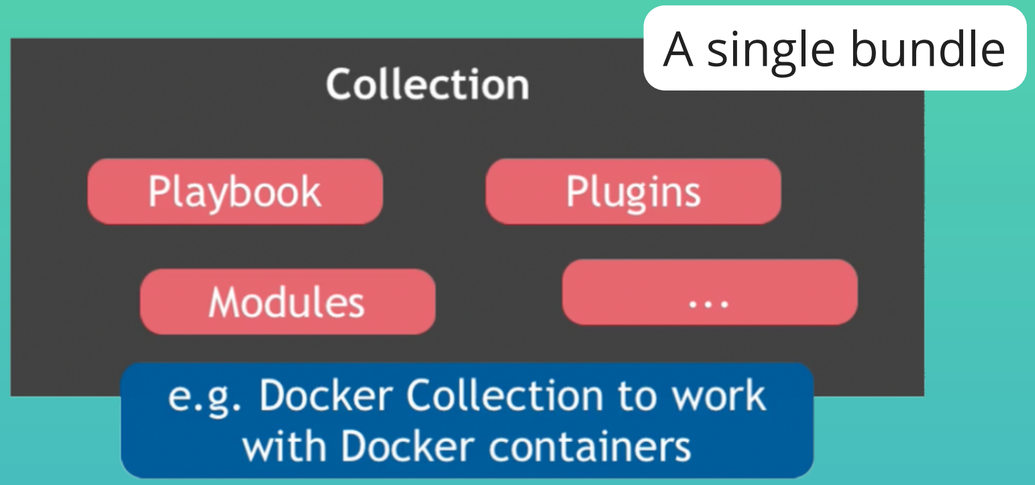


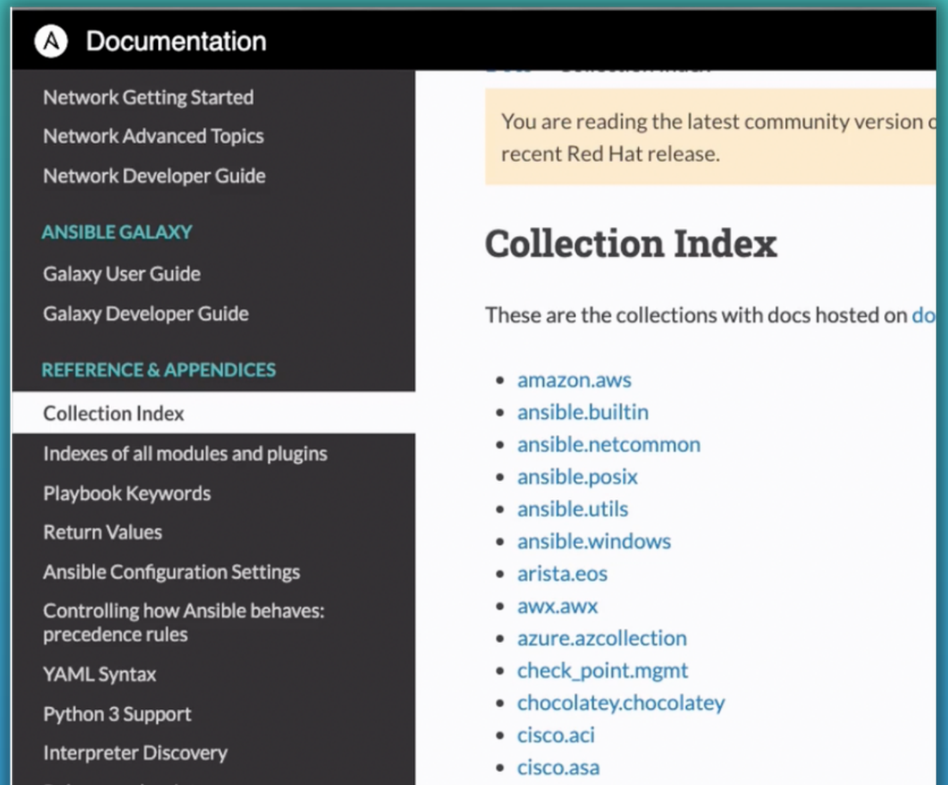
- Can be **released independently**
- Have their own repositories

TECHWORLD WITH NANA

Ansible Collections - 1

- A packaging format for bundling and distributing Ansible content
- Can be released and installed independent of other collections





- All modules are part of a collection
- Collections can also contain Plugins for example

Plugin

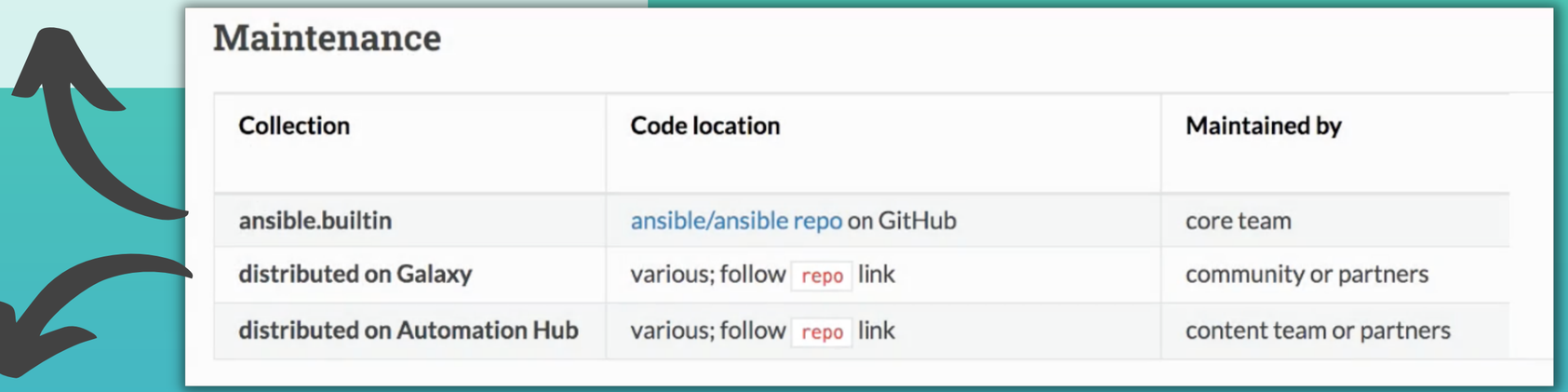
- Code that add to Ansible's functionality or modules
- Use existing plugins or write own ones

TECHWORLD WITH NANA

Ansible Collections - 2

Built-In Collections

- Many of the collections and modules are built-in



Maintenance

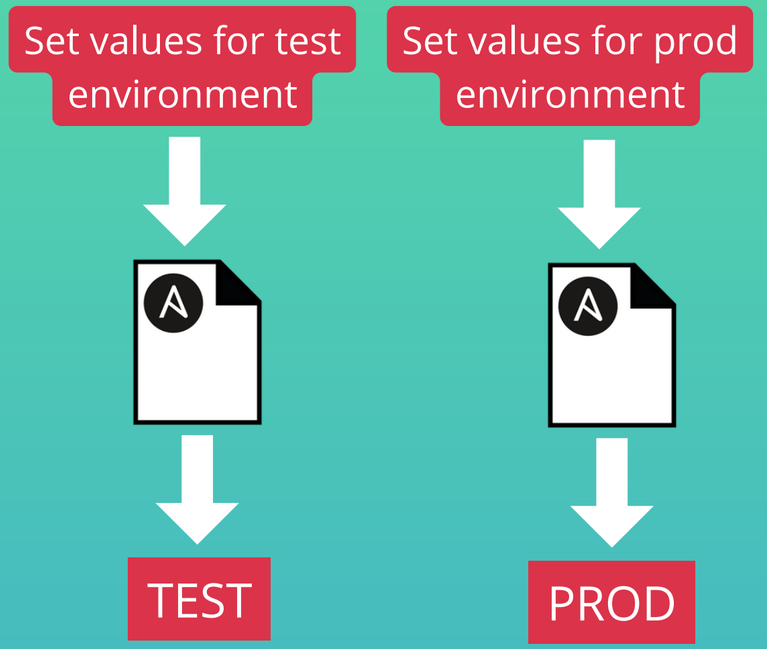
Collection	Code location	Maintained by
ansible.builtin	ansible/ansible repo on GitHub	core team
distributed on Galaxy	various; follow `repo` link	community or partners
distributed on Automation Hub	various; follow `repo` link	content team or partners

Ansible Galaxy

- Collections, which are not built-in can be installed from Ansible Galaxy
- It's an online hub for **finding and sharing Ansible community content**
- Comparable to Terrform Registry, PyPI etc.
- Also **CLI utility to install collections**

Ansible Variables - 1

- Variables can be used to **parameterize your Playbook** to make it customizable
- So we can use the same Ansbile script for different environments, by substituting some dynamic values



Registering variables

- With "register" you can **create variables from the output of an Ansible task**
- This variable can be used in any later task in your Play

```
21          group: admin
22        register: user_creation_result
23        - debug: msg={{user_creation_result}}
24
25    - name: Deploy nodejs app
26      hosts: 159.89.1.54
27      become: True
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

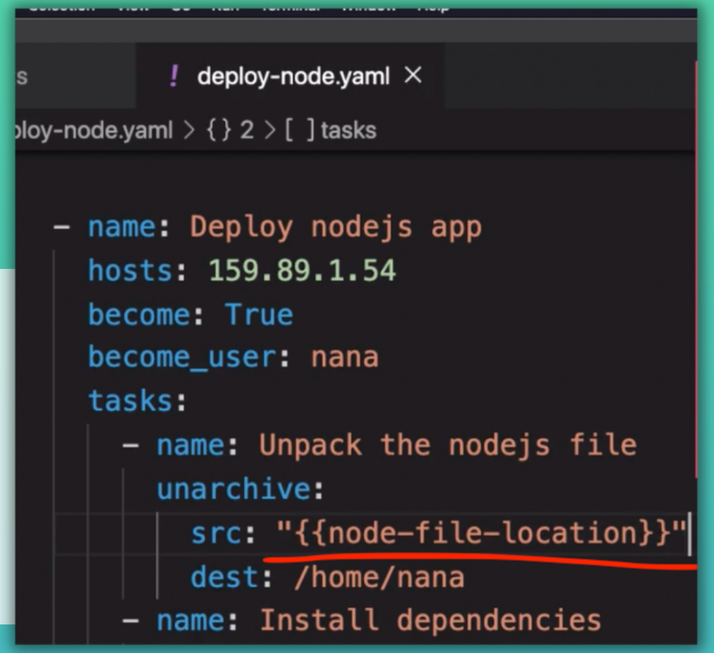
```
ok: [159.89.1.54] => {
    "msg": {
        "append": false,
        "changed": false,
        "comment": "Nana Admin",
        "failed": false,
        "group": 116,
        "home": "/home/nana",
        "move_home": false,
        "name": "nana",
        "shell": "/bin/sh",
        "state": "present",
        "uid": 1000
```

TECHWORLD WITH NANA

Ansible Variables - 2

Referencing variables

- Using **double curly braces**
- If curly braces {{ value }} directly after attribute, you must quote the whole expression to create valid YAML syntax



Naming of variables

- **Wrong**: Playbook keywords, such as environment
- **Valid**: Letters, numbers and underscores
- Should always start with a letter
- **Wrong**: linux-name, linux name, linux.name or 12
- **Valid**: linux_name

Ansible Variables - 3

- Once variables are configured in the Playbook, we need to **set the values:**

1) Setting vars directly in the Playbook

- Set value with vars attribute

```
hosts: 159.89.1.54
become: True
become_user: nana
vars:
  - location: /Users/nanajanashia/Demo-projects/Bootcamp/no
  - version: 1.0.0
  - destination: /home/nana
tasks:
  - name: Unpack the nodejs file
    unarchive:
      src: "{{location}}/nodejs-app-{{version}}.tgz"
      dest: "{{destination}}"
  - name: Install dependencies
    npm:
```

3) Using external configuration file

Best way!!

- Variables file uses YAML syntax

```
- name: Deploy nodejs app
  hosts: 159.89.1.54
  become: True
  become_user: "{{linux_name}}"
  vars_files:
    - project-vars
  tasks:
    - name: Unpack the nodejs file
```

2) Setting values on Command Line

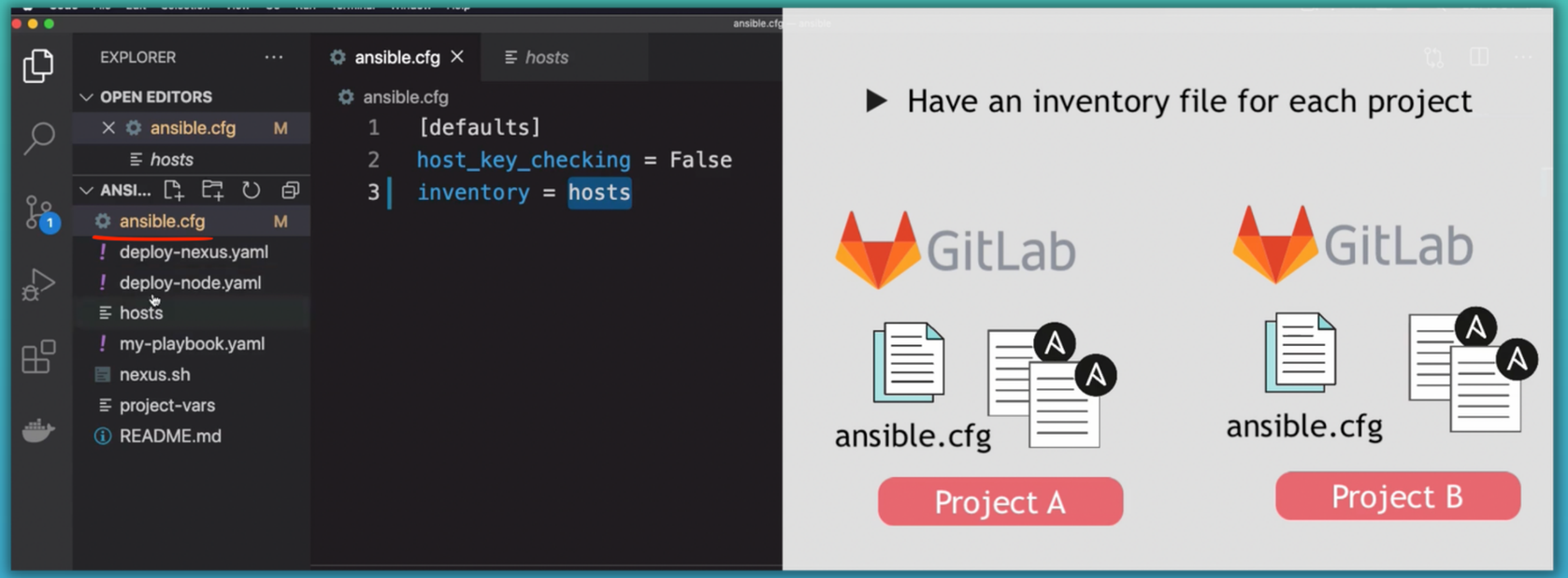
Options 1) and 2) are both impractical!

```
[\W]$ ansible-playbook -i hosts deploy-node.yaml -e "version=1.0.0 location=/Users/nanajanashia/Demo-projects/Bootcamp/nodejs-app"
```

TECHWORLD WITH NANA

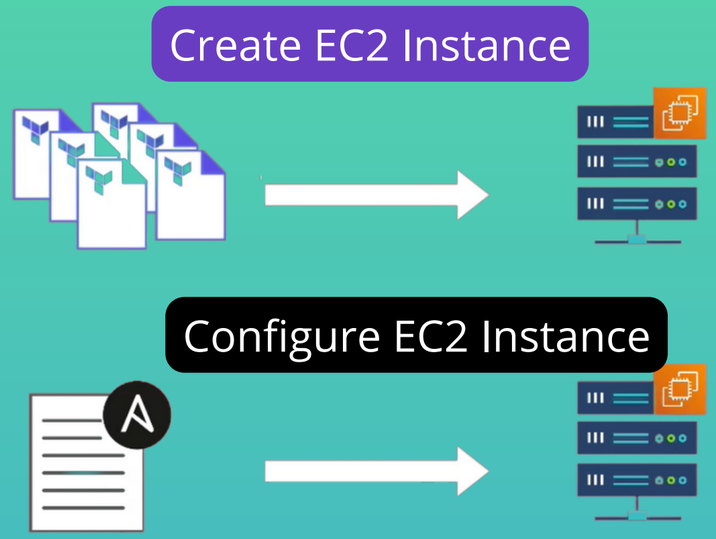
Ansible Configuration

- Ansible supports several sources for configuring its behavior, including an ini file named "ansible.cfg"
- You can configure the file globally or for each project, by creating ansible.cfg file in the project



Terraform and Ansible - 1

- With Terraform we automate provisioning the infrastructure (creating EC2 instances)
- With Ansible we now can automate the whole configuration



Integrate Ansible Playbook Execution in Terraform

Now we need to integrate the Ansible exeuction to **avoid the manual steps between provisioning and configuring of**:

- Getting the IP address manually from TF output
- Update the hosts file manually
- Execute Ansible command

Terraform and Ansible - 2

Terraform Side

- Configure Terraform to execute Ansible Playbook and pass all needed values to connect to the server

```
124   availability_zone           = var.avail_zone
125
126   tags = {
127     Name = "${var.env_prefix}-server"
128   }
129
130   provisioner "local_exec" {
131     working_dir = "/Users/nanajanashia/Demo-projects/Bootcamp/ansible"
132     command = "ansible-playbook --inventory ${self.public_ip}, --private-key  deplo
133   }
134 }
```

Ansible Side

- Ansible will match "all" hosts and also first check and wait until the server is fully initialized

deploy-docker-new-user.yaml

```
3     hosts: all
4     gather_facts: False
5     tasks:
6       - name: Ensure ssh port open
7         wait_for:
8           port: 22
9           delay: 10
10          timout: 100
11          search_regex: OpenSSH
12          host: '{{ (ansible_ssh_host|default(ansible_host))|default(inv
13        vars:
14          ansible_connection: local
15
16
17  - name: Install python3, docker, docker-compose
18    hosts: all
19    become: yes
```

Dynamic Inventory

Why?

- Managing an inventory, which fluctuates over time
- Hosts spinning up and shutting down all the time
- E.g. auto-scaling to accomodate for business demands

project-vars | hosts

hosts

```
1  [nexus_server]
2  134.122.92.122 ansible_ssh_private_key_file=~/.ssh/id_rsa ansible_user=root
3
4  [docker_server]
5  35.180.25.170 ansible_ssh_private_key_file=~/.ssh/id_rsa ansible_user=ec2-use
```

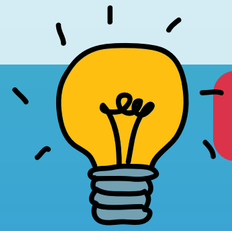
hard-coding the IP addresses is not a good idea!

We want to **dynamically set them**

How it works

Use **inventory plugins** or **inventory scripts** to

- connect to AWS account (or any other cloud provider platform)
- Get server information



Plugins are recommended over scripts!

Inventory Plugins

- Make use of Ansible features
- Written in YAML, while scripts are written in Python

TECHWORLD WITH NANA

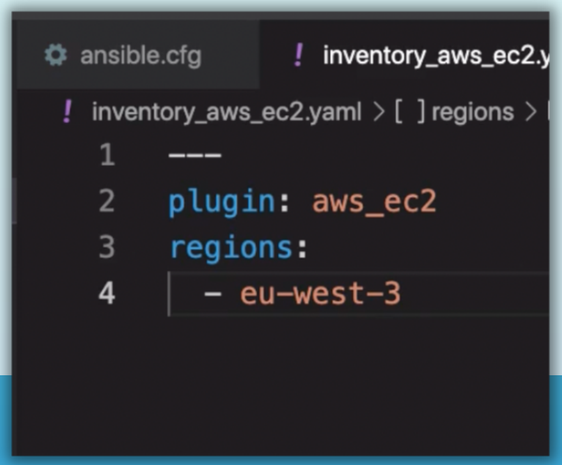
Dynamic Inventory for EC2 servers - 1

Plugin for AWS

- You can use "ansible-doc -t inventory -l" to see list of available plugins
- Available plugins for each specific infrastructure provider
- For AWS we have: "aws_ec2" plugin

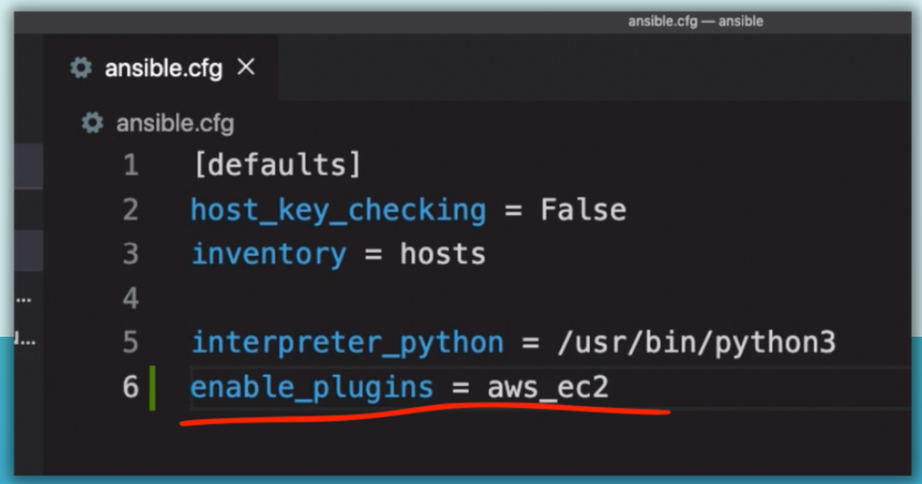
Write plugin file configuration

- Config file must end with "aws_ec2.yaml"



Prerequisites

- boto3 (to create, configure and manage AWS services)
- Enable plugin



TECHWORLD WITH NANA

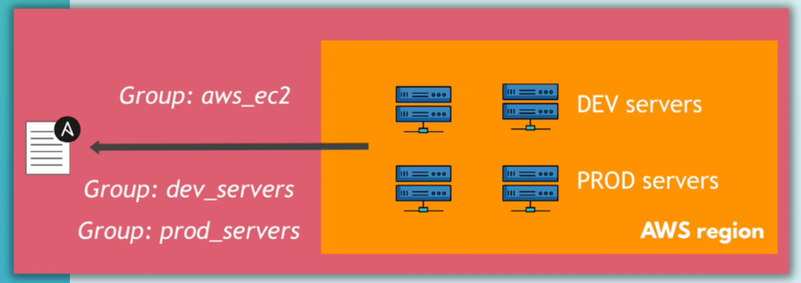
Dynamic Inventory for EC2 servers - 2

- To check result of plugin configuration:

```
[\W (master)]$ ansible-inventory -i inventory_aws_ec2.yaml --list
```

Dynamic Groups

- Get servers grouped by ...



inventory_aws_ec2.yaml > { } filters > instance-state-name

```yaml
---
plugin: aws_ec2
regions:
  - eu-west-3
filters:
  tag:Name: dev*
  instance-state-name: running
```

Custom Tags

- Get servers with custom tags, which we can address in Ansible Playbook

ansible.cfg | deploy-docker-new-user.yaml

inventory_aws_ec2.yaml > [ ] keyed_groups > { } 0 > pref

```yaml
plugin: aws_ec2
regions:
  - eu-west-3
keyed_groups:
  - key: tags
    prefix: "tag"
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
|  |--ec2-35-180-137-149.eu-west-3.compute.amaz
|--@ungrouped:
[\W (master)]$ ansible-inventory -i inventory_aws
@all:
|--@aws_ec2:
|  |--ec2-15-237-106-205.eu-west-3.compute.amaz
|  |--ec2-15-237-124-203.eu-west-3.compute.amaz
|  |--ec2-15-237-24-155.eu-west-3.compute.amazo
|  |--ec2-35-180-137-149.eu-west-3.compute.amaz
|--@tag_Name_dev_server:
|  |--ec2-15-237-124-203.eu-west-3.compute.amaz
|  |--ec2-15-237-24-155.eu-west-3.compute.amazo
|--@tag_Name_prod_server:
|  |--ec2-15-237-106-205.eu-west-3.compute.amaz
|  |--ec2-35-180-137-149.eu-west-3.compute.amaz
|--@ungrouped:
```

Ansible Roles - 1

- You can **group your content in roles** to easily reuse and share them with other users
- Break up large Playbooks into smaller manageable files

Usage of roles in your play

```yaml
- name: Create new linux user
  hosts: all
  become: yes
  vars:
    user_groups: adm,docker
  roles:
    - create_user


- name: Start docker containers
  hosts: all
  become: yes
  become_user: nana
  vars_files:
    - project-vars
  roles:
    - start_containers
```

Roles can contain

- **Tasks** that the role executes
- **Static files** that the role deploys
- (Default) **variables** for the tasks (which you can overwrite)
- **Custom modules**, which are used within this role

Ansible Roles - 2

- Roles are like small applications
- Easy to maintain and reuse
- Develop and test separately

Standard file structure

- A role has a defined directory structure
- So it's easy to navigate and maintain

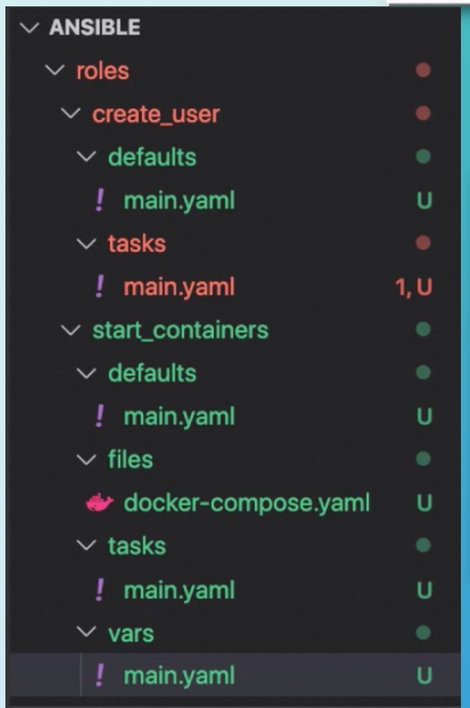
```
# playbooks
site.yml
webservers.yml
fooservers.yml
roles/
    common/
        tasks/
        handlers/
        library/
        files/
        templates/
        vars/
        defaults/
        meta/
    webservers/
        tasks/
        defaults/
        meta/
```

Use existing roles

- Write own roles or use existing ones from community
- Download from Ansible Galaxy or Git repository

Like functions

- Extract common logic
- Use function/role in different places with different parameters



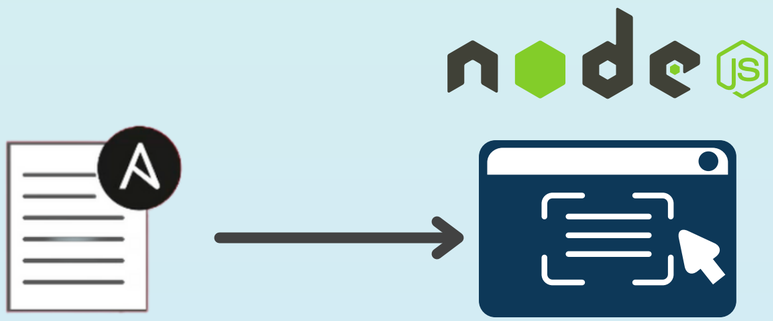

```
- name: Create new
  hosts: all
  become: yes
  vars:
    user_groups: adm
  roles:
    - create_user
```

TECHWORLD WITH NANA

Demo Projects - 1

Deploy Node App

1. Create a Droplet
2. Write Ansible Playbook
    a. Install node & npm on server
    b. Copy node artifact and unpack
    c. Start application
    d. Verify app running successfully



Automate Nexus Deployment

1. Create a Droplet
2. Write Ansible Playbook
    a. Download Nexus binary and unpack
    b. Run Nexus application using Nexus user

Deploy Docker container

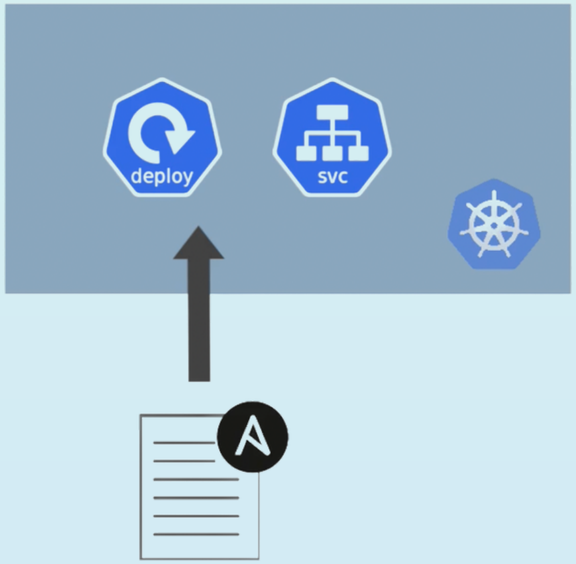
1. Create AWS EC2 instance (with TF)
2. Configure inventory file to connect to instance
3. Write Ansible Playbook
    a. Install Docker and docker-compose
    b. Copy docker-compose file to server
    c. Start Docker containers to run application

TECHWORLD WITH NANA

Demo Projects - 2

Automate K8s Deployment

1. Create K8s cluster on AWS
2. Configure Ansible to connect to EKS cluster
3. Deploy Deployment & Service resources



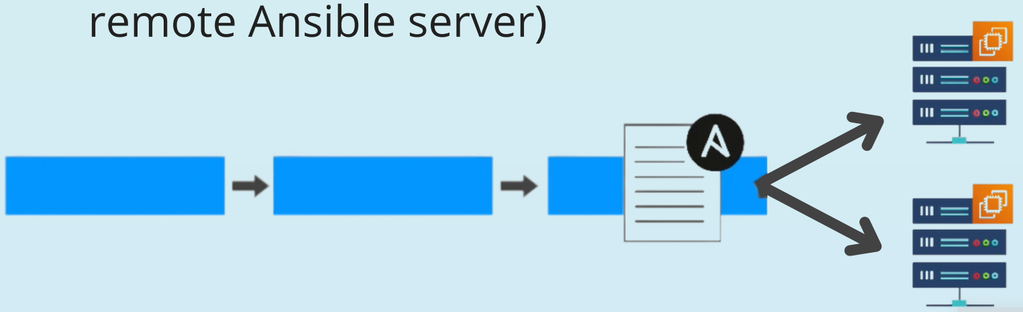
Ansible Integration in Jenkins CI/CD

1. Create a DO server for Jenkins
2. Create dedicated server for Ansible
3. Install Ansible on that server

Run Ansible locally on Jenkins

Or have a dedicated server for Ansible

4. Execute Ansible Playbook from Jenkins Pipeline (by creating Jenkinsfile that executes Playbook on the remote Ansible server)



Best Practices & Tips

Official Best Practices

- https://aap2.demoredhat.com/decks/ansible_best_practices.pdf
- https://docs.ansible.com/ansible/latest/user_guide/playbooks_best_practices.html
- https://www.ansible.com/blog/ansible-best-practices-essentials

Tip

- You can access the documentation for each module from the command line with the ansible-doc tool