



Configuration Management with Ansible

Key Takeaways

Introduction to Ansible

Ansible is a tool to automate IT tasks

Configure systems, deploy software or orchestrate more advanced IT tasks

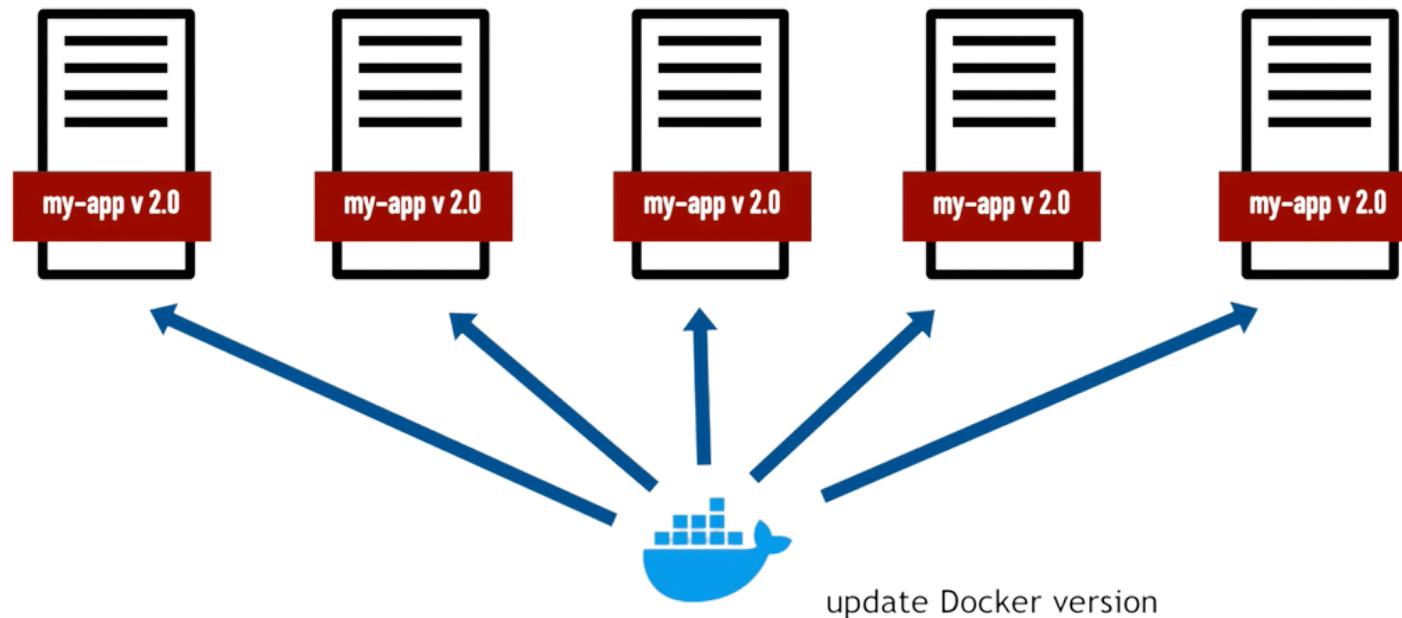
Use Cases for Ansible

For **repetitive tasks** like:

- Updates, Backups, Creating Users & Assigning Permissions, Reboots...

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

- Update all at the same time with Ansible

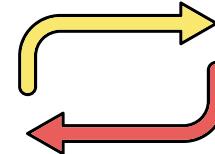


Ansible advantages



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

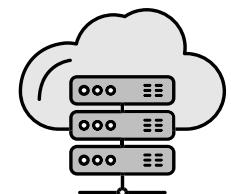
- Configuration/Installation/Deployment steps in a single file



- Re-use same file multiple times for different environments



- More reliable and less error prone



- Supports all infrastructure from OS to cloud providers

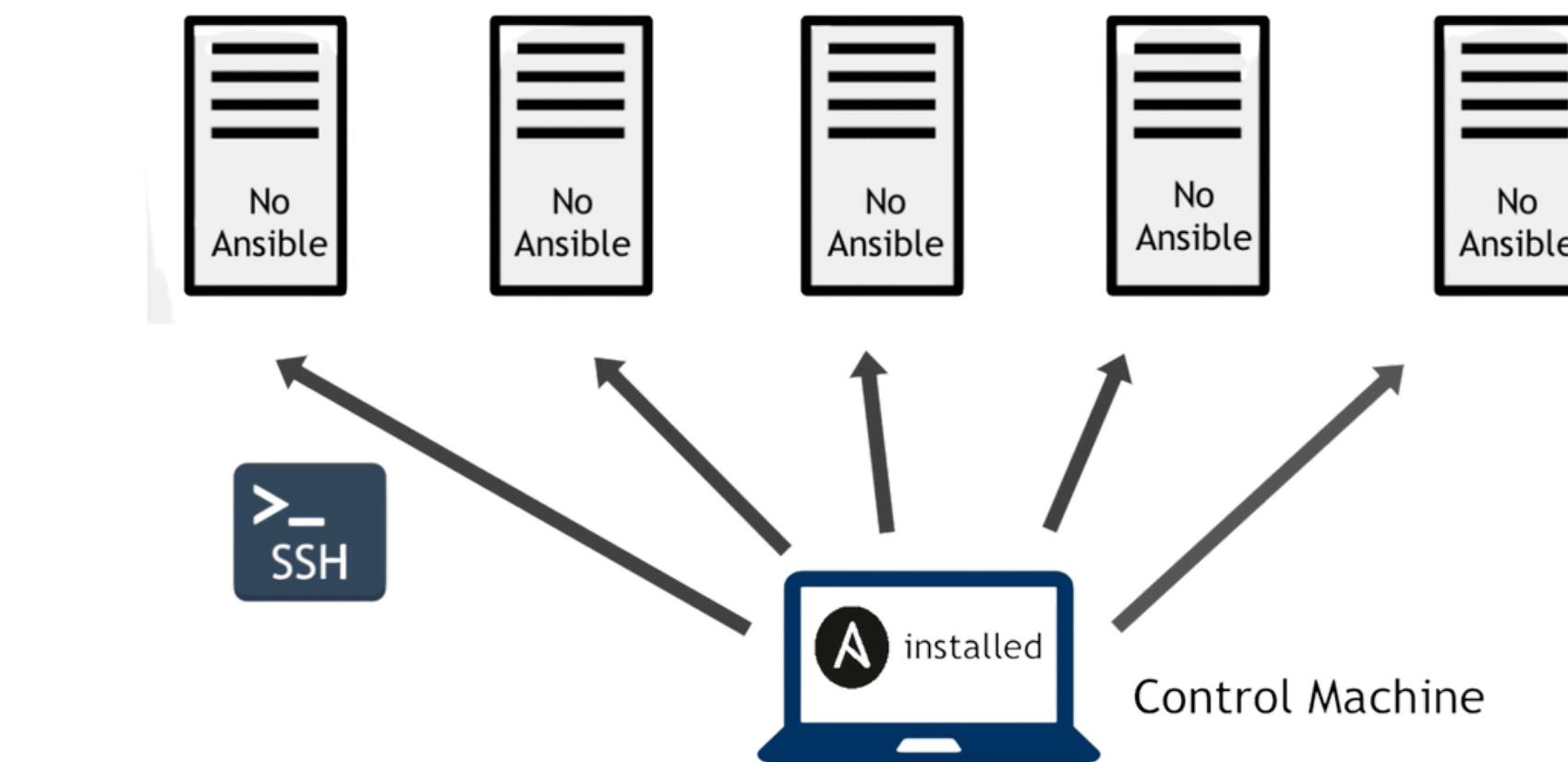
How Ansible works

Ansible uses YAML

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

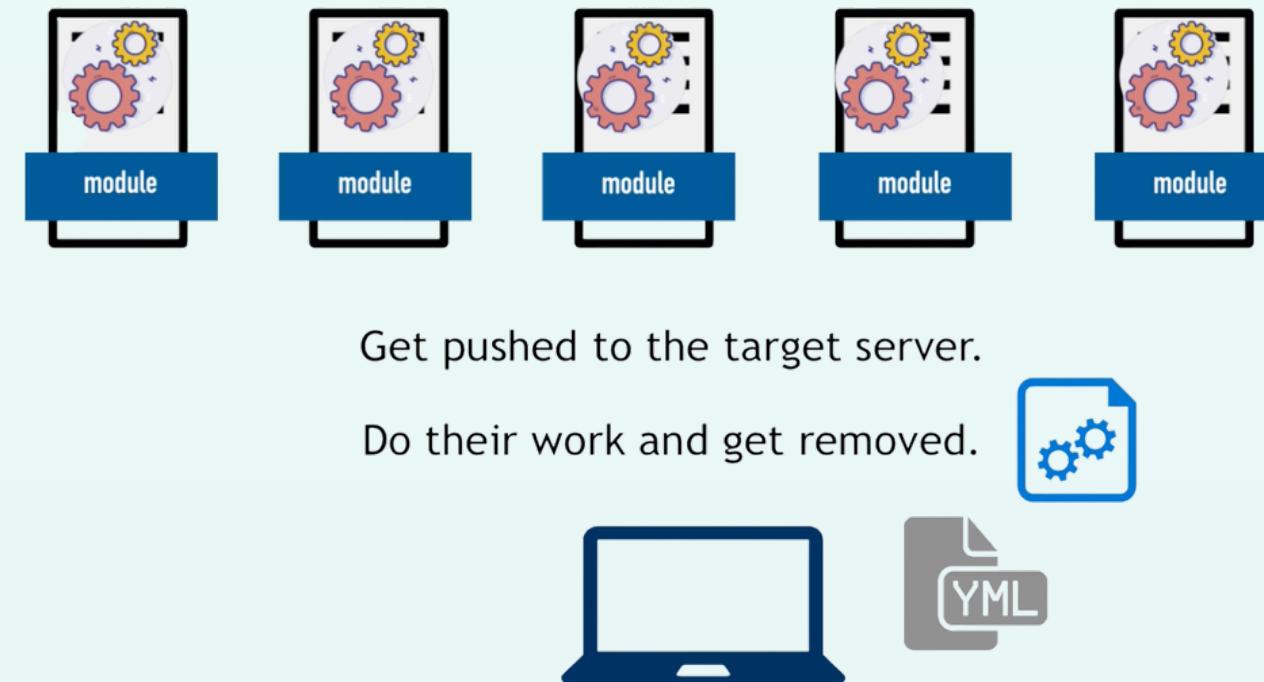
Ansible is agentless

- 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 your own modules or **use existing ones**, for example "jenkins_job" module:

Creating a job:

```
# 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 job:

```
# Delete a jenkins job using the token
- jenkins_job:
  name: test
  token: asdfasfasfasdfasdfasfasdfasdfc
  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 the 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:  
      name: foo
```

HOSTS:

- Defines where these tasks should get executed

REMOTE_USER:

- Defines with which user the tasks should be executed

Ansible Playbooks - 2

```
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

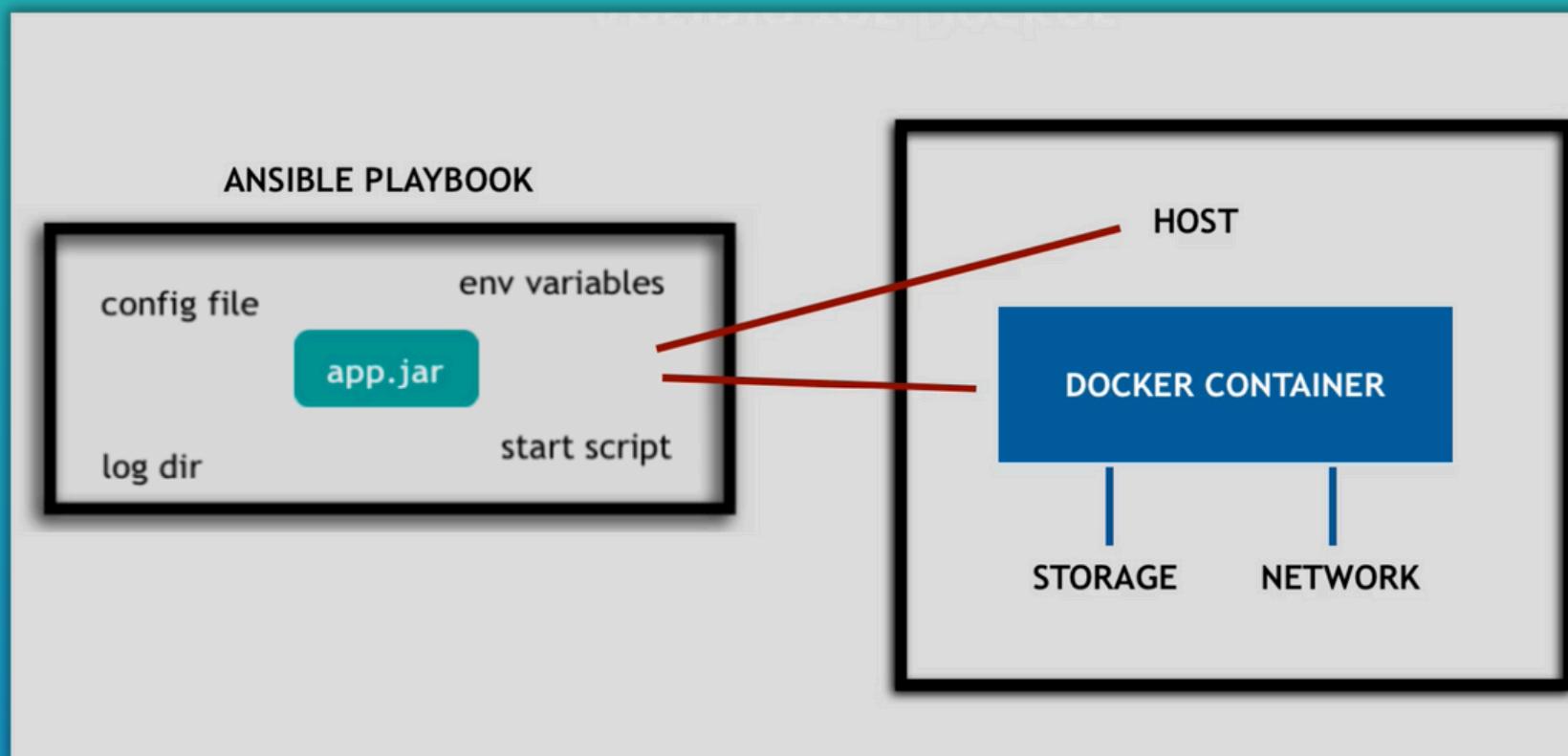
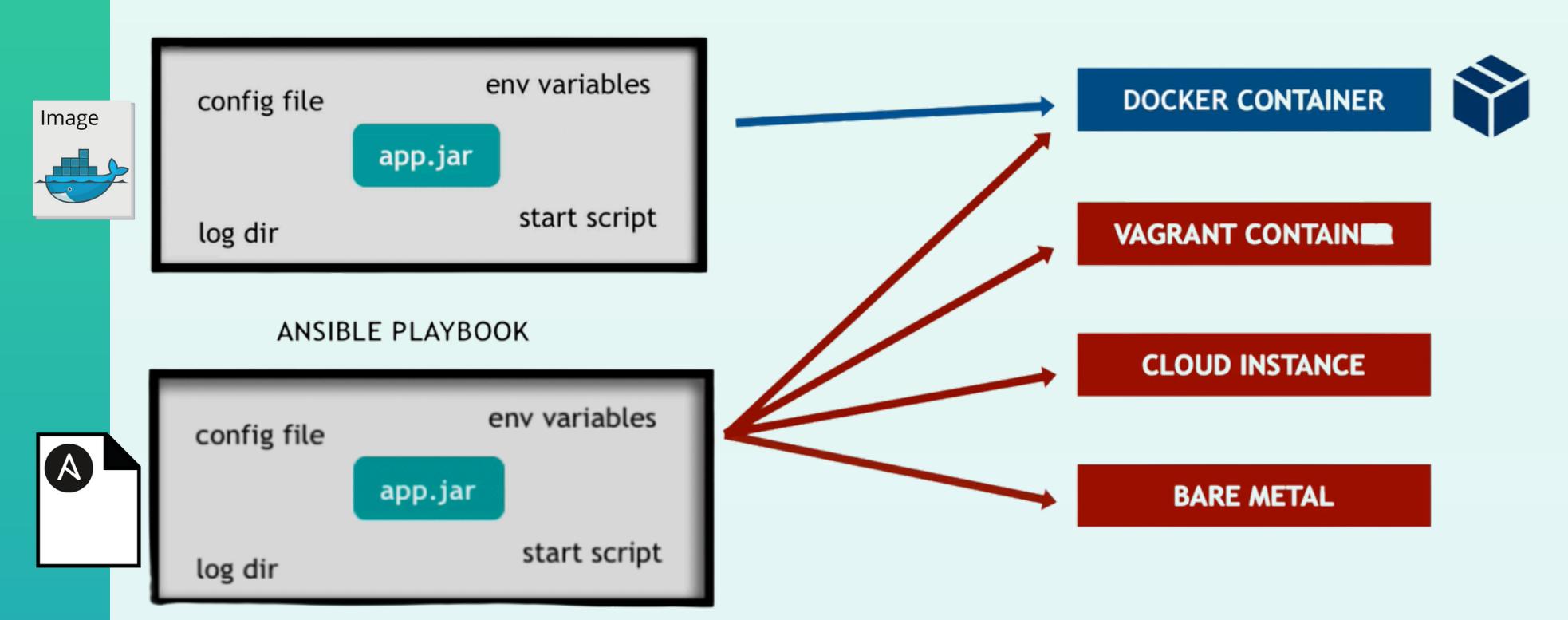
- 1 or more "plays" in an ordered list
- Each "play" executes part of the overall goal of the "playbook"
- A play runs 1 or more tasks
- Each task calls an Ansible module

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

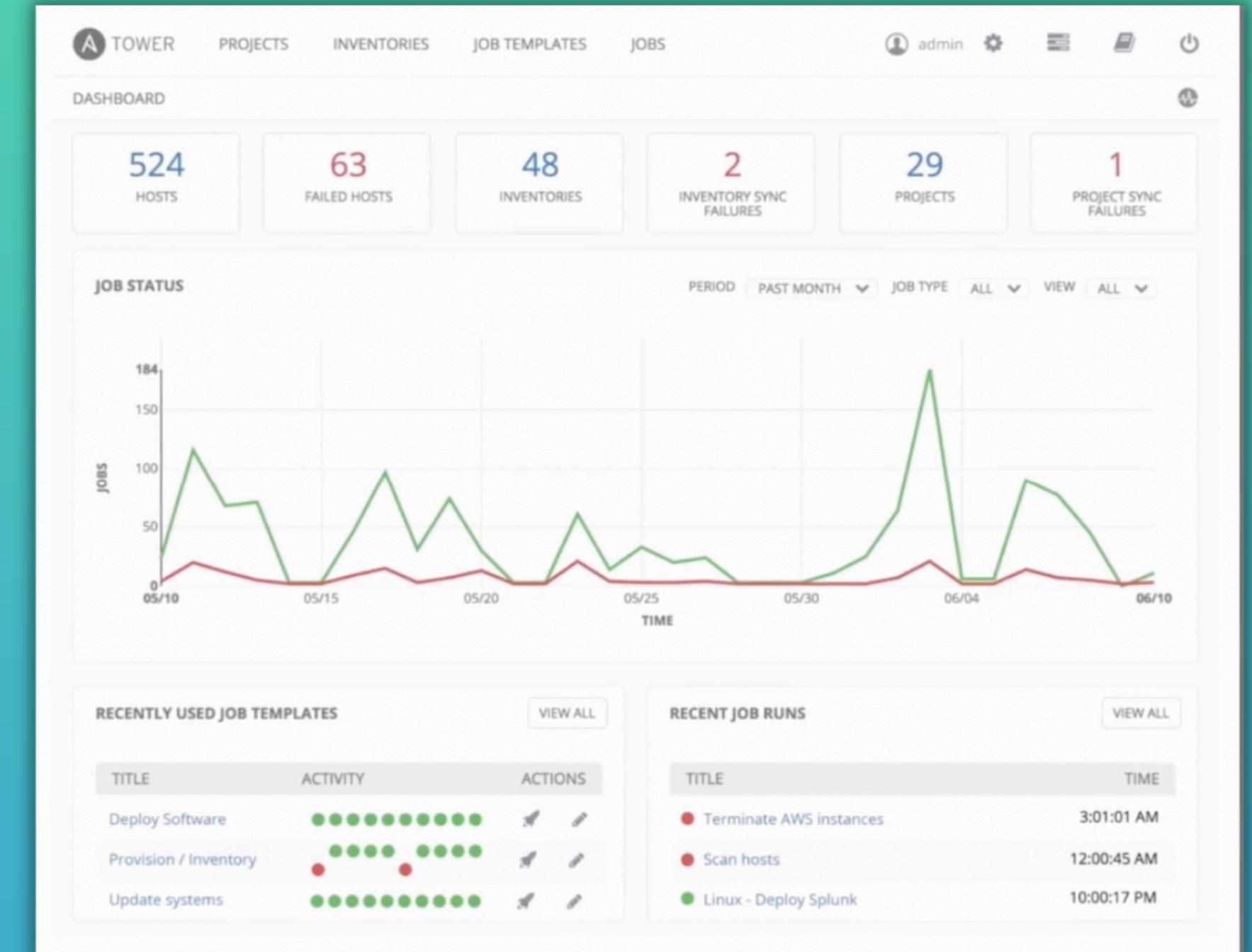


- Manage **both** container and its host

Ansible Tower

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

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



Comparable Tools

Ansible

- Simple YAML
- Agentless



Puppet and Chef

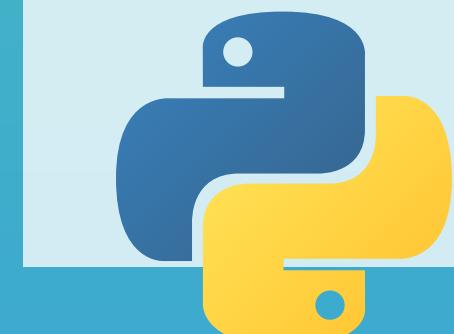
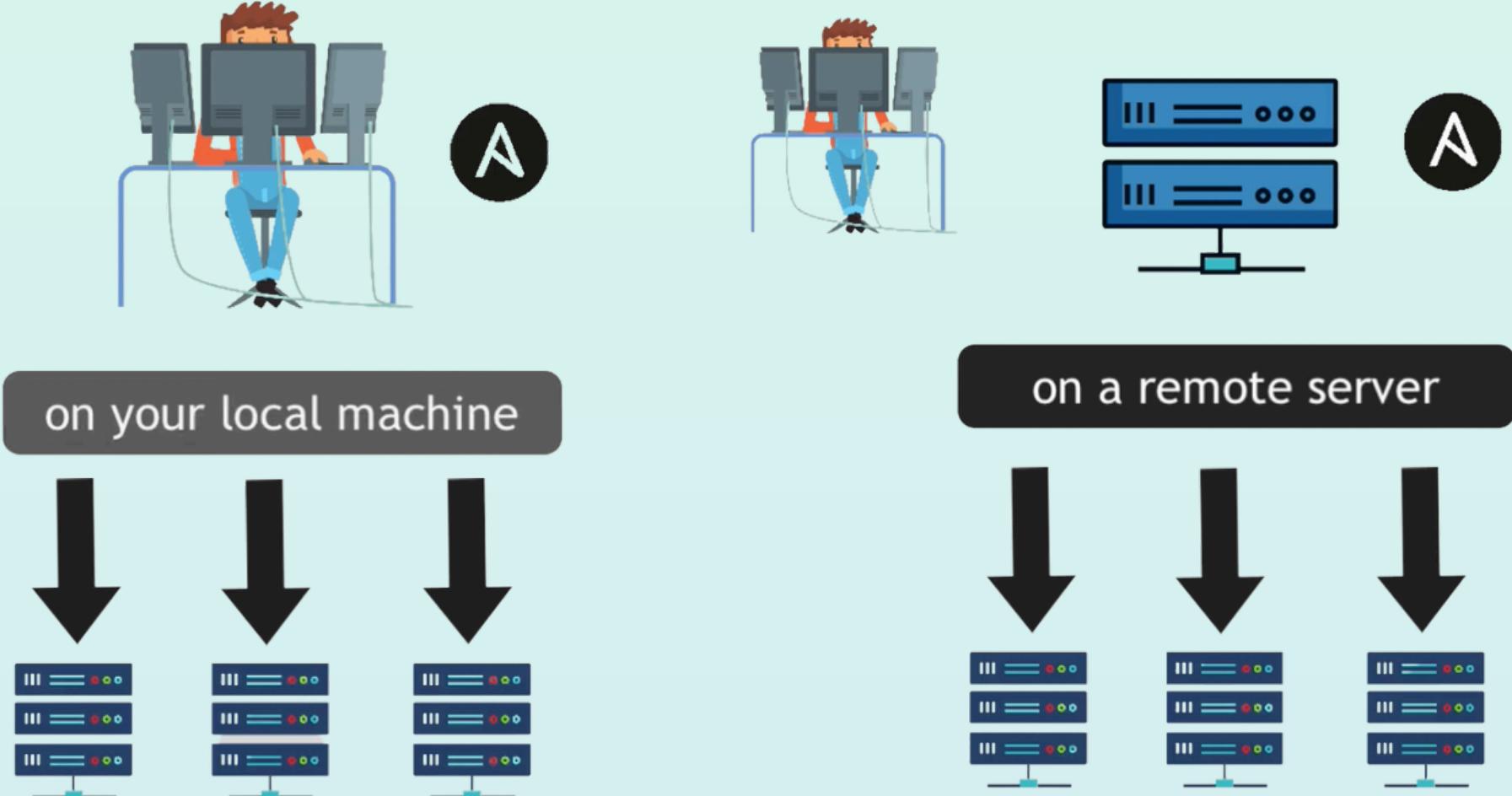
- Ruby more difficult to learn
- Installation needed
- Need for managing updates on target servers



Deep Dive into Ansible

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

```
● ● ●
[nodename]
192.168.1.100
192.168.1.101

[nodename:vars]
ansible_ssh_private_key_file=~/ssh/id_rsa
ansible_user=root
```

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

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:

```
nana@macbook /Users/nana
% ansible all -i hosts ping
```

[pattern] = targeting hosts and groups

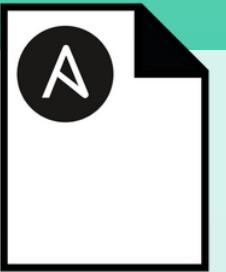
- "all" = default group, which contains every host

[module] = which module you want to execute

Ansible Playbooks

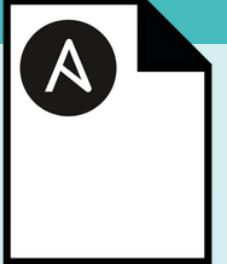
Summary

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



Example Play

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



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"



"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



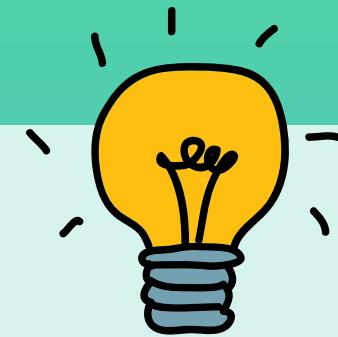
You can **search for any module**, whether maintained by Ansible or community

The screenshot shows the Ansible Community website's search results for the term 'service'. The search bar at the top contains 'service'. Below it, a list of modules is displayed:

- inspur.sm.service_info module – Get service information. — Ansible Commu...
- inspur.sm.service_info module – Get service information. · ansible-core Roadmaps Ansible Collection Index Collections in the Inspur Namespace Inspur.Sm inspur.sm.service_info module –
- inspur.sm.edit_service module – Set service settings. — Ansible Community ...
- inspur.sm.edit_service module – Set service settings. · ansible-core Roadmaps Ansible Collection Index Collections in the Inspur Namespace Inspur.Sm inspur.sm.edit_service module
- community.general.docker_service — Ansible Community Documentation
- community.general.docker_service · Ansible community.general.docker_service
- fortinet.fortimanager.fmgr_wanprof_system_sdwan_service_sla module – \$...
- fortinet.fortimanager.fmgr_wanprof_system_sdwan_service_sla module – Service level agreement · Ansible Roadmap ansible-core Roadmaps Ansible Collection Index Collections in
- netapp.ontap.na_ontap_service_policy module – NetApp ONTAP service pol...
- netapp.ontap.na_ontap_service_policy module – NetApp ONTAP service policy configuration · output Roadmaps Ansible Roadmap ansible-core Roadmaps Ansible
- vmware.vmware_rest.appliance_vmon_service_info module – Returns the st...
- vmware.vmware_rest.appliance_vmon_service_info module – Returns the state of a service. · Roadmaps Ansible Roadmap ansible-core Roadmaps Ansible Collection Index Collections in

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

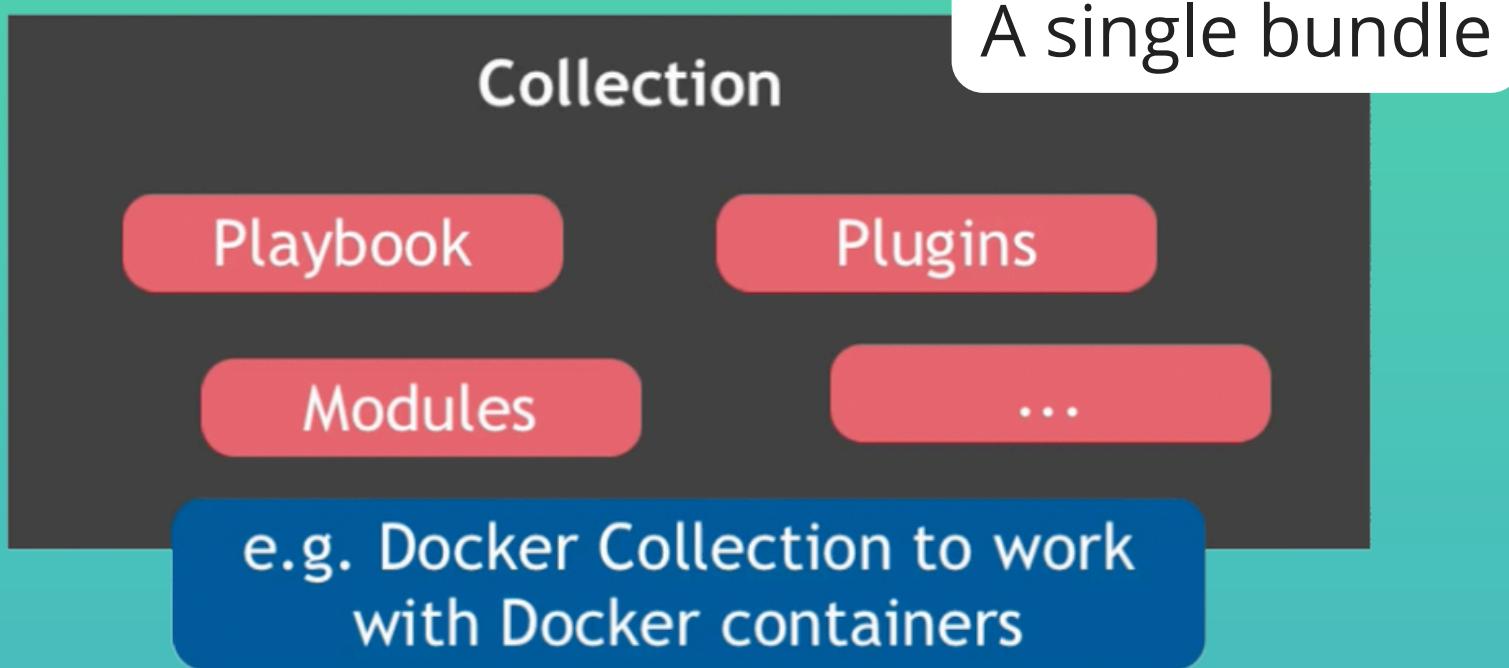


	Default: conflict, force confold"	
force boolean	Choices: <ul style="list-style-type: none">• no ←• yes	Corresponds to the <code>--force-yes</code> to <code>apt-get</code> . This option will disable checking both the <code>conflict</code> and <code>confold</code> dependencies. This option *is not* the equivalent of passing <code>--force</code> . **This is a destructive operation with the potential to corrupt your system. Use with care and read the documentation for more information.
force_apt_get boolean added in 2.4	Choices: <ul style="list-style-type: none">• no ←• yes	Force usage of <code>apt-get</code> instead of <code>aptitude</code> .
install_recommends boolean	Choices: <ul style="list-style-type: none">• no• yes	Corresponds to the <code>--no-install-recommends</code> option. By default, Ansible will use the same default behavior as <code>apt</code> . Aliases: <code>install-recommends</code>
name -		A list of package names, like <code>foo</code> , or package names with version numbers, like <code>foo=1.0*</code> are also supported. Aliases: <code>package</code> , <code>pkg</code>
only_upgrade boolean	Choices: <ul style="list-style-type: none">• no ←• yes	Only upgrade a package if it is already installed.
policy_rc_d integer added in 2.8	Default: null	Force the exit code of <code>/usr/sbin/policy-rc.d</code> . For example, if <code>policy_rc_d=101</code> the installed package will exit with code 101. If <code>/usr/sbin/policy-rc.d</code> already exist, it is overwritten. If <code>null</code> , the <code>/usr/sbin/policy-rc.d</code> isn't created.

Ansible Collections - 1

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

The screenshot shows the 'Collection Index' page from the Ansible Documentation. The left sidebar contains links for 'Documentation', 'ANSIBLE GALAXY', and 'REFERENCE & APPENDICES'. The main content area displays a message about reading the latest community version, followed by a list of collections: amazon.aws, ansible.builtin, ansible.netcommon, ansible.posix, ansible.utils, ansible.windows, arista.eos, awx.awx, azure.azcollection, check_point.mgmt, chocolatey.chocolatey, cisco.aci, and cisco.asa.



- 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

Ansible Collections - 2

Built-In Collections

- Many of the collections and modules are built-in



Maintenance

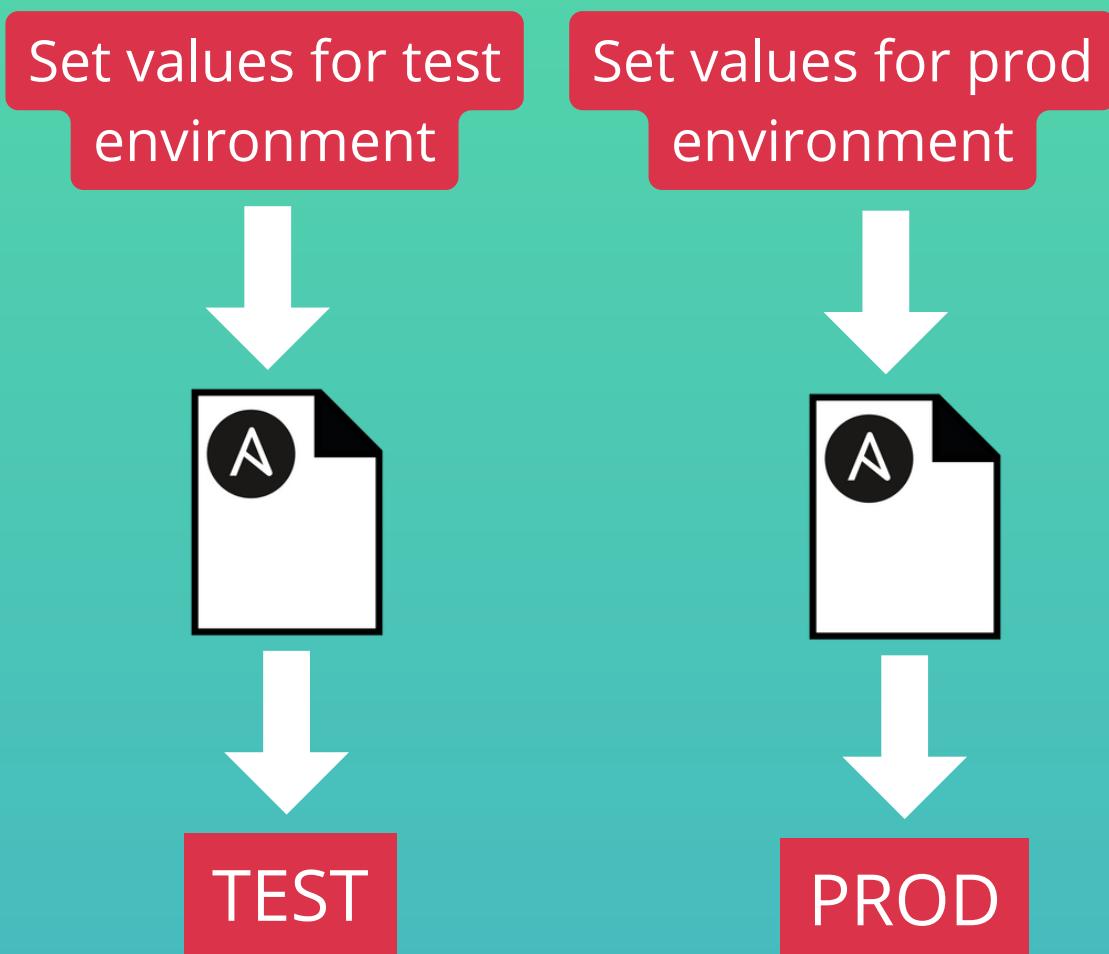
Collection	Code location	Maintained by
ansible.builtin	ansible/ansible repo on GitHub	core team
distributed on Galaxy	various; follow repo link	community or partners
distributed on Automation Hub	various; follow repo link	content team or partners

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 Terraform 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 Ansible 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

A screenshot of a code editor showing an Ansible playbook. The code includes a task to create a user and a task to deploy a Node.js application. The user creation task uses the 'register' keyword to store the result in a variable, which is then debugged. The resulting output is shown in the terminal tab of the code editor.

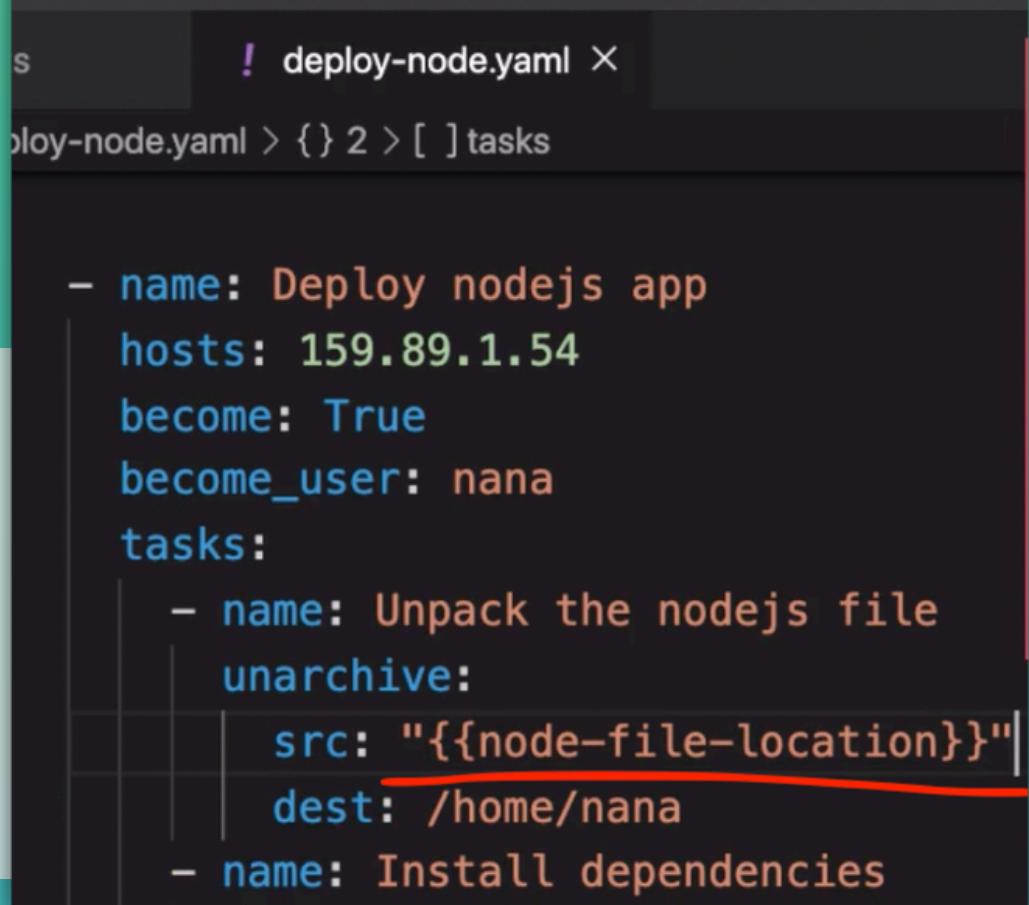
```
21   group: admin
22   register: user_creation_result
23   - debug: msg={{user_creation_result}}
24
25   - name: Deploy nodejs app
26     hosts: 159.89.1.54
27     become: True
```

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
 }
}

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



```
! deploy-node.yaml X
deploy-node.yaml > {} 2 > [ ] tasks
-
  - name: Deploy nodejs app
    hosts: 159.89.1.54
    become: True
    become_user: nana
    tasks:
      - name: Unpack the nodejs file
        unarchive:
          src: "{{node-file-location}}"
          dest: /home/nana
      - name: Install dependencies
```

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/nodejs-app
  - 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

- 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
```

Best way!!

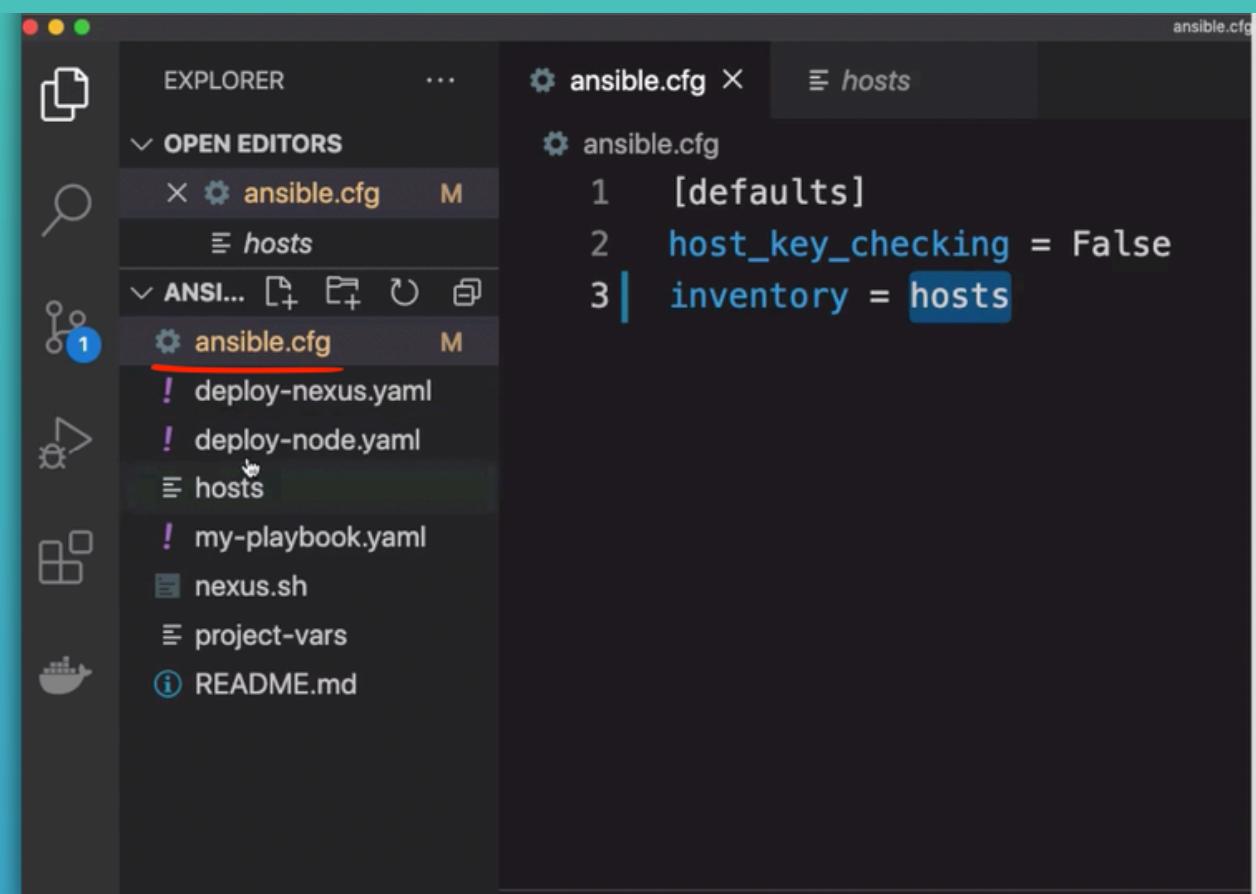
2) Setting values on Command Line

Options 1) and 2) are both impractical!

```
nana@macbook ~ % ansible-playbook -i hosts deploy-node.yaml -e "version=1.0.0 location=/Users/nana/Demo-projects/Bootcamp/nodejs-app"
```

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



► Have an inventory file for each project

GitLab

GitLab

ansible.cfg

Project A

ansible.cfg

Project B

The image shows a screenshot of the Visual Studio Code (VS Code) interface. On the left is the Explorer sidebar with a tree view of files and folders. In the center is the Editor pane displaying an 'ansible.cfg' file. The file's content is as follows:

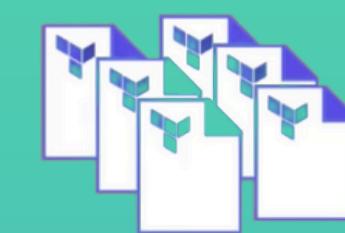
```
[defaults]
host_key_checking = False
inventory = hosts
```

To the right of the code editor, there is a diagram illustrating the concept of having an inventory file for each project. It features two 'GitLab' logos at the top. Below each logo is a file icon labeled 'ansible.cfg'. Underneath 'Project A', there are two 'ansible.cfg' files, one of which has a 'A' icon above it. Underneath 'Project B', there are three 'ansible.cfg' files, with the middle one having a 'A' icon above it.

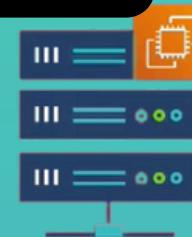
Terraform and Ansible - 1

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

Create EC2 Instance



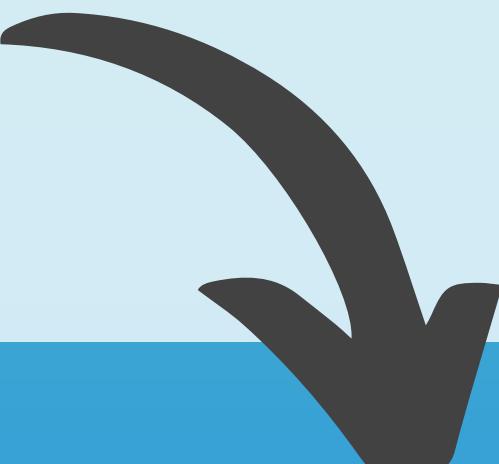
Configure EC2 Instance



Integrate Ansible Playbook Execution in Terraform

Now we need to integrate the Ansible execution 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 deploy-key.yml"
133   }
134 }
```

Ansible Side

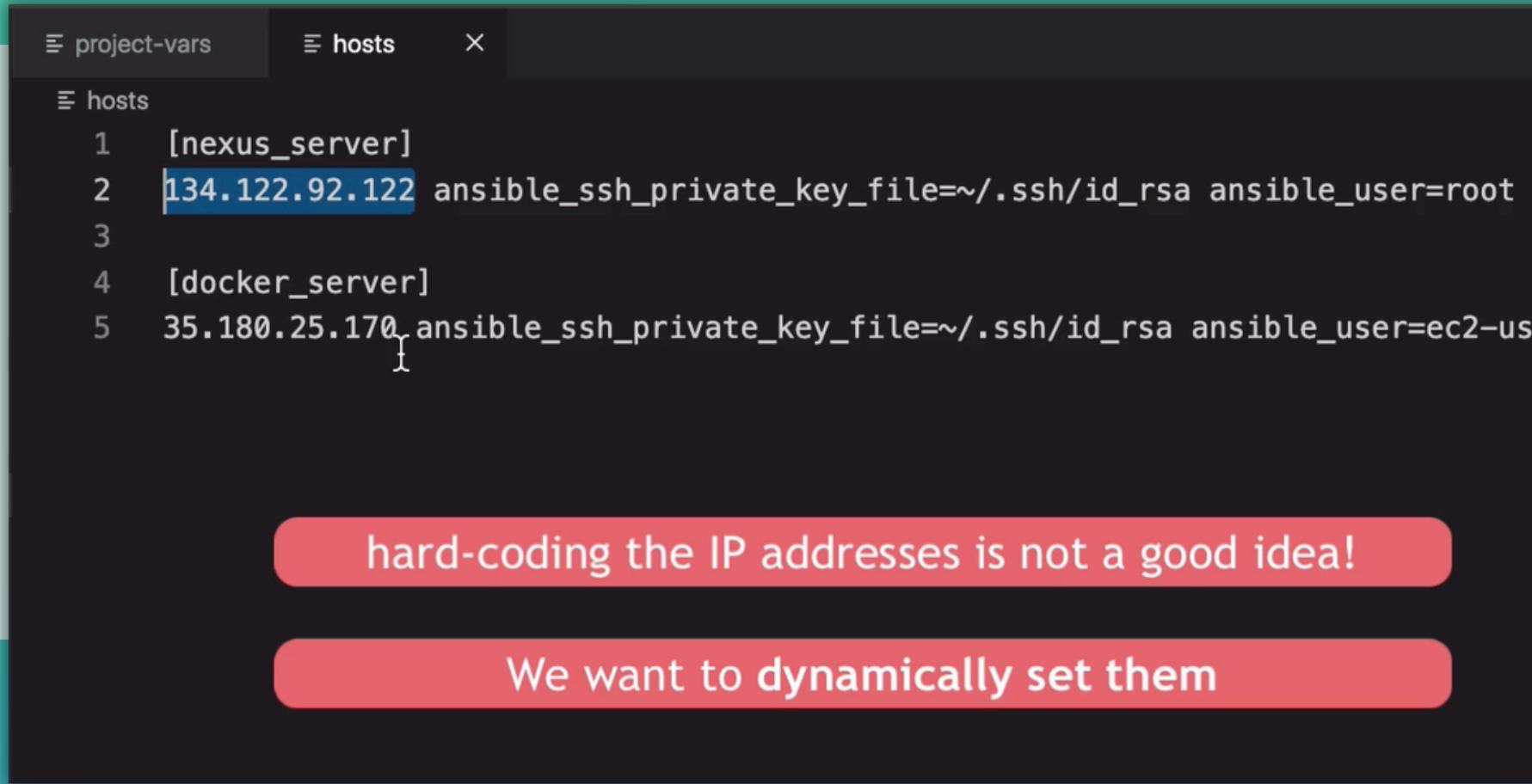
- Ansible will match "all" hosts and also first check and wait until the server is fully initialized

```
! deploy-docker-new-user.yaml ×
! deploy-docker-new-user.yaml > {} 0 > [ ] tasks > {} 0 > {} vars > ansible_connection
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 |      timeout: 100
11 |      search_regex: OpenSSH
12 |      host: '{{ (ansible_ssh_host|default(ansible_host))|default(inv
13 |
14 | vars:
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 accommodate 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-user
```

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

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"

```
! ansible.cfg ! inventory_aws_ec2.yaml
! inventory_aws_ec2.yaml > [ ] regions >
1 --- &lt;br&gt;
2 plugin: aws_ec2 &lt;br&gt;
3 regions: &lt;br&gt;
4 - eu-west-3
```

Prerequisites

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

```
! ansible.cfg &lt;br&gt;
! inventory_aws_ec2.yaml &lt;br&gt;
1 [defaults] &lt;br&gt;
2 host_key_checking = False &lt;br&gt;
3 inventory = hosts &lt;br&gt;
4 &lt;br&gt;
5 enable_plugins = aws_ec2
```

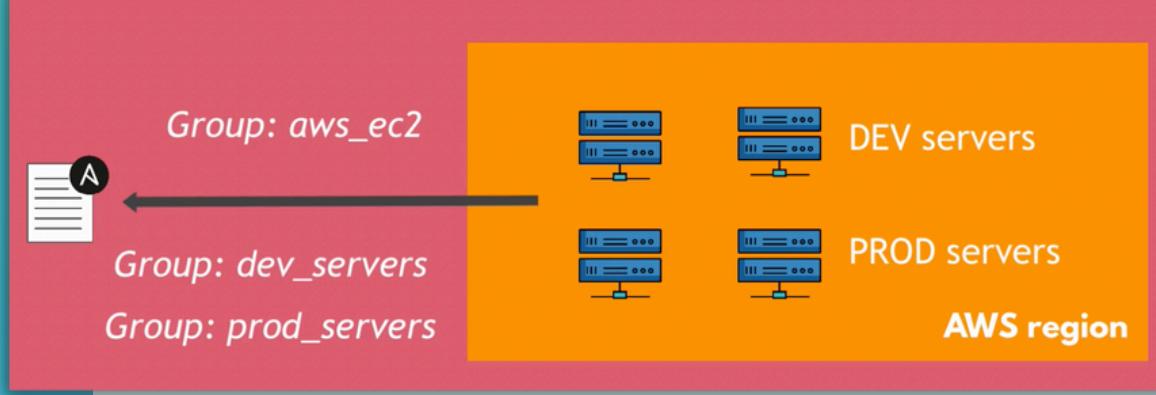
Dynamic Inventory for EC2 servers - 2

- To check result of plugin configuration:

```
nana@macbook /Users/nana
% ansible-inventory -i inventory_aws_ec2.yaml --list
```

Dynamic Groups

- Get servers grouped by ...



Custom Tags

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

```
! inventory_aws_ec2.yaml > {} filters > instance-state-name
1 ---
2 plugin: aws_ec2
3 regions:
4   - eu-west-3
5 filters:
6   tag:Name: dev*
7     instance-state-name: running
```

This screenshot shows a code editor interface with several files open. On the right, the "inventory_aws_ec2.yaml" file is displayed, showing configuration for the "aws_ec2" plugin with regions set to "eu-west-3" and a filter for "tag:Name: dev*" and "instance-state-name: running". A red box highlights the "prefix: "tag"" line. Below the editor, the terminal window shows the output of the command "ansible-inventory -i inventory_aws_ec2.yaml --list", listing servers under groups like "@aws_ec2", "@tag_Name_dev_server", and "@tag_Name_prod_server". Other tabs in the code editor include "ansible.cfg", "deploy-docker-new-user.yaml", and "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", and "TERMINAL".

```
! inventory_aws_ec2.yaml > [ ] keyed_groups > {} 0 > abc prefix
1 plugin: aws_ec2
2 regions:
3   - eu-west-3
4 keyed_groups:
5   - key: tags
6     prefix: "tag"
7
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
| |--ec2-35-180-137-149.eu-west-3.compute.amazonaws
|--@ungrouped:
[ \W (master)]$ ansible-inventory -i inventory_aws_ec2.yaml --list
@all:
--@aws_ec2:
|--ec2-15-237-106-205.eu-west-3.compute.amazonaws
|--ec2-15-237-124-203.eu-west-3.compute.amazonaws
|--ec2-15-237-24-155.eu-west-3.compute.amazonaws
|--ec2-35-180-137-149.eu-west-3.compute.amazonaws
--@tag_Name_dev_server:
|--ec2-15-237-124-203.eu-west-3.compute.amazonaws
|--ec2-15-237-24-155.eu-west-3.compute.amazonaws
--@tag_Name_prod_server:
|--ec2-15-237-106-205.eu-west-3.compute.amazonaws
|--ec2-35-180-137-149.eu-west-3.compute.amazonaws
@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

```
- 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

*tasks/main.yml vars/main.yml
files/my-file.yml defaults/main.yml
library/my_module.py*

Role



Ansible Roles - 2

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

Use existing roles

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



my_role



role_A role_LL
role_CC role_XY
.... role_RX

Standard file structure

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

Like functions

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

```
ANSIBLE
  roles
    create_user
      defaults
        main.yaml
      tasks
        main.yaml
    start_containers
      defaults
        main.yaml
      files
        docker-compose.yaml
      tasks
        main.yaml
      vars
        main.yaml
```

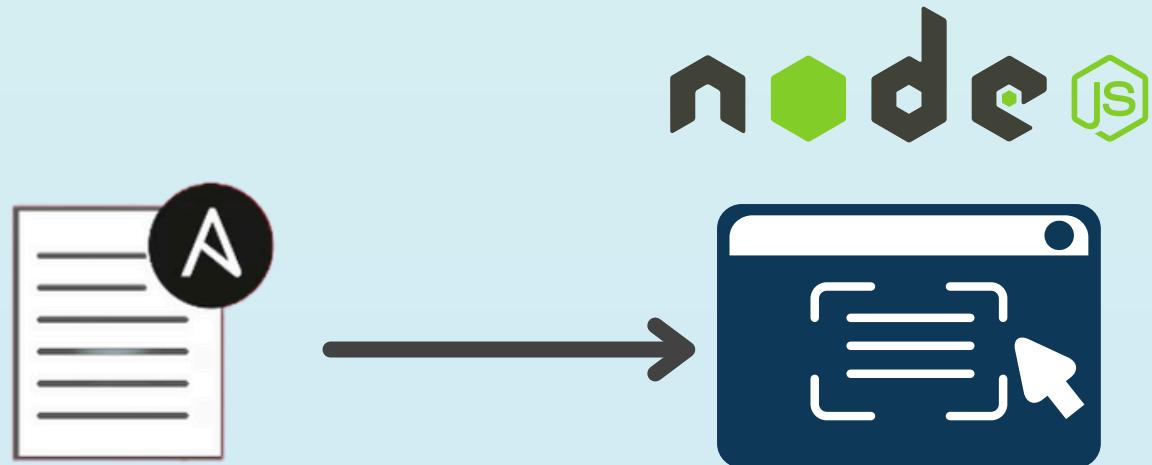
```
# playbooks
site.yml
webservers.yml
fooservers.yml
roles/
  common/
    tasks/
    handlers/
    library/
    files/
    templates/
    vars/
    defaults/
    meta/
  webservers/
    tasks/
    defaults/
    meta/
```

```
1
2   - name: Create new user
3     hosts: all
4     become: yes
5     vars:
6       user_groups: adm
7     roles:
8       - create_user
9
```

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



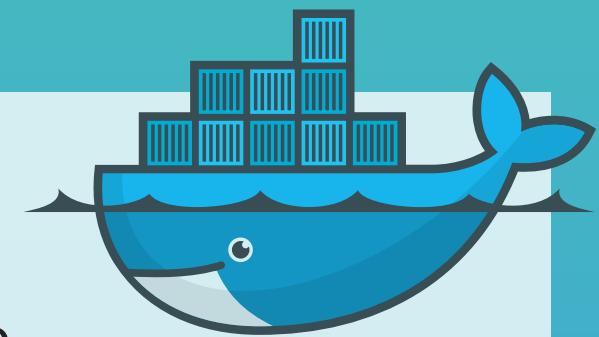
Automate Nexus Deployment



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

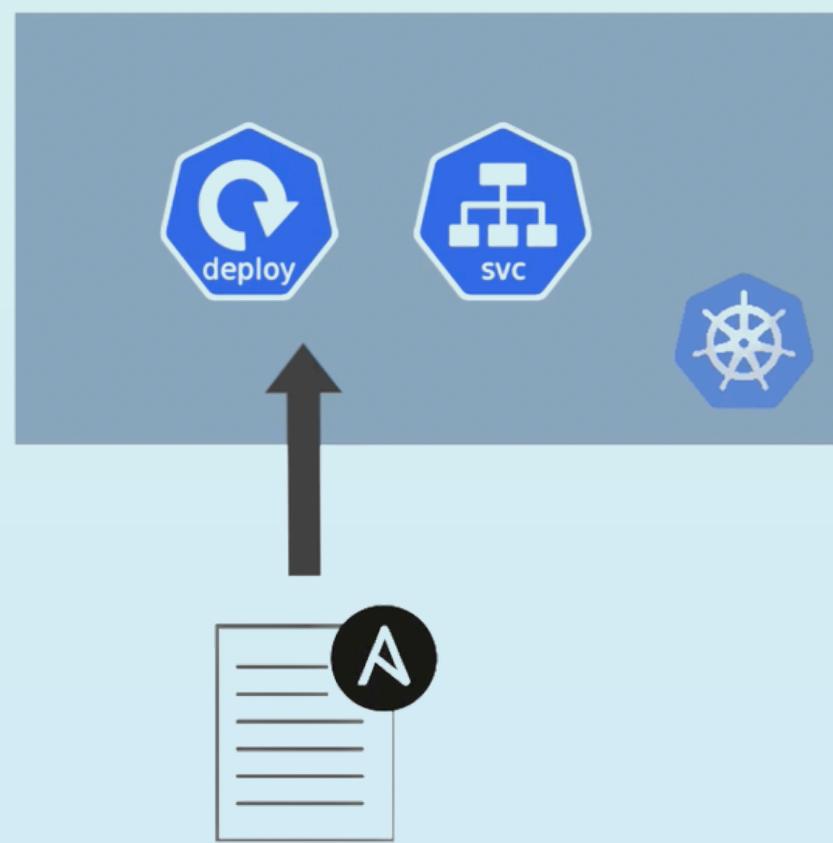


Demo Projects - 2

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

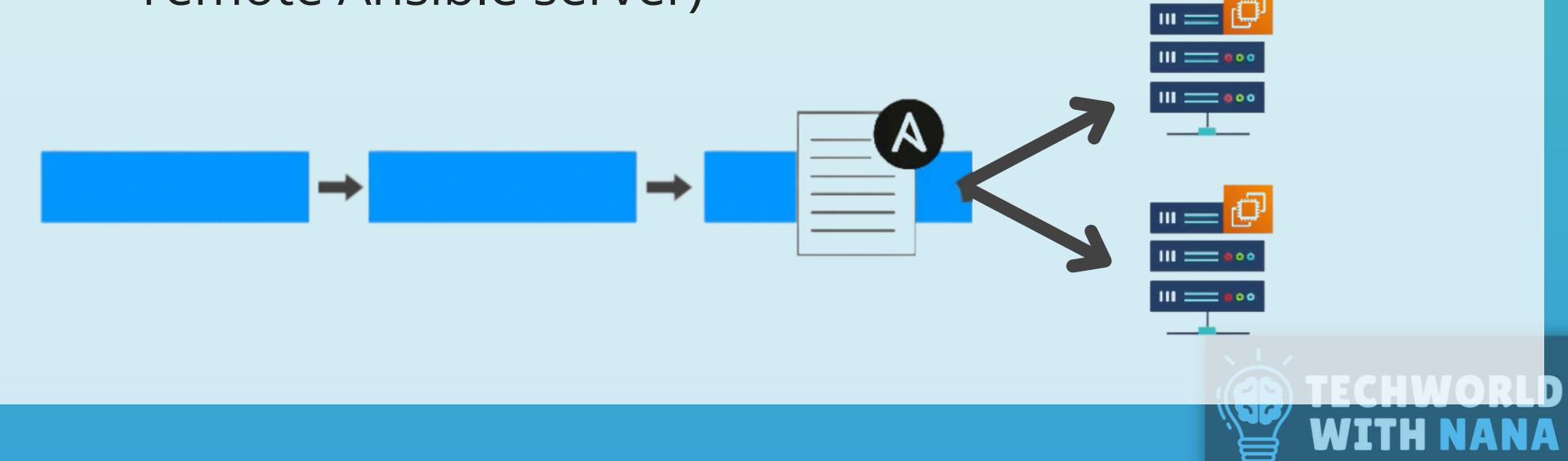


1. Create a DO server for Jenkins
2. Create dedicated server for Ansible
3. Install Ansible on that server

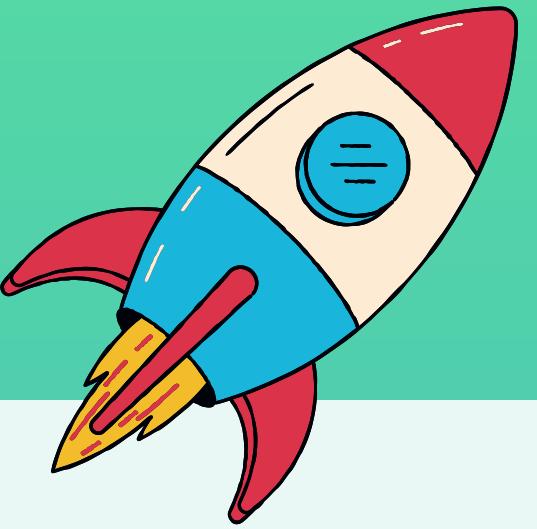
Run Ansible locally on Jenkins

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

Tip

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