Managing Server Configuration

What is Ansible?

- End to end automation tool for
  - Provisioning cloud infrastructure
  - Configuration management
  - Application deployment
- Overlaps with Terraform for infrastructure provisioning
  - Preference for Terraform over Ansible for provisioning and managing infrastructure
  - Ansible is used to configure servers after they are provisioned
- Build on Python
- Agentless, uses SSH or WinRM to connect to targets

IaC Tools

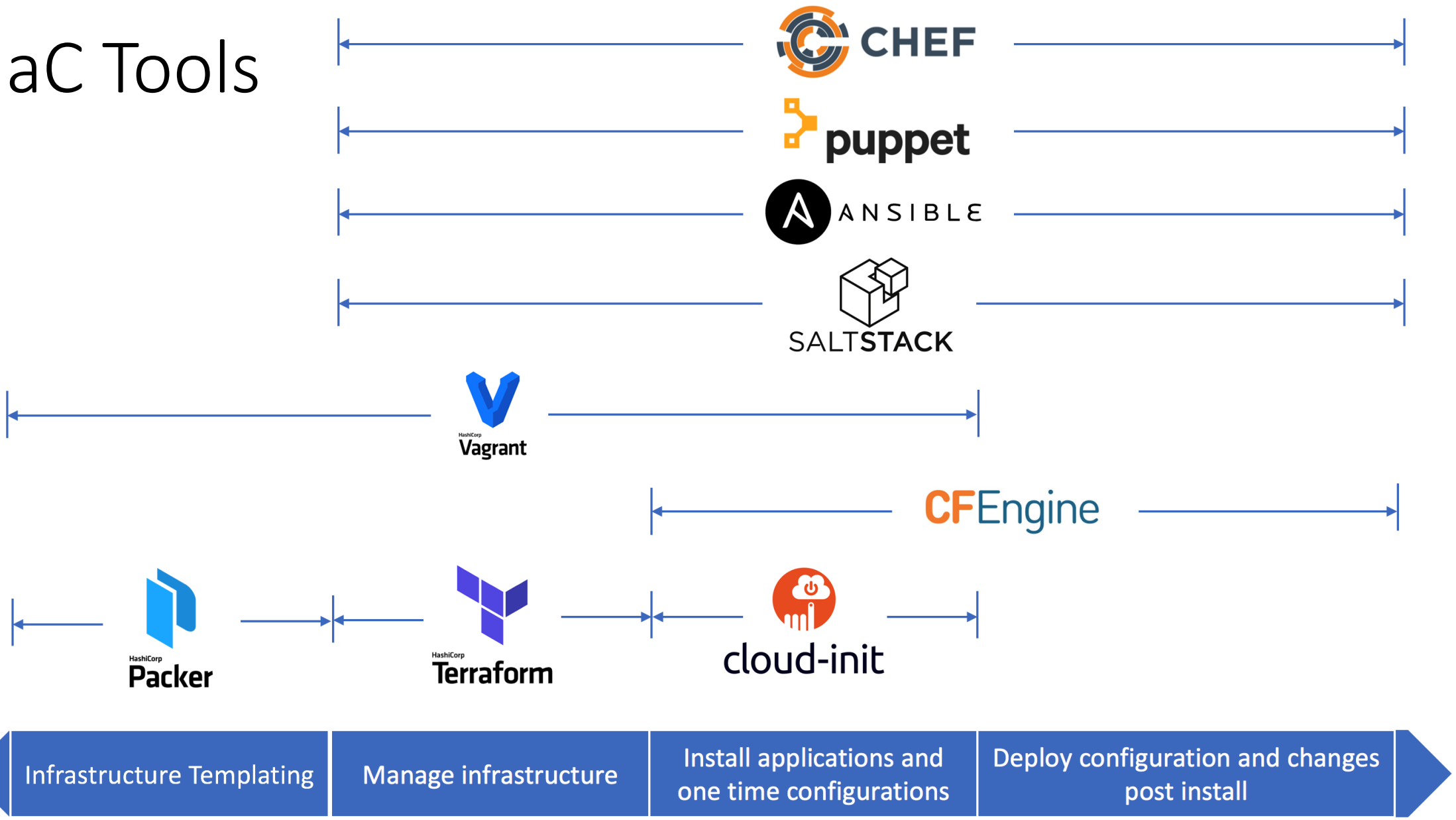


Image from https://medium.com/cloudnativeinfra/when-to-use-which-infrastructure-as-code-tool-665af289fbde

Why is Ansible?

Generic OS

Install software
Copy content
Copy configuration
Setup security rules
Start web service

Web Server

- Prepare server by performing software installation, configuration, etc on the server
  - Mutate a server from generic to a specific use eg. web server, database
- Ansible automates the software installation and configuration
- Uses modules to perform sysadmin task
  - Some modules are idempotent - eg. starting a service, installing a package
  - Use conditions to guard task that are not idempotent eg. deleting a file

Ansible Architecture

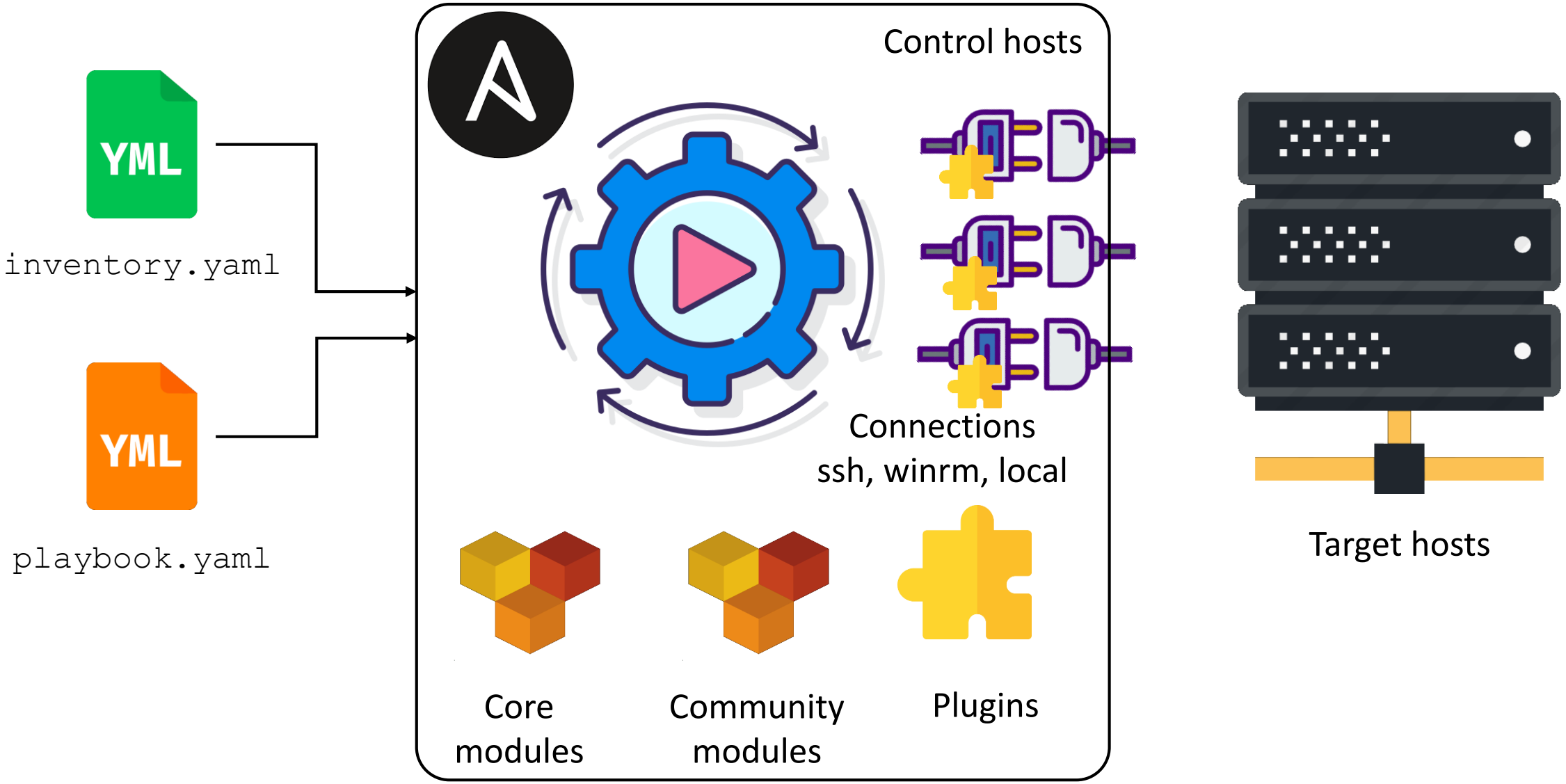
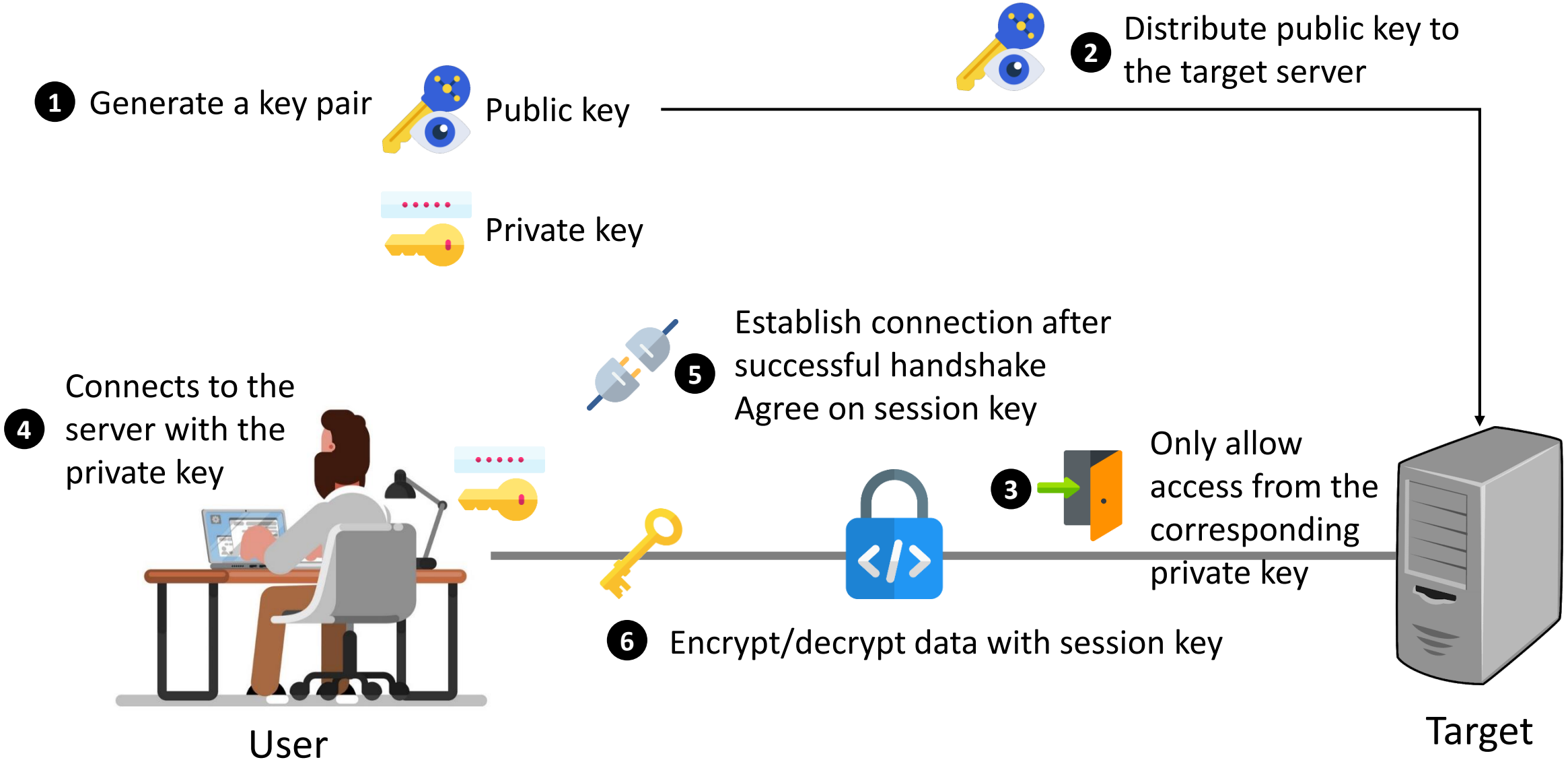


Image adapted from https://geekflare.com/wp-content/uploads/2019/06/ansible-architecture-1.png

Control and Target Host

- Control host is the host that runs the you run Ansible
  - Ansible core software, modules are installed
  - Require Python 3.x, SSH
  - Windows control host is currently not supported
  - Workaround is to use WSL, not supported in production
- Target host can be Linux, Windows or OSX
  - Must have SSH or WinRM installed
- Control host uses SSH to connect to targets
  - Perform operations over the connection
  - Can use password instead of keys but discouraged
- Ansible is agentless
  - Requires no special software to be installed on targets besides the 2 mentioned above

How Does SSH Work?



Managing Public/Private Keys with SSH

- Generate a public/private key for SSH

```
ssh-keygen -t rsa -b 4096 -C “fred@gmail.com” -f fred_rsa4096
```

- Distribute public key to target server

```
ssh-copy-id -i ./fred_rsa4096.pub fred@server.com
```

- Trust public keys from servers

```
ssh-keyscan -H server.com >> ~/.ssh/known_hosts
```

- Connecting to target server

```
ssh -i ./fred_rsa4096 fred@server.com
```

- Remove keys associated with a host

```
ssh-keygen -R server.com
```

Terms

- Inventory
  - List of IP address to be managed
  - Inventory list can be static or dynamic
- Playbooks
  - Apply plays to the IP addresses in the inventory
  - A play is a sequence of task to be performed on a server
- Role
  - Describes the purpose of a server eg. a database server
  - Uses plays to transform a generic server to a specific server

Inventory

- Inventory is a list of host to be managed by Ansible
- Location is typically in your Ansible's project directory
  - Default location is at `/etc/ansible/hosts`
- File format can be in YAML or INI format
- Inventory file contains
  - IP address, hostname, FQDN or alias
  - Groupings of the above
  - Variables associated with the IP addresses, hostname, FQDN or aliases
    - Eg. connection parameters, database name, etc

Example - Inventory File

Special `all` group

List of all the hosts

Server name/alias

```
all:
  hosts:
    server-0:
      ansible_host: 192.168.0.100
      ansible_connection: ssh
      ansible_user: fred
      ansible_password: fred
    server-1:
      ansible_host: 192.168.0.101
      ansible_connection: ssh
      ansible_user: fred
      ansible_password: fred
    server-2:
      ansible_host: 192.168.0.102
      ansible_connection: ssh
      ansible_user: fred
      ansible_password: fred
      db_user: barney
      db_password: barney
      db_name: inventory
```

Variables associated with the server
Used by Ansible and playbooks
Variable names that starts with `ansible_` are special variable

Example - Inventory File

Host variables; group common variables across all host under `vars`

```
all:
  vars:
    ansible_user: fred
    ansible_password: fred
    ansible_connection: ssh
    ansible_python_interpreter: /usr/bin/python3

  hosts:
    server-0:
      ansible_host: 192.168.0.100
    server-1:
      ansible_host: 192.168.0.101
    server-2:
      ansible_host: 192.168.0.102
      db_user: barney
      db_password: barney
      db_name: inventory
```

Variables specific to the host will be merged with the common variables

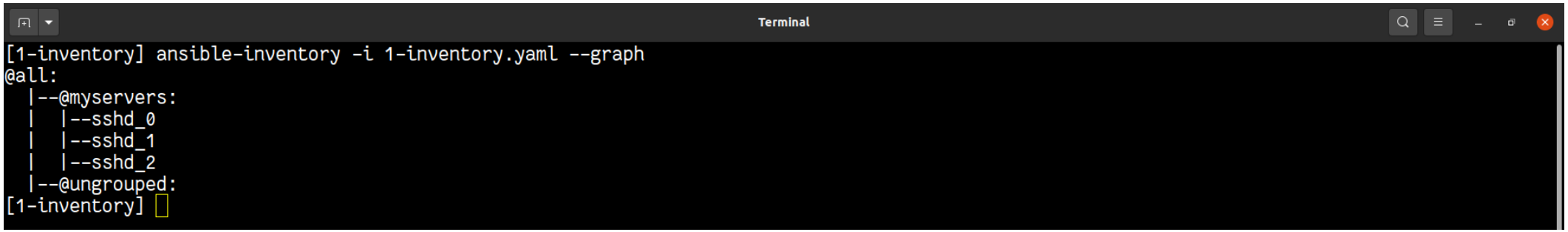
List Inventory File

- List the contents as JSON

```
ansible-inventory -i inventory.yaml --list
```

- Display the inventory as a graph

```
ansible-inventory -i inventory.yaml --graph
ansible-inventory -i inventory.yaml --graph --vars
```



List Inventory File

```
[1-inventory] ansible-inventory -i 1-inventory.yaml --graph --vars
@all:
  |--@myservers:
  |  |--sshd_0
  |  |  |--{ansible_host = 172.17.0.2}
  |  |  |--{ansible_password = fred}
  |  |  |--{ansible_python_interpreter = /usr/bin/python3}
  |  |  |--{ansible_user = fred}
  |  |  |--{db_name = inventory}
  |  |  |--{db_password = fred}
  |  |  |--{db_user = fred}
  |  |--sshd_1
  |  |  |--{ansible_host = 172.17.0.3}
  |  |  |--{ansible_password = fred}
  |  |  |--{ansible_python_interpreter = /usr/bin/python3}
  |  |  |--{ansible_user = fred}
  |  |--sshd_2
  |  |  |--{ansible_host = 172.17.0.4}
  |  |  |--{ansible_password = fred}
  |  |  |--{ansible_python_interpreter = /usr/bin/python3}
  |  |  |--{ansible_user = fred}
  |--@ungrouped:
  |--{ansible_password = fred}
  |--{ansible_python_interpreter = /usr/bin/python3}
  |--{ansible_user = fred}
[1-inventory]
```

Example - Inventory File

```
all:
  vars:
    ...
  hosts:
    server-0:
      ...
    server-1:
      ...
    server-2:
      ...
  children:
    apps:
      hosts:
        server-0:
        server-1:
    database:
      hosts:
        server-2:
```

children attribute groups the hosts into specific groups

Two groups: apps and database

```
all:
  vars:
    ...
  hosts:
    server-0:
      ...
    server-1:
      ...
    server-2:
      ...
  children:
    apps:
      vars:
        ...
      hosts:
        server-0:
        server-1:
    database:
      hosts:
        server-2:
```

Variables common to all the hosts in the group

Variables in the group overrides the hosts

Connection Variables

- `ansible_connection` - how to connect to the target
  - Valid values are `ssh`, `winrm`, `local`
- `ansible_host` - IP, host name or FQDN of the target
- `ansible_user` - login user's name
- `ansible_password` - login password
  - Should not be using this
- `ansible_port` - change the SSH port, defaults to 22
- `ansible_ssh_private_key_file` - path to the private key file for passwordless SSH connection

Validate Connections

- Customary to run a ping to validate the connection configurations in the inventory file

All the targets from `inventory.yaml`
`all` is a special/reserved group name

Use the ping module

```
ansible all -i inventory.yaml -m ping
```

```
ansible apps -i inventory.yaml -m ping
```

Only ping the targets from `apps` group

Gathering Facts

- Gather information about remote host
  - Eg. OS version, packages install, network interfaces, etc
- Use the information to populate `hostvar` map
- Many modules require these information to work correctly
- This is performed automatically whenever we run a playbook

```
ansible all -i inventory.yaml -m setup
```

Playbooks

- Playbook consist of an ordered list of tasks
- These tasks can be logically group into plays
  - Eg. Setup database, configure iptable
- Playbooks applies operations (plays) to the hosts/groups in an inventory
- Plays are applied to hosts or groups

Example - Playbook

playbook.yaml

```
- name: Connectivity test
  hosts: apps
  tasks:
  - name: Ping all the target servers
    ping:
```

List of host that this play targets

One or more plays in a playbook

One or more task(s) in a play

Module used in this task

Inventory used by this playbook

```
ansible-playbook playbook.yaml -i inventory.yaml
```

Example - Playbook

Plays order of execution

```
- name: Sanity check
  hosts: all
  tasks:
  - name: Check connectivity
    ping:
  - name: Echo variables
    debug:
      msg: ansible_user {{ ansible_user }}
- name: setup server
  hosts: apps
  tasks:
  - name: Install packages
    ...
```

Dynamic expression evaluated by Jinja2 template

Tasks are executed sequentially
Ansible will wait for a task to complete before executing the next task

Task

- Task is a single unit of action
- A play contains one or more tasks
- Task are performed sequentially and in the order in which they are listed
  - Ordering is important eg cannot create a MySQL database if MySQL packages are not installed
- Common attributes in tasks
  - `name` - description of the task
  - module name - this is where the 'action' is
    - Eg. `apt`, `user`, `iptable`
  - `loop` - execute the task with the given list of items

- Common attributes in tasks
  - `register` - captures the output from the module
    - Eg. check if a file exists
  - `when` - executes a condition, if it is false do not execute the task
    - Eg. if a file exists and is readeable
  - `become` - escalate privilege to root when executing this task
  - `environment` - sets an environment variable
  - `until` - repeat a task until a certain condition is met
- See https://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html#task

Example - Install MongoDB Playbook

```
- name: Install MongoDB
  hosts: apps
  tasks:
  - name: Import GPG key for the repo
    apt_key:
      url: https://www.mongodb.org/static/pgp/server-5.0.asc
     state: present
  - name: Add MongoDB source
    apt_repository:
      repo: “deb [arch=amd64,arm64] https://repo.mongodb.org/apt/ubuntu
focal/mongodb-org/5.0 multiverse”
  - name: Install MongoDB
    apt:
      name: mongodb-org
      update_cache: yes
      state: latest
```

User and Group Management

```yaml
group:
  name: mysql
  state: present
```

```
groupadd mysql
```

```yaml
group:
  name: mysql
  state: absent
```

```
groupdel mysql
```

```yaml
user:
  name: fred
  groups: mysql,sudo
  append: yes
  shell: /bin/bash
  state: present
```

```
useradd -g mysql -G sudo \
  -s /bin/bash fred
```

Creating Files and Directories

```yaml
file:
  path: /opt/tmp
  state: directory
  owner: root
  group: users
  mode: '0755'
```

```
mkdir /opt/tmp
chmod 755 /opt/tmp
sudo chown root /opt/tmp
sudo chgrp users /opt/tmp
```

```yaml
file:
  src: /usr/local/bin
  dest: /home/fred/bin
  state: link
```

```
ln -s /usr/loca/bin /home/fred/bin
```

Finding Files

```
find:
 path: /home/fred/docs
```

```
find /home/fred/docs -print
```

```
find:
 path: /var/logs
 patterns: '*.log.gz,*.log'
 size: 10m
 age: 28d
```

```
find /var/logs \
  -type f
  -size +10m \
  -mtime +28 \
  \(-name '*.log.gz' -o -name '*.log' \) \
  -print
```

Install Packages with apt

```yaml
apt:
  update_cache: yes
  name: git
  state: present
```

```
sudo apt update
sudo apt install git
```

```yaml
apt:
  name: git
  state: absent
```

```
sudo apt update
sudo apt remove git
```

Service Management with systemd

```yaml
systemd:
  name: nginx
  enabled: yes
  state: started
```

```
sudo systemctl enable nginx
sudo systemctl start nginx
```

```yaml
systemd:
  daemon_reload: yes
  name: nginx
  state: restarted
```

```
sudo systemctl daemon-reload
sudo systemctl restart nginx
```

Downloading Remote File

```
get_url:
  url: https://server.com/myfile.zip
  dest: /opt/downloads
  checksum: 'md5:abc123'
```

```
curl https://server.com/myfile.zip -o /opt/downloads
md5sum /opt/downloads/myfile.zip
```

Unarchive

```yaml
unarchive:
  src: myapp.zip
  dest: /opt/src
```

```
scp -i id_rsa myapp.zip \
  root@server.com:/opt/src
ssh -i id_rsa root@server.com \
  'unzip /opt/src/myapp.zip'
```

```yaml
unarchive:
  src: https://server.com/myfile.zip
  dest: /home/fred/src
  remote_src: yes
```

```
ssh -i id_rsa root@server.com
curl http://server.com/myfile.zip -o /opt/src
unzip /opt/src/myfile.zip
```

Executing Arbitrary Command

```
shell:
  cmd: "ls -1"
  chdir: /etc
```

```
cd /etc
ls -1
```

```
shell:
  cmd: "ip addr > /tmp/ip.txt"
  creates: /tmp/ip.txt
```

```
test -f "/tmp/ip.txt" || \
  ip addr > /tmp/ip.txt
```

```
command:
  argv:
  - /usr/bin/mysqladmin
  - ping
```

```
/usr/bin/mysqladmin ping
```

Loop

Control variable

```
- name: Install Python environment
  tasks:
  - name: Install packages
    apt:
      name: {{ item }}
      state: present
    loop:
    - python3.8
    - python3-pip
    - build-essentials
    - libssl-dev
    - libffi-dev
    - python3-dev
```

Execute this module with items from the loop

List of values to be iterated

Loop

```
- name: Add user
  vars:
    users:
    - name: fred
      state: present
      groups: sudo,users
    - name: barney
      group: users
      state: present
    - name: riddler
      groups: ''
      state: absent

  tasks:
    ...
```

```
tasks:
- name: Create users
  user:
    name: {{ item.name }}
    groups: {{ item.groups }}
    state: {{ item.state }}
  loop: '{{ users }}'

- name: Force users to change password
  command:
    cmd: chage -d 0 '{{ item.name }}'
  loop: '{{ users }}'
```

Dynamic expression must be quoted

Dynamic Expression

```
{{ expression | filter | filter | ... }}
```

- Expression (1st term) produces a value; can be
  - a variable
  - function call - eg. lookup an environment variable, flattening a list
- Values can then the modified by filters
  - Eg. converting values to upper case
  - Eg. providing a default value if expression evaluates to a ‘null’ value
- Expressions are chained by the |
  - Values flow from left to right
- Dynamic expressions are evaluated before they are executed by Ansible on the target machine

Example - Dynamic Expression

- Converting a string value to upper case

```
{{ ansible_distribution | upper }}
```

- Default value

```
{{ name | default('not set') }}
```

- Minimum value from a list

```
{{ a_list | min }}
```

- Test if a string is a valid IP address

```
{{ server.ip_addr | ipaddr }}
```

- Lookup an environment variable

```
{{ lookup('env', 'DO_TOKEN') }}
```

- Query JSON with modified JSON path (omit `$.` prefix)
  - From https://openweathermap.org/current

```
{{ result | json_query('main.temp') }}
```

- Remove :port from the host

```
{{ host | regex_replace(':\\d+$') }}
```

- https://docs.ansible.com/ansible/2.8/user_guide/playbooks_filters.html

Example - Dynamic Expression

```
- name: Setup
  vars:
    pkgs:
      mysql_server: latest
      nginx: latest
      libreoffice: absent

- name: Install packages
  apt:
    name: {{ item.key }}
    state: {{ item.value }}
  loop: '{{ pkgs | dict2items }}'
```

Converts to this format

```
  - key: mysql_server
    value: latest
  - key: nginx
    value: latest
```

```
- name: Setup
  vars:
    dbs:
    - name: customer
      target: /tmp/custdb.sql.bz2
    - name: inventory
      target: /tmp/invdb.sql.bz2
      force: yes
- name: Restore database
  mysql_db:
    name: {{ item.name }}
    target: {{ item.target }}
    state: import
    force: {{ item.force | default(omit) }}
  loop: '{{ dbs }}'
```

Omit the 'force' parameter if `item.force` is not set. omit is a special value to enable this behaviour

Capturing Result from a Task

- Ansible task produces output
  - Eg. `command` module produces output in `stdout`, error in `stderr`
  - Eg. `mysql_query` module returns the SQL query result
- These results can be used by the next task
  - Eg only execute the task if a particular file exists
- `register` attribute captures the result from a task
- Note what is captured with register is highly specific to the task
  - See Return Values for the module used in the task

Example - register

Run this task on the localhost (127.0.0.1) rather than from the inventory

```
tasks:
- name: Get users from database
  mysql_query:
    login_user: {{ db_user }}
    login_password: {{ db_password }}
    login_db: {{ db_name }}
    query: select username from user
  delegate_to: 127.0.0.1
  register: users
- name: Add users to the
  user:
    name: {{ item.username }}
    group: sudo
    state: present
  loop: "{{ users[0] }}"
```

Query result is stored in a 'variable' called `users`
The structure of the result is a list of list

Use the result from the query to create users in the current host

Conditionals

- Only execute the task when certain condition is true

```
tasks:
- name: Check if file exists
  stat:
   path: /tmp/runme.sh
  register: file
- name: Run file if exists
  command:
    cmd: /tmp/runme.sh
  when: file.stat.exists and file.stat.executable
```

Only execute this task if runme.sh exists and is executable
No {{ }} required because the string to the right is considered a dynamic expression

Conditional with Loop

- When a loop is present in a task with a condition, the loop will be unrolled with
- The condition will be tested against every item from the loop

```
tasks:
- name: Find all files
  find:
    paths: /etc
    file_type: file
  register: result
- name: Record all public readable
  lineinfile:
    path: /var/log/readeables.log
    create: true
    state: present
    line: "{{ item.path }}"
  when: item.roth and item.rgrp
  loop: "{{ result.files }}"
```

Every item in the loop is evaluated by the condition
So when must evaluate the loop's control variable item

Until Loop

- Use to retry an task until a condition is met
  - Eg. Poll a remote resource until it is ready

Use the URI module to invoke an endpoint

Wait until the endpoint returns a 200 before continuing

```
tasks:
- name: Check if application is ready
  uri:
    url: http://acme.com/api/ready
  register: response
  until: response.status == 200
  retries: 10
  delay: 5
  failed_when: response.status >= 500
```

Number of times to retry and the wait duration between retires
Default retry is 3

Define a fail condition

Templates

- Use template to generate text files
  - Eg. HTML, configuration files, scripts, etc.
- Ansible variables are available to the script
  - `hostvars` - map of all defined variables for a host
  - `groups` - map of all hosts for a group
  - host specific variables - eg. `ansible_hostname`, `mysql_user`, etc.
- Template files to be executed by the template module ends with `.j2` suffix
- See https://palletsprojects.com/p/jinja

Templates

- Dynamic expressions

```
{{ expression }}
```

- If directive

```
{% if expression %} {% else %} {% endif %}
```

- Loop directive

```
{% for x in collection %} {% endfor %}
```

- New lines
  - Add new line before or after a directive
  - Suppress new line before or after a directive

```
{%+ for x in collection -%}
  {{ x }}
{%- endfor +%}
```

Example - Template

```
nginx.conf.j2
http {
  upstream app_instances {
    {%+ for host in groups['apps'] %}
      server {{ hostvars[host].ansible_host }}:3000;
    {% endfor %}
  }

  limit_req_zone: $request_uri zone=MYZONE:10m
      rate={%- if (groups['apps']|length) <= 5) %}10{% else %}15{%- endif %}r/s;

  ...
}
```

For directive

`groups` is a special variable (map) containing all the groups from the inventory file

Group name, must exist in inventory file

`hostvar` is a special variable (map) containing all the variables defined for a host

Condition directive

Length filter

+/− controls new line insertion.
Can be appended to either `{%` or `%}`
No whitespace

Example - Template

```yaml
- name: Install and configure Nginx
  hosts: rev_proxy
  tasks:
  - name: Install Nginx
    apt:
      name: nginx
      state: latest
  - name: Generate config file for Nginx
    template:
      src: ./nginx.conf.j2
      dest: /etc/nginx/nginx.conf
    notify:
    - Reload Nginx
  handlers:
  - name: Reload Nginx
    service:
      name: nginx
      state: reload
```

Template module

Generate the template and copy over to the target

Event handlers triggered with a notification. Allow reusing task in a play

Handler's name must be unique in the playbook

Example - Template Output

```
nginx.conf
http {
  upstream app_instances {
    server 192.168.10.1:3000;
    server 192.168.10.2:3000;
    server 192.168.10.3:3000;
  }
  limit_req_zone: $request_uri zone=MYZONE:10m
    rate=10r/s;
}
```

One ‘server’ line for every host in the inventory `app` group

Rate limiting set according to the number of host in `app` group

Privilege Escalation

- Certain task requires root privileges
  - Eg. installing packages
  - Eg. listening on port 80
- Will not have root privileges if SSH user is not root
  - Assume root by sudo, su
  - User must be in the sudo group, for Ubuntu

```
inventory.yaml
all:
  vars:
    ansible_user: fred
    ansible_connection: ssh
    ansible_ssh_private_key_file: priv_key
```

Non root user

```
playbook.yaml
...
tasks:
- name: install packages
  become: true
  apt:
    name: "{{ item }}"
    state: latest
  loop: "{{ list_of_packages }}"
```

Use sudo to become root

```
ansible-playbook all -i inventory.yaml \
   playbook.yaml --ask-become-pass
```

Prompt for fred's password on the host so fred can become root

Protecting Secrets

- Ansible files are in plain text
- Sensitive information are exposed
  - Eg. password, API keys, etc
- Plain files should be encrypted with Ansible vault
  - Can encrypt any Ansible files with sensitive information
  - playbooks, inventory

```
[root@rhel-8 ~]#
[root@rhel-8 ~]# cat mysecrets.yml
$ANSIBLE_VAULT;1.1;AES256
30633536343539366535353961656365343934343936353765633036346239373830336334643861
35306162333366353966353036306161663334363962343 90a35303966353464336233353461366 1
6562646234336330343732333739386566363232623437313539343937363862393338303039643 3
6538633038396637330a313033363865343561313762313366303734346365613661313361663462
3561353639396635323335346336326430363136323762636466636361626164333737336165383 8
3539376339643135376533376562633865393235323733393837
[root@rhel-8 ~]#
```

Ansible Vault

- Encrypt a file, will be prompted with a password

```
ansible-vault encrypt inventory.yaml
```

- View a Vault encrypted file

```
ansible-vault view inventory.yaml
```

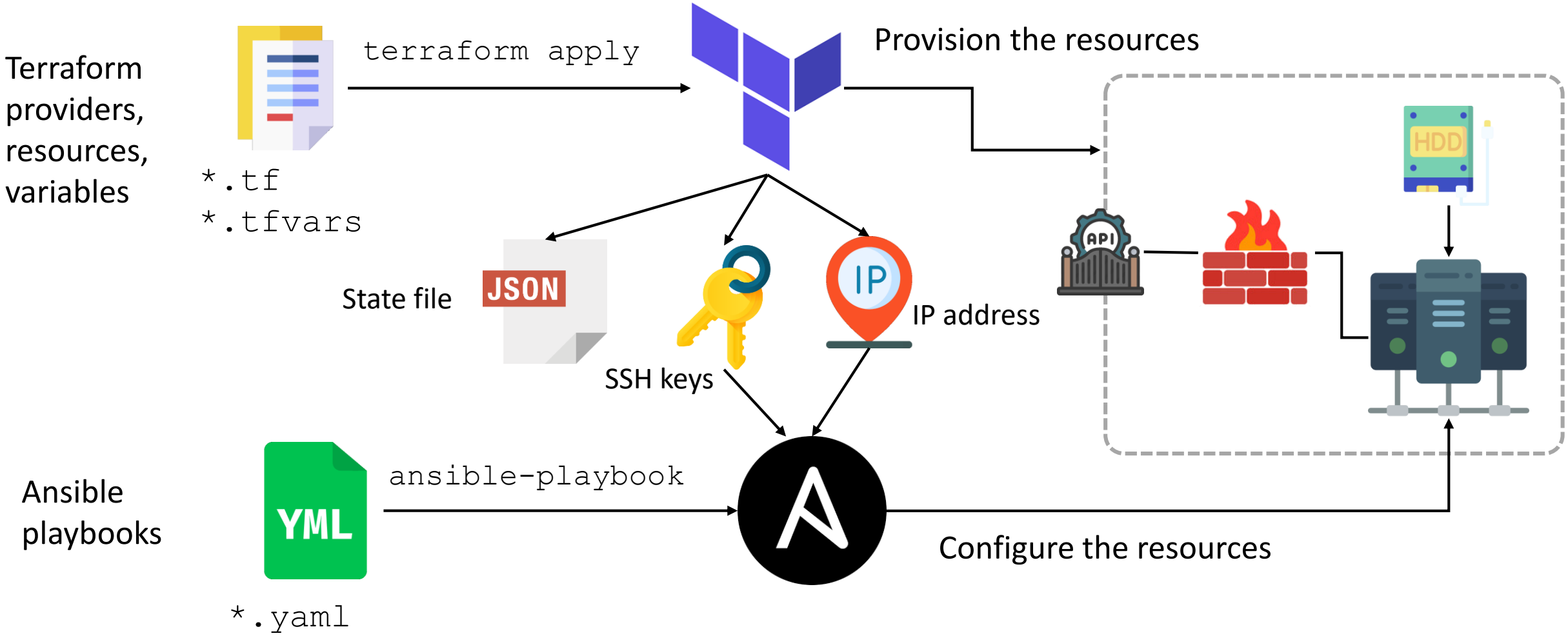
- Decrypt a Vault encrypted file

```
ansible-vault decrypt inventory.yaml
```

- Use a Vault encrypted file

```
ansible-playbook all -i inventory.yaml \
  playbook.yaml --ask-vault-pass
```

Using Terraform and Ansible Together



Picture adapted from https://ibm.github.io/cloud-enterprise-examples/static/ced342e4102a26420170f80f60846589/1ec2b/IaC-Ansible_Design_1.png

Appendix

Example - Generate Ansible Inventory from Terraform

```
resource digitalocean_ssh_key mykey {
  name = “mykey”
  public_key = file(var.public_key_file)
}

resource digitalocean_droplet myservers {
  count = 5
  name = “server_${count.index}”
  region = var.region
  image = var.droplet_image
  size = var.droplet_size
  ssh_keys = [ digitalocean_ssh_key.mykey.fingerprint ]
}
```

Example - Generate Ansible Inventory from Terraform

```
resource local_file inventory {
  filename = “inventory.yaml”
  content = templatefile(“./inventory.yaml.tpl”,
    {
      user = var.user,
      private_key = var.private_key_file,
      hosts = {
        for h in digitalocean_droplet.myservers:
          h.name => h.ipv4_address
      }
    }
  )
}
```

Example - Generate Ansible Inventory from Terraform

```
inventory.yaml.tpl
all:
  vars:
    ansible_user: ${user}
    ansible_connection: ssh
    ansible_ssh_private_key_file: ${private_key}
  hosts:
    %{- for name, ip in hosts }
    ${name}:
      ansible_host: ${ip}
    %{ endfor }
```

Structure of a Task

- Common attributes
  - ~~name~~
  - ~~register~~
    - structure of the output
    - stdout, stderr
    - register on a loop - https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#registering-variables-with-a-loop
  - ~~when~~
  - run_once
  - environment
- become
- become_user
- set_fact
  - https://techsemicolon.github.io/blog/2019/07/07/ansible-everything-you-need-to-know-about-set-facts/
  - https://stackoverflow.com/a/43499924/311624 - set_fact example
- until
  - https://ttl255.com/ansible-until-loop/

- Email - https://www.redhat.com/sysadmin/configure-gmail-using-ansible
- Uses password - Ansible vault
- Attributes to use
  - delegate_to
  - ignore_errors
- https://www.slideshare.net/sumit23kumar/hands-on-ansible-112396469

Modules

- ~~apt, apt_repository, apt_key~~
- ~~service~~
- ~~lineinfile~~
- copy
- stat
- ~~shell~~
- ~~user~~
- git
- iptables, ufw
- pip
- mysql_user, mysql_db