Linux Day Milano 2022

Ansible

The recipe for a good server

POLITECNICO OPEN
unix LABS

Luca Biscaldi
bisca@poul.org

What is Ansible?

What is Ansible?

Ansible is a *free and open-source* software that **automates**:

- **Provisioning**: preparing an IT infrastructure to execute specific tasks.
- **Configuration management**: maintaining the infrastructure in the desired operating state, even after updates or changes.
- **Applications deploy**: installation, configuration and startup of applications.
- **Orchestration**: coordination between IT systems.

Ok, all nice, but...

It's not like I have to manage a datacenter!

What can / do with Ansible?

When can Ansible be useful *to me*?

Some realistic use cases:

- I have to **configure from scratch** a new machine, but it's the tenth time I install Linux and I've had enough.

- I have a machine that I have been using for a long time and I want to **do a clean reinstall**, but I don't remember how I configured it two years ago.

When can Ansible be useful *to me*

- I have to **configure *a lot* of containers** in my new server and I don't wantto do it manually.

- I've already configured my machine but I want it to remain in a known **steady state over time**.

Why can't I use a *simple* script on every machine?

Ansible vs script

1. A script executes **actions** on single machines,

Ansible makes sure that the entire infrastructure is in a specific **state**.

Example 1: Ansible vs script

I want the file A to contain the line B

Script: Check that the file exists (otherwise create it),
Check that it contains the line (otherwise insert it).

Ansible: File A *must* contain line B.

Ansible vs script

2. Ansible uses a more **readable** language than a script.

Example 2: Ansible vs script

I want the file A to contain the line B.

Script:

```
#!/bin/bash
FILE_PATH=A
LINE=B
if [ ! -f "$FILE_PATH" ]; then
    touch $FILE_PATH
fi
if ! grep -Fxq "$LINE" $FILE_PATH; then
    echo "$LINE" >> $FILE_PATH
fi
```

Ansible:

```
- lineinfile:
    path: A
    line: B
    create: yes
```

3. Ansible allows the parallel management of more machines.

Example 3: Ansible vs script

I want both A and B packages to be installed
in HOST_A and HOST_B machines,
but only if the Linux distribution is Ubuntu.

Ansible:

```
- hosts: HOST_A, HOST_B
  tasks:
    - apt:
      pkg:
        - A
        - B
  when: ansible_distribution == 'Ubuntu'
```

Script:

```
askStackOverflow();
```

Ansible strenghts:

- **Minimal**: It does not need the target machines to have specific software installed (only `python` and a SSH server*).
- **Secure** : All* operations are transmitted using SSH.

Strengths

- **Simple**: The files that make up our project are intuitive to read.
- **Idempotent**: Successfully executed operations will always have the *same result*, regardless of how many times they are executed, or the initial state of the machine.
- **Scalable**: The number of managed machines does not influence the project maintenance difficulty.

I'm convinced,

Let's see how it works!

Before starting...

Ansible has to be installed only on one machine, called**control node**, that has to be able to communicate with all the machines we want to manage, called **controlled nodes**.

On the controlled nodes it is sufficient that `python` and a SSH server* are installed, nothing more is needed!

...Let's install it!

In order to install Ansible we can:

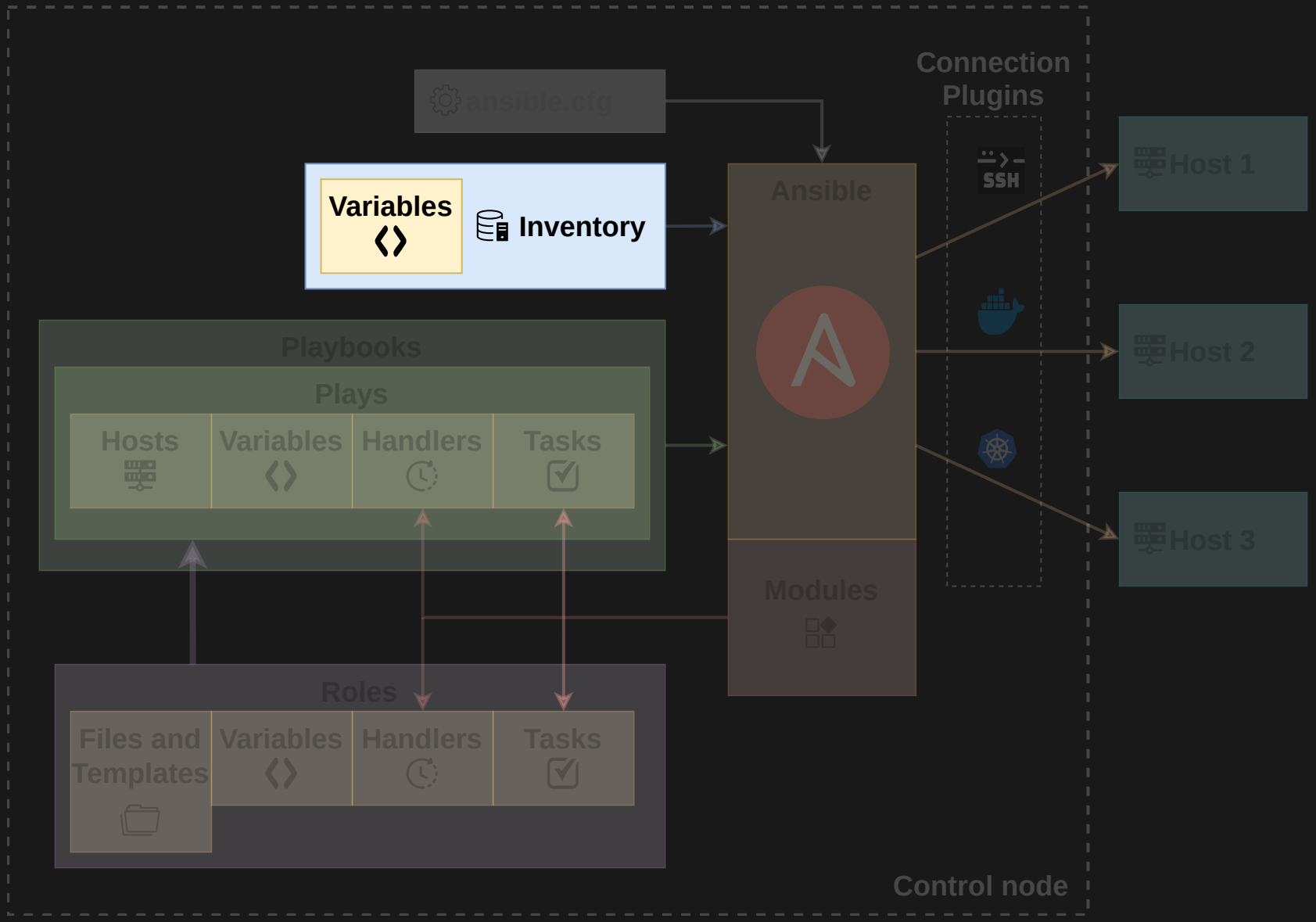
- Use the **package manager** of our GNU/Linux distribution.
- Use **`pip`**:

```
$ pip install ansible
```

- Compile it from source.

How do we "introduce" our other machines to Ansible?

Operating architecture


ansible.cfg
Variables
Inventory
Playbooks
Plays
Hosts
Variables
Handlers
Tasks
Roles
Files and Templates
Variables
Handlers
Tasks
Ansible
Modules
Connection Plugins
SSH
Host 1
Host 2
Host 3
Control node

Inventory

The **inventory** is the component in our project that defines the infrastructure Ansible will operate on.

It can be a single file (**static inventory**) with simple infrastructures, but we can also use more files, also editable by external services (**dynamic inventory**).

The default inventory is `/etc/ansible/hosts` (single file).

Content

Inside the inventory we can define:

- **Hosts**: target machines on which Ansible will act.
- **Groups**: logical "containers" to group hosts (or othergroups).
- **Variables**: properties associated to groups or hosts.

Format

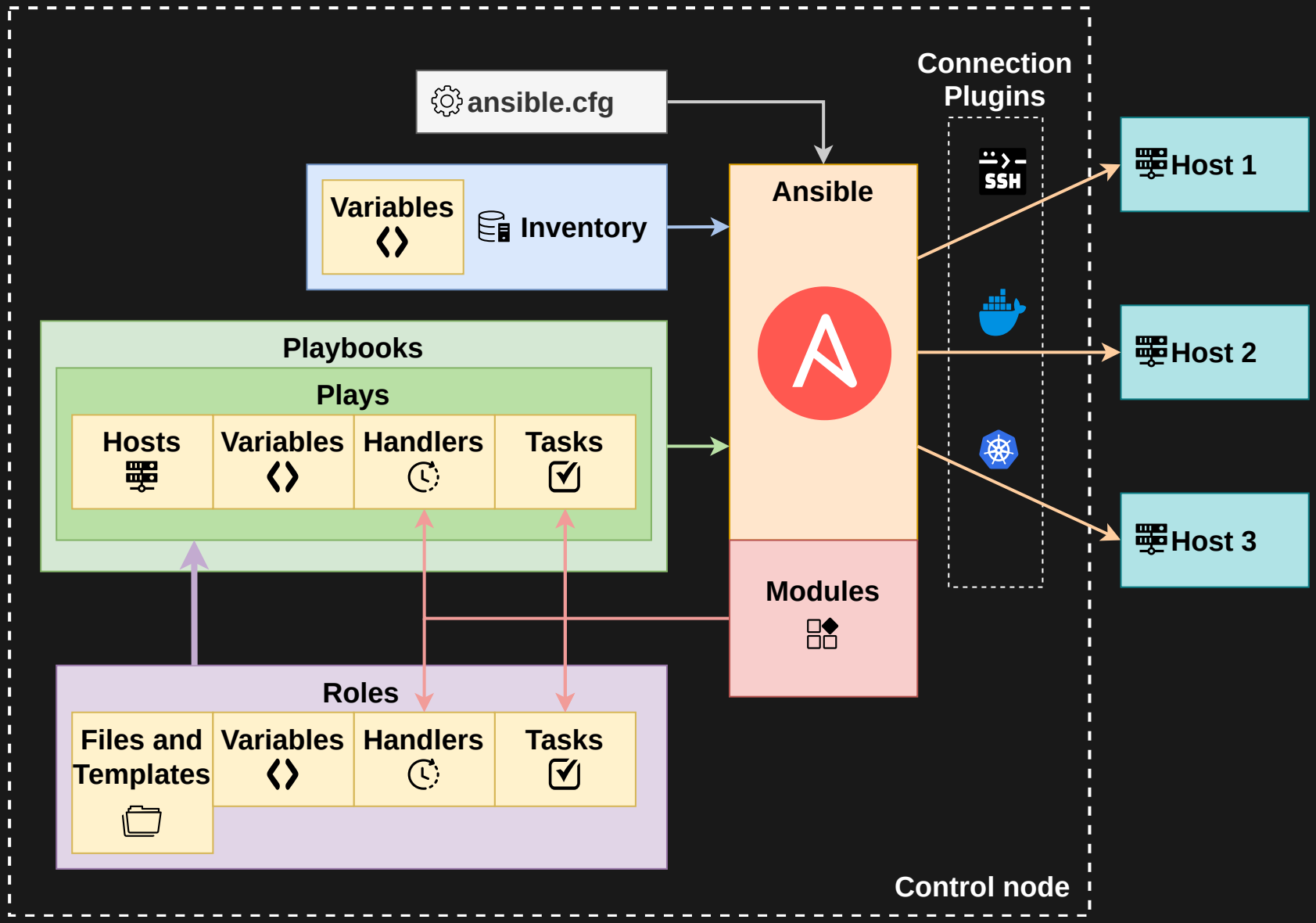
An inventory can be written in INI syntax or in YAML syntax.

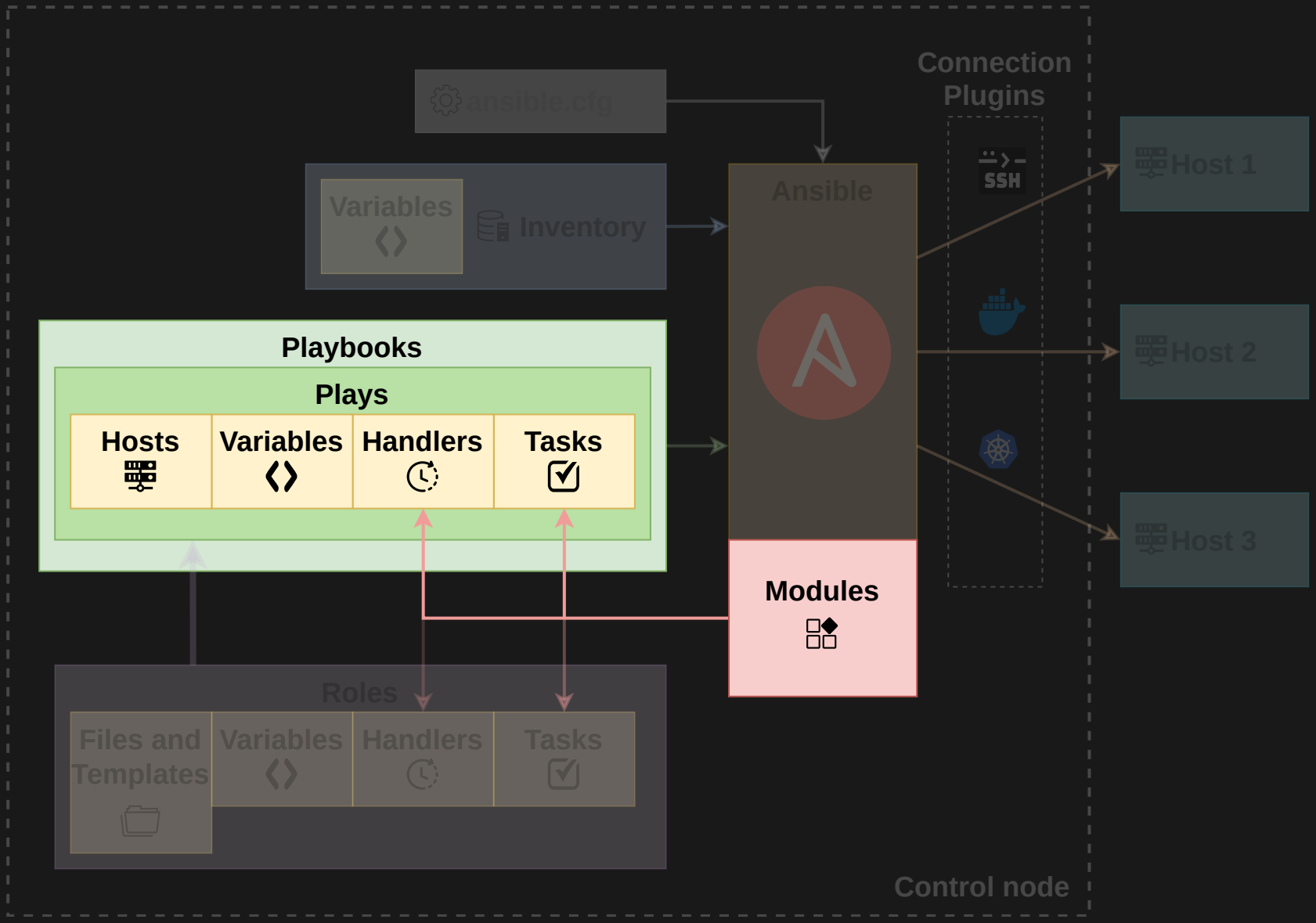
Inventory

INI format is linear and more convenient for simple infrastructures:

```
mailserver.example.com db_backend=db_three.example.com

[webservers]
foo.example.com db_backend=db_one.example.com
bar.example.com db_backend=db_two.example.com

[webservers:vars]
http_port=80

[dbservers]
db_one.example.com
db_two.example.com
db_three.example.com

[ubuntu:children]
webservers
```

Let's do something useful!


ansible.cfg
Variables
Inventory
Playbooks
Plays
Hosts
Variables
Handlers
Tasks
Roles
Files and Templates
Variables
Handlers
Tasks
Ansible
Modules
Connection Plugins
SSH
Host 1
Host 2
Host 3
Control node


ansible.cfg
Variables
Inventory
Ansible
Connection Plugins
SSH
Host 1
Host 2
Host 3
Playbooks
Plays
Hosts
Variables
Handlers
Tasks
Modules
Roles
Files and Templates
Variables
Handlers
Tasks
Control node

Modules

Modules are code units that Ansible executes on remote hosts.

Each module serves a specific function and might require specific parameters in order to be invoked.

Modules can be individually executed:

```
$ ansible <hosts> -m module_name -a module_parameters
```

Task

A **task** is an action unit. It includes (at least):

- The **module** to invoke.
- The **parameters** needed by the module.

Play

A **play** is a unit that contains all the information needed to execute a task, such as (at least):

- Information on the **hosts** on which we execute the tasks
- Information on the **tasks** to execute

Playbook

A **playbook** is a file that contains one or more **plays** to run, in **YAML** syntax.

Playbook example

```
---
- name: Update web servers
  hosts: webservers

  tasks:
  - name: Apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest
  - name: Apache service is enabled and running
    ansible.builtin.service:
      name: httpd
      state: started
      enabled: yes
```

A playbook can be run with:

```
$ ansible-playbook <hosts> <my_playbook.yml>
```

Handlers

A **handler** is a task that will be executed *only if* another task changes something on the remote host.

To call a handler we use the keyword `notify`.

Example

Restart Apache daemon only if another task changes its configuration file.

```
- name: My play
  hosts: webservers

  tasks:
  - name: Apache config file is updated
    ansible.builtin.copy:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
    notify:
    - Restart apache

  handlers:
  - name: Restart apache
    ansible.builtin.service:
      name: httpd
      state: restarted
```

Variables

Variables are objects containing information.
They can be defined:

Variables

1. Inside the **inventory**:

```
all:
hosts:
  mailserver.example.com
vars:
  my_variable: antani
```

or inside the `host_vars/` and `group_vars/` subfolders.

2. Inside a **play**:

```
- name: My play
  hosts: webservers
  vars:
    http_port: 80
  tasks:
    ...
```

Variables

3. Inside a **role**, in the `vars/` and `defaults/` subfolders.

4. From command line, with `-e variable=value` flag.

Using variables

Variables can be used by calling them with:

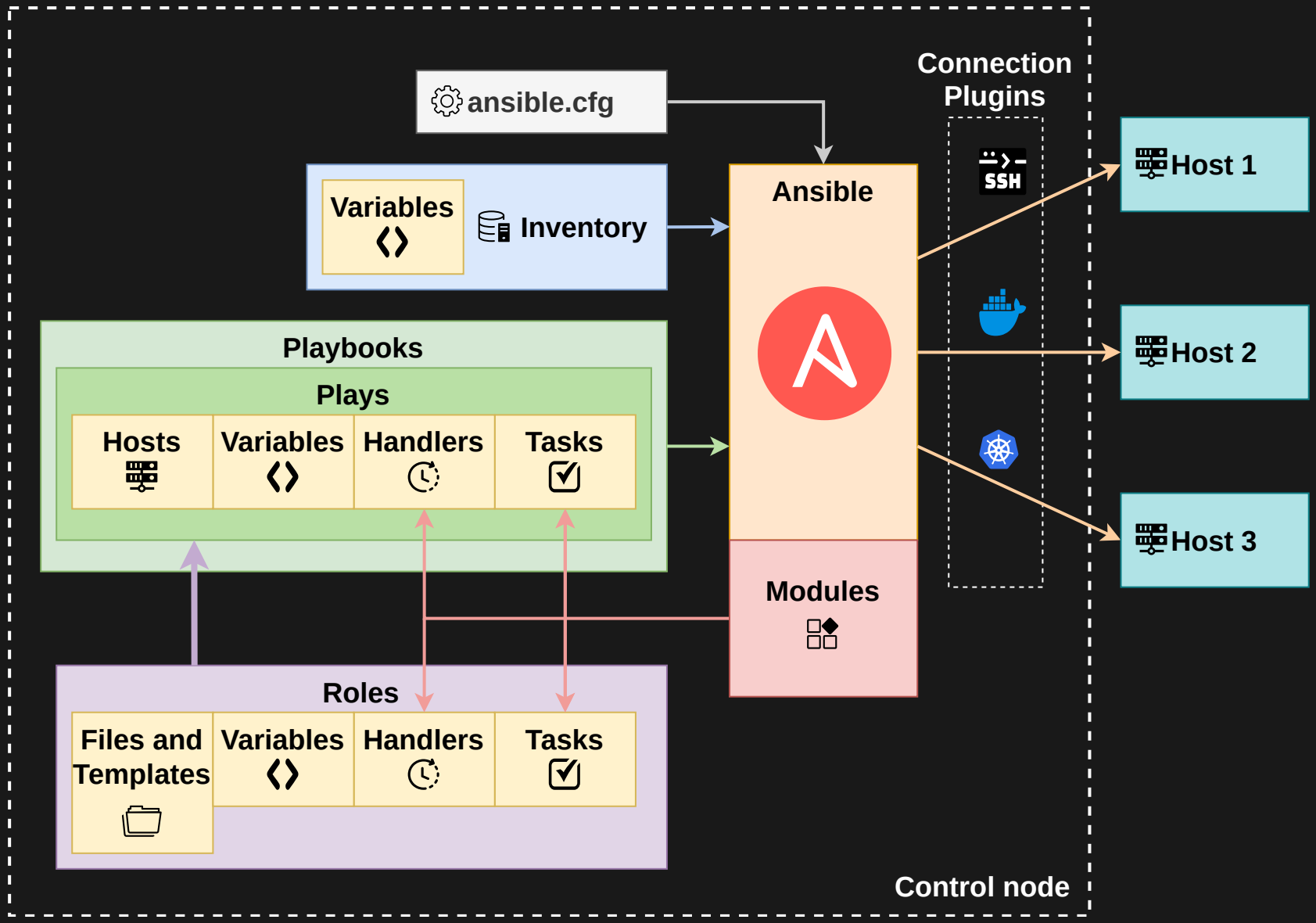
```
"{{ my_variable }}"
```

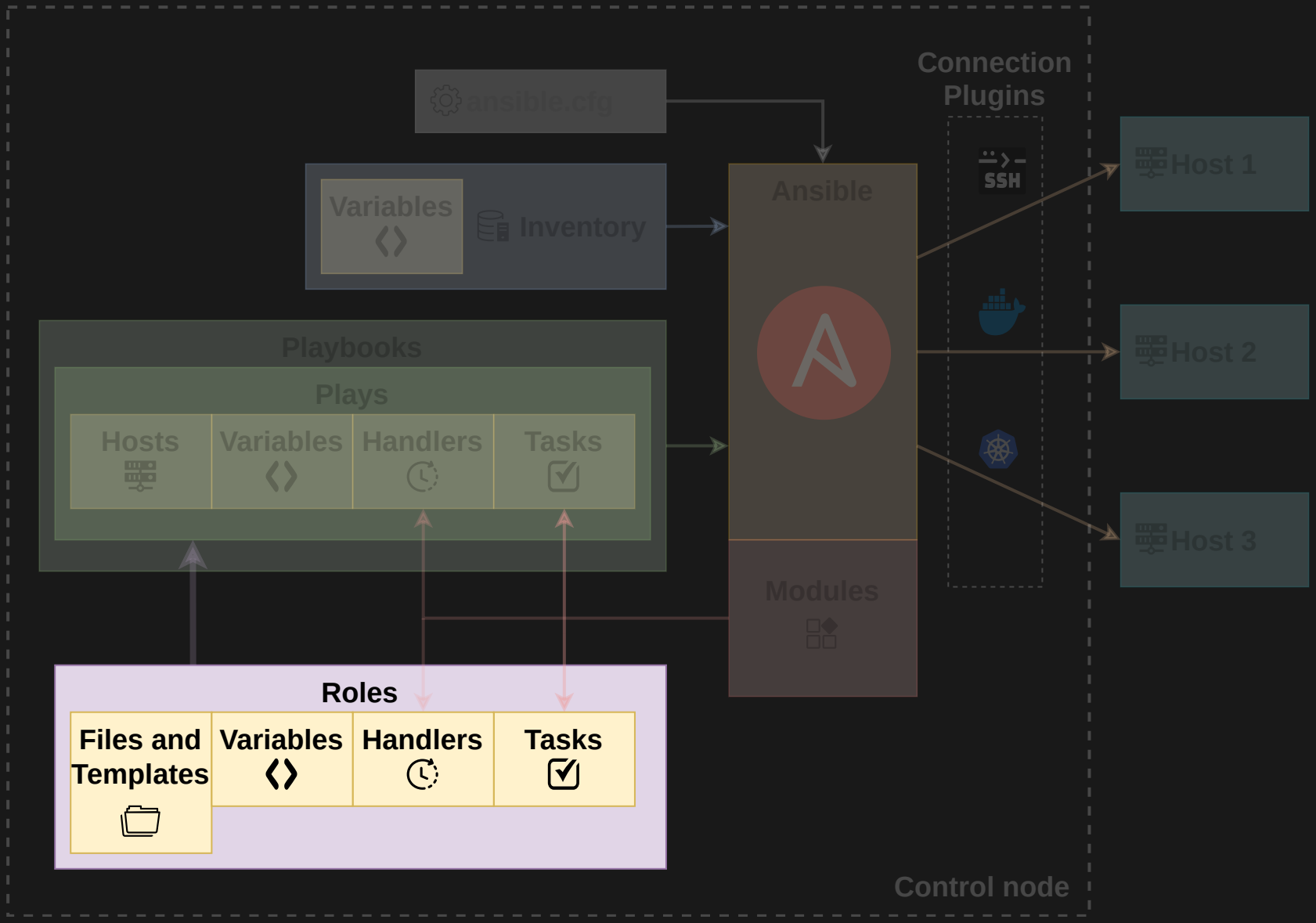
Double quotes are mandatory!

What if I declare the same variable in different places?

Precendence rules
(last one has precedence over the others):

1. Role defaults
2. Inventory
3. Play
4. Role variables
5. Task
6. Command Line


ansible.cfg
Variables
Inventory
Playbooks
Plays
Hosts
Variables
Handlers
Tasks
Roles
Files and Templates
Variables
Handlers
Tasks
Ansible
Modules
Connection Plugins
SSH
Host 1
Host 2
Host 3
Control node


ansible.cfg
Variables
Inventory
Playbooks
Plays
Hosts
Variables
Handlers
Tasks
Roles
Files and Templates
Variables
Handlers
Tasks
Ansible
Modules
Connection Plugins
SSH
Host 1
Host 2
Host 3
Control node

Roles

A **role** is a set of:

- Tasks
- Handlers
- Variables
- Files and templates

that can be loaded in a **play**.

What are roles for?

Roles are useful to **organize** and to make the project **modular**.

Example:

- I write a **common** role with the tasks I want to run on allmachines (i.e. import SSH keys, create users...).
- I write a **webservers** role with the tasks needed for the configuration of a webserver (i.e. install needed packages).

My playbook will be:

```
- name: Setup webservers
  hosts: webservers
  roles:
  - common
  - webservers

- name: Setup other machines
  hosts: others
  roles:
  - common
```

To use a role we can:

1. Add it in a play in the `roles` **section**:

```
- hosts: all
  roles:
  - MY_ROLE
```

2. **Include** it as a task in a play:

```
- hosts: all
  tasks:
  - name: My beautiful role
    include_role:
      name: MY_ROLE
```

3. **Import** it as a task in a play:

```
- hosts: all
  tasks:
  - name: My beautiful role
    import_role:
      name: MY_ROLE
```

Useful modules

Setup

Gather information about remote hosts.

```
$ ansible <hosts> -m setup
```

Setup module registers a special variable, `ansible_facts`, for each host. It is a dictionary containing *a lot of* useful information about the remote host:


```
{
    "ansible_all_ipv4_addresses": [
        "REDACTED IP ADDRESS"
    ],
    "ansible_all_ipv6_addresses": [
        "REDACTED IPV6 ADDRESS"
    ],
    "ansible_apparmor": {
        "status": "disabled"
    },
    "ansible_architecture": "x86_64",
    "ansible_bios_date": "11/28/2013",
    "ansible_bios_version": "4.1.5",
    "ansible_cmdline": {
        "BOOT_IMAGE": "/boot/vmlinuz-3.10.0-862.14.4.el7.x86_64",
        "console": "ttyS0,115200",
```

Setup module will be run *by default* at the start of every playbook.

To override this behaviour we can use

```
gather_facts: no
```

inside our play.

Template

Create a file based on a template. It's useful when the content of the file I want to create depends on variables.

Templates (by default) are located in the `[role]/templates/` subfolder and are written in *Jinja2* syntax.

Template example (for `/etc/network/interfaces`):

```
auto {{ interface.name }}
iface {{ interface.name }} inet static
    address {{ interface.address }}/{{ interface.netmask }}
    gateway {{ interface.gateway }}
```

The task that allows to use this template would be:

```
- name: update network interfaces
  template:
    src: my_template.jn2
    dest: /etc/network/interfaces
```

Other useful modules

- **copy**: it copies a file from the control node to the controlled node.
- **lineinfile**: it checks that a specified line is present inside a file.
- **file**: it manages files, directories and their properties.

What if I don't want to use SSH as `root` user?

Privilege escalation

Privilege escalation

Changing user, gaining administrative privileges.

Privilege escalation

How?

Using 2 parameters in the play:

- Declare that we intend to change user:

```
become: yes
```

- Declare which user we want to be (default is `root`):

```
become_user: root
```

What did we *not* mention?

I have a lot of secrets, do I keep them under the mattress?

Ansible Vault

Ansible Vault

It allows to save sensible information inside our project.

All information will be **encrypted** and a password will be asked every time we run our playbook.

I am *very lazy* and i don't want to write all my playbooks from scratch.

Ansible Galaxy

Ansible Galaxy

It's an hub where roles and entire projects are shared, from which we can freely draw.

Ansible Galaxy

How to use it?

From command line!

```
$ ansible-galaxy <action> <name>
```

- **search**: to search a term.
- **info**: to gather information about a role or a project we found.
- **install**: to install the role or the project locally.

I am *even lazier* and I am tired of using the terminal.

Ansible AWX

Ansible AWX

It's a GUI in the form of a **hostable service** that allows us to manage our Ansible projects.

Ansible AWX

Capabilities:

- **Realtime jobs update**: We can have the output of each executed task, for each machine, in real time.
- **Multiple users**: we can have different users, with different permissions, organizedin groups.
- **Logs**: we can see who did what.
- **Scheduling**: we can automate our automation.

Links

- Ansible Webpage
- Ansible Docs Page

Thank you for your attention!

POLITECNICO OPEN
unix LABS

Luca Biscaldi
bisca@poul.org