ANSIBLE

Writing Ansible module for fun and profit

Ganesh B. Nalawade
PSE, Ansible Engineering
Github/IRC: ganeshrn

Abhijeet Kasurde
SSE, Ansible Engineering
Github/IRC: akasurde

redhat

Agenda

- Setting up dev environment (workshop)
- Ansible overview
- Deep dive with Ansible module
- Develop hello world module (workshop)
- Develop your module-1 (remote execution with module debugging)
- Develop your module-2 (local execution with connection=local)
- Develop your module-3 (local execution with connection=httpapi)

About me - Ganesh

* Principal Software Engineer at Ansible by Red Hat

* Work primarily as upstream developer in Ansible Networking

* Worked extensively on Network management plane developing software for on/box automation and programmability infra.

* Co Organiser for Ansible meetup group in Pune

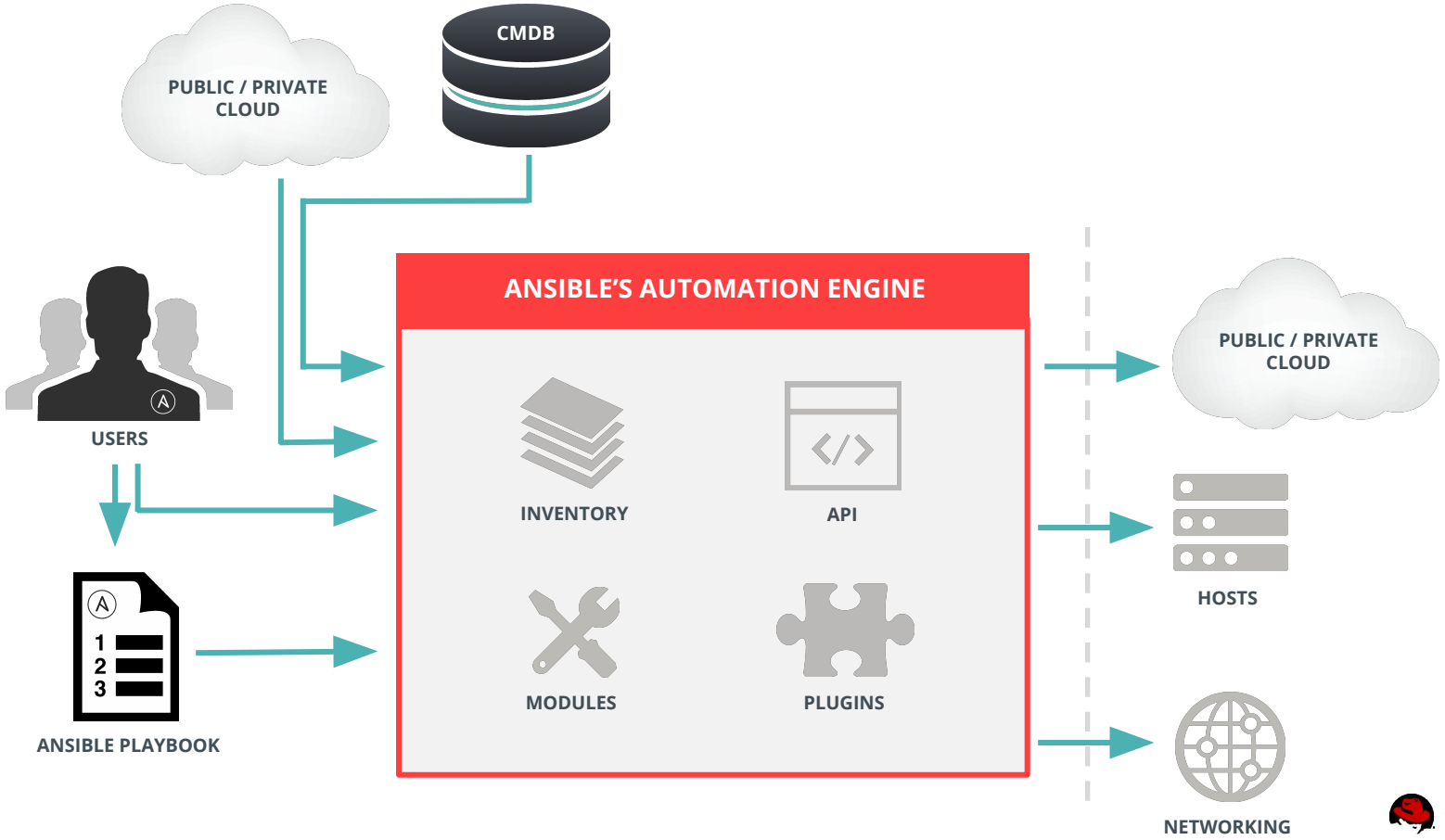
About me - Abhijeet

- Free and Open Source software evangelist
- Senior Software Engineer at Ansible by Red Hat
- Works primarily as maintainer for Ansible VMware and cloud related modules
- Co-host for Ansible Pune meetup group

Setting up dev environment

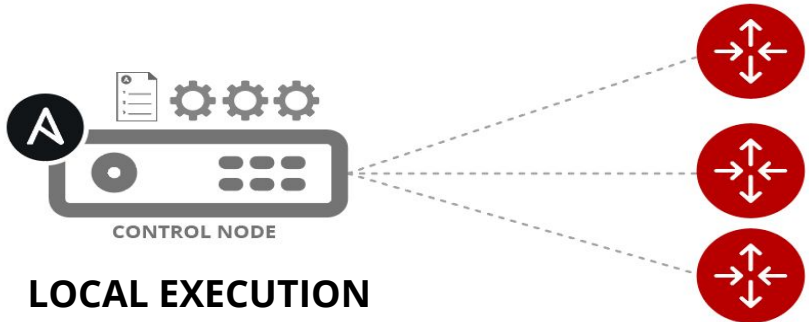
https://github.com/ganeshrn/ansible_module_development/tree/master/1.1-setup-dev-env

Ansible Overview



Ansible module execution types

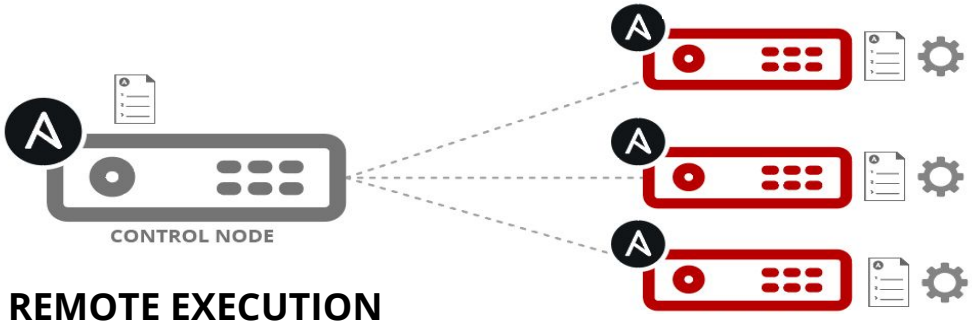
Module code is executed locally on the control node



LOCAL EXECUTION

NETWORKING DEVICES

Module code is copied to the managed node, executed, then removed

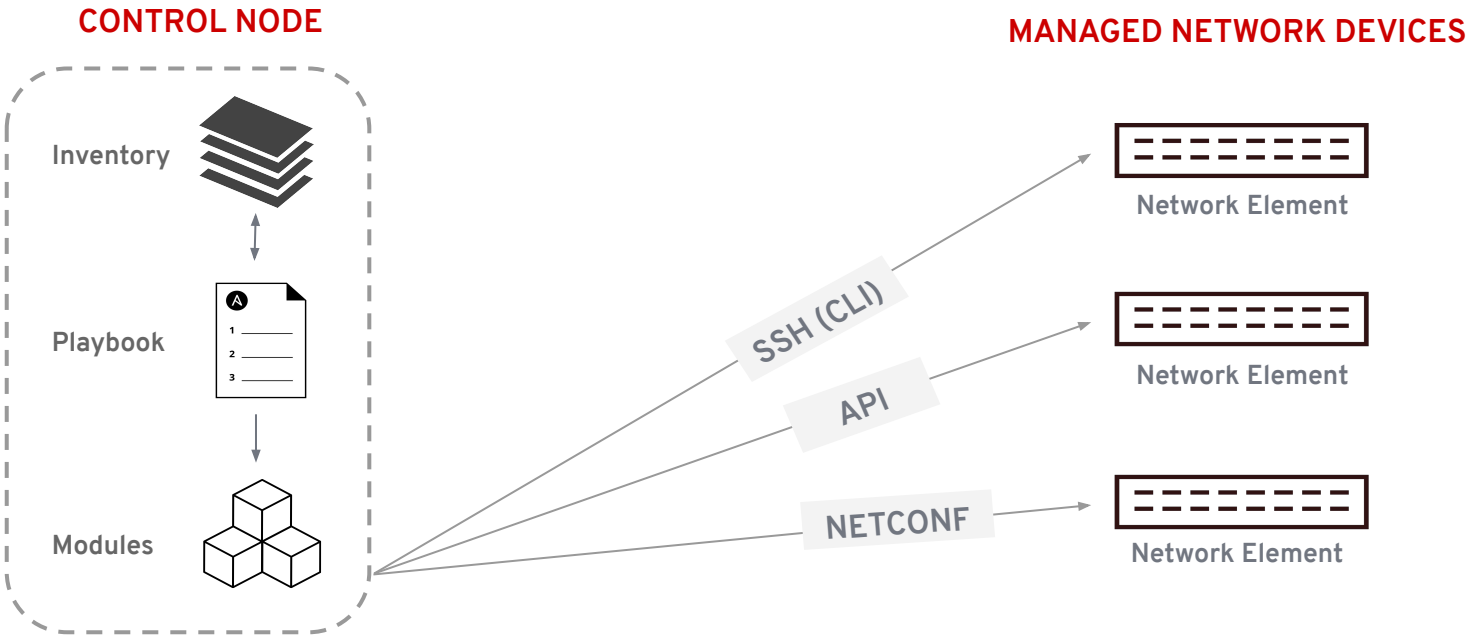


REMOTE EXECUTION

LINUX/WINDOWS HOSTS

redhat.

Ansible module execution types (contd.)



Managed Nodes (Inventory):
A collection of endpoints being managed via SSH or API.

Control Node:
Any client system (server, laptop, VM) running Linux or Mac OSX

Modules:
Handles execution of remote system commands

Module internals

AnsibleModule

- Parses and validates module arguments including complex parameter requirements.
- Define helper methods, which greatly simplifies module writing so that author can focus on the module functionality.
- These helper method include methods for
  - File operations permissions
  - Running commands
  - Failing/exiting gracefully for modules

Argument Specification

- Used to define parameters used for the module.
- Enforces and validated the parameters:
  - Value type
  - Default values.
  - Required parameters.
  - A list of “choices” that restrict the value of parameters.
  - Whether or not parameter should be logged.
  - Parameter aliases.
- The format is a Python dictionary of dictionaries.

Argument Specification (contd..)

```
# from the copy.py module
module = AnsibleModule(
    argument_spec=dict(
        src=dict(type='path'),
        original_basename=dict(type='str'),
        content=dict(type='str', no_log=True),
        dest=dict(type='path', required=True),
        backup=dict(type='bool', default=False),
        force=dict(type='bool', default=True, aliases=['thirsty']),
        validate=dict(type='str'),
        directory_mode=dict(type='raw'),
        remote_src=dict(type='bool'),
        local_follow=dict(type='bool'),
    ),
    ...
)
```

Argument Specification (contd..)

Assuming a standard Python module is being executed:

- Parameters are read from stdin and parsed from JSON.
- Aliases for parameters are configured.
- “no_log” parameters are handled.
- Arguments are compared against the spec to determine if there are any invalid parameters.
- Mutually exclusive parameters are checked.

Argument Specification (contd..)

- Default values are set (except for those marked required=True).
- Required arguments are checked.
- Required together/one of/if are checked.
- Defaults are set for everything.
- Check for “options”, which are basically nested argument spec.

Argument Specification (contd..)

The type parameter of the argument spec can be one of the following:

- Standard Python types (**str**, **bool**, **int**, **float**, **dict**, **list**).
- **path**, which ensures the value is a string and also fully expands the path (removing variables and shell shortcuts).
- **bytes,** which uses the *human_to_bytes()* to ensure the value is a Python byte array.
- **raw**, which does no other validations.

Other AnsibleModule Options

```
module = AnsibleModule(
    argument_spec=dict(
        ...
    ),
    mutually_exclusive=[...],
    required_one_of=[...],
    required_together=[...],
    required_if=[...],
    supports_check_mode=True,
    add_file_common_args=True,
)
```

Argument Specification (contd..)

mutually_exclusive:

- Ensures that only one of the specified parameters is set and raises an error if both are present.
- Format of each list element: [‘param1’, ‘param2’].
- Example...

Argument Specification (contd..)

```
# example from cloud/amazon/ec2.py
mutually_exclusive=[
    ['group_name', 'group_id'],
    ['exact_count', 'count'],
    ['exact_count', 'state'],
    ['exact_count', 'instance_ids'],
    ['network_interfaces', 'assign_public_ip'],
    ['network_interfaces', 'group'],
    ['network_interfaces', 'group_id'],
    ['network_interfaces', 'private_ip'],
    ['network_interfaces', 'vpc_subnet_id'],
],
```

Argument Specification (contd..)

required_one_of:

- Ensures that at least one of the parameters is present.
- Format of list elements: [‘param1’, ‘param2’, ...].
- Example...

Argument Specification (contd..)

```
# example from cloud/amazon/ec2_group.py
required_one_of=[['name', 'group_id']],

# example from packaging/os/pkgin.py
required_one_of = [
    ['name', 'update_cache', 'upgrade', 'full_upgrade', 'clean']
],
```

Argument Specification (contd..)

required_together:

- This creates a bidirectional dependency between two or more parameters, such that if one parameters, such that if one parameter is set the other must be set as well
- As with mututally_exclusive, there is no requirement that any of the specified parameters be set.
- List element format: [‘param1’, ‘param2’, ...]
- Example ...

Argument Specification (contd..)

```
# example from packaging/os/redhat_subscription.py
required_together=[
    ['username', 'password'],
    ['activationkey', 'org_id'],
    ['server_proxy_hostname', 'server_proxy_port'],
    ['server_proxy_user', 'server_proxy_password']
],
```

Argument Specification (contd..)

required_if:

- Creates a one-way dependency on other parameters if a given parameter is set to a certain value.
- An optional parameter can be set to specify at least of the dependent parameters be set, otherwise all listed parameters are required.
- Format of each list element.
  ['param1', 'value', ['param2', ...], required_one_of]
- Example ...

Argument Specification (contd..)

```
# example from cloud/amazon/ec2_group.py
required_if=[['state', 'present', ['name']]]

# example from packaging/os/redhat_subscription.py
required_if=[
    ['state', 'present', ['username', 'activationkey'], True]
],
```

Other AnsibleModule Configuration Options

support_check_mode=True|False:

- Use to indicate whether the module will honor check mode or not.
- Module typically use the following if/else construct to wrap actions that you modify state of the target machines.

```
if not module.check_mode:
    result = do_changes()
else:
    result = {'changed': True}
```

Module internals

AnsibleModule.exit_json / AnsibleModule.fail_json

- Helper function to gracefully exit from a module. These should always be used to terminate the execution of a module.
- exit_json:
  - Cleans up any files marked as needing to be cleaned up.
  - Adds some standard values to the result dictionary passed in from the module code(warnings, deprecations, file data etc.)
  - Call sys.exit() to terminate the python interpreter.
- Fail_json also:
  - Requires a failure message (‘msg’ in result dictionary).
  - Adds any traceback information which may have occurred during the module failure.
  - Call sys.exit(1) to terminate with failure return code.

AnsibleModule.run_command

- A helper method that uses the Python subprocess module to execute a command.

- Gracefully handles:
  - Sending data to the command, including binary data.
  - Reading back the output (including errors) while dealing with a possible prompt from the command.
  - Setting the current working directory (cwd).
  - Setting the umask during the command execution.
  - Setting additional environment variables.
  - Dealing with input/output.

Develop hello world module

https://github.com/ganeshrn/ansible_module_development/tree/master/2.2-Hello-world-module

Develop your module-1 (remote execution)

https://github.com/ganeshrn/ansible_module_development/tree/master/2.2-Hello-world-module

Develop your module-2 (local execution with connection=local)

https://github.com/ganeshrn/ansible_module_development/tree/master/2.4-Sample-flask-server-module-with-local-connection

Develop your module-3 (local execution with connection=httpapi)

https://github.com/ganeshrn/ansible_module_development/tree/master/2.5-Sample-flask-server-with-httpapi-connection

Contributing to Ansible

- Community groups
- IRC channels
- Google Groups:
  https://groups.google.com/forum/#!forum/ansible-project
  https://groups.google.com/forum/#!forum/ansible-devel
- Ansible galaxy

Thank You

GitHub / IRC: @ganeshrn
GitHub / IRC: @akasurde