

Let's prepare the exercise environment.

The VSCode editor will be displayed in the upper right corner of the screen. It may take some time to display.

Preparation Work

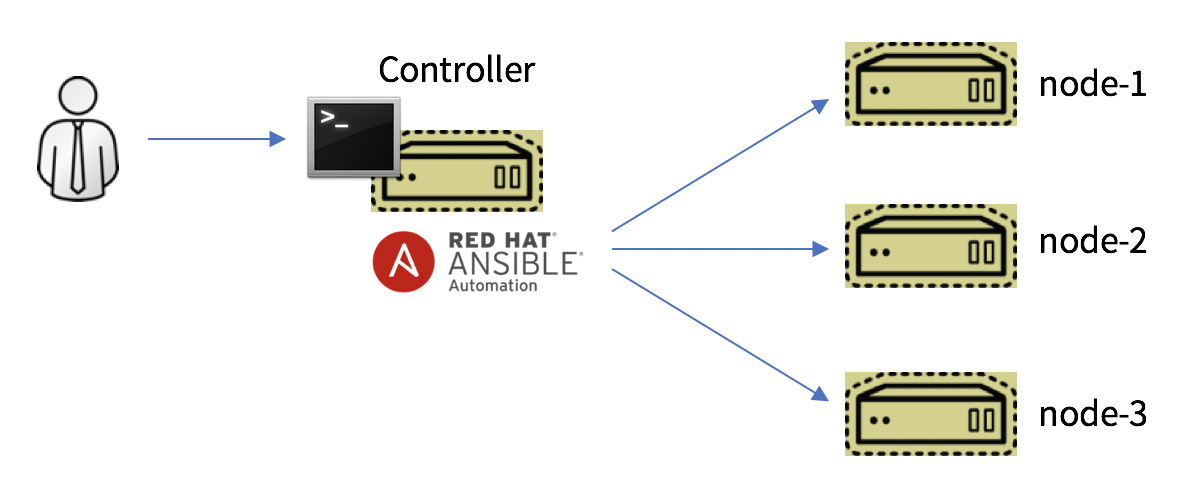
Prepare the exercise environment by running the following command. This should take about 1-2 minutes (the command will automatically be copied to the terminal and executed when you click on it).

apt install -y python3-pip && mkdir -p /opt/kata-materials && cd /opt/kata-materials && git clone --depth 1 https://github.com/irixjp/katacoda-scenarios.git . && pip install virtualenv && virtualenv ansible && source /opt/kata-materials/ansible/bin/activate && cd tools && bash ./kata\_setup.sh && cd ~/

Note: The Ansible environment for the exercise will be built in a virtualenv.

Environment Overview

In this exercise, we will use an environment built as follows. Three servers, node-1 , node-2 , and node-3 , are running, and we will use Ansible to perform various automated operations on them.



Note: These server instances will be started as containers.

Supplementary Information

At the top of the terminal, you will see the tabs node-1 , node-2 , and node-3 . Clicking on these will connect to port 80 on each server. Right now nothing is running on each node, so nothing will happen when you click on it, but we will use this tab in the exercise to access the HTTP servers we have built on the nodes.

Note: These tabs are actually accessing ports 8081, 8082, and 8083 on the host side and redirecting from there. The accesses are host:8081 -> node-1:80, host:8082 -> node-2:80.

If the step in the exercise asks you to "access the node with a browser," click on this tab.

# **Coding Conventions**

Ansible has a high degree of freedom in coding and can write playbooks in a variety of ways. However, there are cases where this degree of freedom can cause problems. That's happens when the team proceeds automation.

When an individual creates a playbook at will, some people add name to the task, but others do not.As such, when there is an imbalance in content for each individual, the cost of ensuring quality increases.

Therefore, the team needs coding conventions. By creating rules, teams can write a code in common, leading to equalization of skills and lower review costs. However, on the other hand, it is also necessary to check whether it complies with the terms and conditions.

Therefore, Ansible provides a way to automatically verify compliance with the terms and conditions, so we will learn how to use them.

## Ansible Lint

Ansible offers a program called [ansible-lint](https://github.com/ansible/ansible-lint" \t "_blank). This can be done by a static analysis of playbook to check if there is any violations the rules. By default, the rules you check are commonly used, and you can define your own rules.

We are preparing the following 2 playbooks as samples.

* ~/working/lint\_ok\_playbook.yml
* ~/working/lint\_ng\_playbook.yml

Both of these playbooks can run correctly and print the output ps-ef . Please do two things as a test.

cd ~/working

ansible-playbook lint\_ok\_playbook.yml

ansible-playbook lint\_ng\_playbook.yml

Both should have been successful. Now, let's apply ansible-lint to these 2 playbooks.

ansible-lint lint\_ok\_playbook.yml

This will be end normally.

Note: Warning may appear, but please ignore it.

ansible-lint lint\_ng\_playbook.yml

Example Error 1

[502] All tasks should be named

lint\_ng\_playbook.yml:6

Task/Handler: shell set -o pipefail

ps -ef |grep -v grep

Example Error 2

unnamed-task: All tasks should be named

lint\_ng\_playbook.yml:6 Task/Handler: shell set -o pipefail

ps -ef |grep -v grep

The second command should have an error as example. There should also be other errors.

Note: This is because there is a slight difference in the check for each version of the Lint. Basically, newer versions tend to be more stringent.

One of the errors is `All tasks should be named', which shows that all tasks violate the terms and conditions of 'need to keep their names'.

Let's see which rules ansible-lint checks by default.Execute the following command:

ansible-lint -L

You can see that a number of conventions are defined by default. These terms and conditions are tagged so that you can specify tags to apply and exclude them altogether.

To confirm the tag list, execute the following:

ansible-lint -T

You can exclude the tag you want to exclude by using the -x option. Run the line exclude for the rule so that 'int\_ng\_playbook.yml' becomes OK as a test. Check the error, check the tag of the rule which are violating and specify it following -x .

Note: -x can be used multiple times in a single command. For example ansible-lint -x unnamed-task -x yaml

Next, modify the playbook so that `lint\_ng\_playbook.yml' does not fail without using rule exclusion. When modified, run the following to check the results.

ansible-lint lint\_ng\_playbook.yml

## Define non-standard rules

In addition to standard checks, you can define rules specific to projects and organizations.

Custom rules are defined in python, and rules can be easily created by inheriting a class called AnsibleLintRule .

For more information, please check this [sample](https://github.com/ansible/ansible-lint/blob/master/examples/rules/TaskHasTag.py).

The following will be defined in the independent rules.

* Prevent prohibited operations (commands) from entering the playbook
  + For example, if you have a bug in your router's firm and you want to prohibit a command that causes the switch to hang when you execute the command.
  + Risky commands that can cause problems with other commands.

## Additional Check Tools

A more general LINT tool, [YAMLLint](https://github.com/adrienverge/yamllint" \t "_blank), is available to check variable naming conventions and wording for `name1. Use it as needed.

## Answer to exercises

* [lint\_ok\_playbook.yml](https://github.com/irixjp/katacoda-scenarios/blob/master/materials/working/lint_ok_playbook.yml)
* [lint\_ng\_playbook.yml](https://github.com/irixjp/katacoda-scenarios/blob/master/materials/working/lint_ng_playbook.yml)

BACKNEXT

# **Test Automation, Report Generation**

Ansible can also be used to automate testing and verification tasks. In particular, automating various verification tasks, such as large-scale tests and small but repetitive tests, can be very effective.

In this section, we will see how to create a playbook to run tests.

## Commonly used modules for testing

First, let's take a look at some modules that are often used in testing. Of course, there are many other modules that can be used to write automated tests.

* [shell](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/shell_module.html) module: Executes an arbitrary command and collects the results.
* [uri](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/uri_module.html) module: Issues an HTTP method to an arbitrary URL.
* \*\_command module: This module mainly issues arbitrary commands to network devices and collects the results.
* \*\_facts/info module: This module is primarily used to retrieve information about the environment.
* [assert](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/assert_module.html) module: Evaluates a conditional expression and returns ok if it is true.
* [fail](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/fail_module.html) module: Evaluates a conditional expression and returns failed if it is true.
* [template](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/template_module.html) module: Used to output the test results.
* [validate\_argument\_spec](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/validate_argument_spec_module.html) module: Used to validate role parameters.

Note: When testing an environment that has been built or modified with Ansible using Ansible itself, it is recommended to use a different module than the one used to build it. For example, you can use the shell module to check a file distributed using the copy module.

## How to write tests

A common pattern for testing with Ansible is to get information with shell , \*\_command , and \*\_facts , and then judge the result with assert and fail .

Sample

- name: get command AAA result

shell: exec AAA

register: ret\_AAA

- name: check AAA result

assert:

that:

- ret\_AAA.rc == 0

Normally, when playbook encounters an error, it will stop at the task that caused the error. This is not a problem for configuration, but for testing, the test will also stop in the middle. The test must run to the end whether an error occurs or not, and we need to be able to know how many of the total test items succeeded or failed. The test must run to the end whether an error occurs or not, and we need to be able to know how many of the total test items succeeded or failed.

Sample

- ignore\_errors: yes

block:

- name: get command AAA result

shell: exec AAA

register: ret\_AAA

- name: get command BBB result

shell: exec BBB

register: ret\_BBB

- name: get command CCC result

shell: exec CCC

register: ret\_CCC

- name: check test results

assert:

that: "{{ item.failed == false }}"

loop:

- "{{ ret\_AAA }}"

- "{{ ret\_BBB }}"

- "{{ ret\_CCC }}"

In the above sample, the results are judged in a batch loop. This method is convenient, but it is necessary to output information so that the register side can make decisions easily. If you want to set up complex conditions, you may need to write as follows.

- name: check test results

assert:

that:

- ret\_AAA.rc == 0 # Determine the return value

- ret\_BBB.stdout.find("string") != -1 # Output results contain string

- ret\_CCC.stdout.find("string") == -1 # Output results does not contain strings

In the that argument of assert , passing a condition as an array will be treated as an AND condition

## Create a test

Let's actually create a test. As a simple example, we will assume that we have a server with an httpd server installed and running. Specifically, we will test against a server that has run the following.

ansible node-1 -b -m yum -a 'name=httpd state=present'

ansible node-1 -b -m systemd -a 'name=httpd state=started enabled=yes'

To test the above, we will perform the following checks.

* Package httpd is installed
* Process httpd is present (running)
* The service httpd should be automatically started (enabled)

Edit the file ~/working/testing\_assert\_playbook.yml as follows.

---

- name: Test with assert

hosts: node-1

become: yes

gather\_facts: no

tasks:

- ignore\_errors: yes

block:

- name: Is httpd package installed?

shell: yum list installed | grep -e '^httpd\.'

register: ret\_httpd\_pkg

- name: check httpd processes is running

shell: ps -ef |grep http[d]

register: ret\_httpd\_proc

- name: Is httpd service enabled?

shell: systemctl is-enabled httpd

register: ret\_httpd\_enabled

- block:

- name: Assert results

assert:

that:

- ret\_httpd\_pkg.rc == 0

- ret\_httpd\_proc.rc == 0

- ret\_httpd\_enabled.rc == 0

* In the first block , we run the necessary test codes under ignore\_errors and register the results of each.
* In the second block , we check the results with the assert module. Normally, the block here is unnecessary, but we will write it for the next exercise.

Execute the playbook

cd ~/working

ansible-playbook testing\_assert\_playbook.yml

This Playbook should finish successfully.

The next step is to generate an error in the test. We purposely stop httpd and then run the test.

ansible node-1 -b -m systemd -a 'name=httpd state=stopped enabled=yes'

ansible-playbook testing\_assert\_playbook.yml

This time the test should have failed, because httpd is not running, and the check with assert failed.

## Create a report of the test results.

The next step is to output the test results as a report. It is common to use the template module, but here we will use the copy module and the jinja2 style notation to create a report.

Edit the previous file ~/working/testing\_assert\_playbook.yml as follows. The following always will be the added part.

---

- name: Test with assert

hosts: node-1

become: yes

gather\_facts: no

tasks:

- ignore\_errors: yes

block:

- name: Is httpd package installed?

shell: yum list installed | grep -e '^httpd\.'

register: ret\_httpd\_pkg

- name: check httpd processes is running

shell: ps -ef |grep http[d]

register: ret\_httpd\_proc

- name: Is httpd service enabled?

shell: systemctl is-enabled httpd

register: ret\_httpd\_enabled

- block:

- name: Assert results

assert:

that:

- ret\_httpd\_pkg.rc == 0

- ret\_httpd\_proc.rc == 0

- ret\_httpd\_enabled.rc == 0

always:

- name: build report

copy:

content: |

# Test Reports

---

| test | result |

| ---- | ------ |

{% for i in results %}

| {{ i.cmd | regex\_replace(query, '&#124;') }} | {{ i.rc }} |

{% endfor %}

dest: result\_report\_{{ inventory\_hostname }}.md

vars:

results:

- "{{ ret\_httpd\_pkg }}"

- "{{ ret\_httpd\_proc }}"

- "{{ ret\_httpd\_enabled }}"

query: "\\|"

delegate\_to: localhost

* The added always will generate a report of the test results. This way, the report will be generated even if assert fails.
  + In this report, Jinja2 is used directly in the content parameter of the copy module to create a file in Markdown format.
  + The regex\_replace filter replaces a string with a regular expression.
    - Here, | in the command is replaced with <span class='kc-markdown-code-copy'></span>&#124;. This is because <span class='kc-markdown-code-copy'></span>| is a delimiter when outputting the results in table format, so the <span class='kc-markdown-code-copy'></span>| in the command is replaced with another expression (&#124; ).

Try to run the test with a successful pattern. Restart httpd for this.

ansible node-1 -b -m systemd -a 'name=httpd state=started enabled=yes'

ansible-playbook testing\_assert\_playbook.yml

This test should be successful. Check the contents of the report file ~/working/result\_report\_node-1.md that should have been created (right click on the file to open it in Markdown preview mode).

Next, let's check the report by failing the test. Stop the httpd process and then run the test.

ansible node-1 -b -m systemd -a 'name=httpd state=stopped enabled=yes'

ansible-playbook testing\_assert\_playbook.yml

Check how the test reports have changed.

## Create configuration report

In the previous example, we have output the test results, but it is also possible to automatically generate configuration reports in the same way, and there are many examples of this being used in practice. In this section, we will try to generate a simple server configuration report.

Create the file ~/working/reporting\_playbook.yml as following.

---

- name: Report with Ansible

hosts: web

gather\_facts: true

tasks:

- name: build report

copy:

content: |

# Server Configuration Reports: {{ inventory\_hostname }}

---

| name | value |

| ---- | ------ |

{% for key, value in ansible\_default\_ipv4.items() %}

| {{ key }} | {{ value }} |

{% endfor %}

dest: /tmp/setting\_report\_{{ inventory\_hostname }}.md

delegate\_to: localhost

- name: concatenate reports

assemble:

src: /tmp

regexp: 'setting\\_report\\_\*'

dest: setting\_report.md

delimiter: "\n"

run\_once: true

delegate\_to: localhost

* gather\_facts: true Let setup run before running the playbook so that we can take use the results.
* {% for key, value in ansible\_default\_ipv4.items() %} This time, we are retrieving settings related to the network.
  + To check the ansible\_default\_ipv4 variable, run the following.
  + ansible node-1 -m setup -a 'filter=ansible\_default\_ipv4'
* assemble module: A module to combine files.
* run\_once: true If this option is specified, only one host will be executed even if there are multiple hosts. This is because the join operation should be executed only once.

cd ~/working

ansible-playbook reporting\_playbook.yml

When you run it, a file setting\_report.md will be created in the working directory, and you can check the contents. (Please check it in Markdown preview mode)

The report output here can be converted from html format to pdf using [pandoc](https://pandoc.org/" \t "_blank), so it can be submitted as a report if you make it look a little better.

A testing tool (framework) called [molecule](https://github.com/ansible-community/molecule) is also available for systematically executing tests like this one. In addition, molecule can be used to execute tests in a unified manner for high quality automation.

## Answers to the exercises

* [testing\_assert\_playbook.yml](https://github.com/irixjp/katacoda-scenarios/blob/materials/solutions/testing_assert_playbook.yml)
* [reporting\_playbook](https://github.com/irixjp/katacoda-scenarios/blob/materials/solutions/reporting_playbook.yml)

If you find any mistakes or corrections, please contact us at

* <https://github.com/irixjp/katacoda-scenarios/issues>

[Back to top page](https://www.katacoda.com/irixjp)