6. Creating modules

6. Creating modules

Ansible has modules for all of the common system administration tasks. A module is a reusable, standalone script. Ansible runs this script locally or remotely. These modules can communicate with your local machine, an API or a remote system to do specific tasks, for example changing a database password. All the core modules of Ansible are developed in python. These modules are divided into two groups of modules: *ansible-modules-core* and *ansible-modules-extra*. If you want to write your own module for local use, you can write it in any programming language.

6.1 ansible-modules-core

This group of modules contains all of the core modules that Ansible has, such as apt, template and copy. The modules in this group are solid and reviewed by a team from Ansible before changes are released.

6.2 ansible-modules-extras

The modules that doesn't belong in the core module group are stored in the extra group. These modules are maintained by the community. An example of an extra module is debconf or bundler. If you make changes that effects an extra module, the maintainer has to review your changes and is responsible for the module.

6.3 Setup environment

Before you can write your own module, you have to setup an environment to test your modules. This environment has nothing to do with the pervious steps you have done. First, we are gone make a new folder to work and test in:

```
Mkdir ansible-module
Cd ansible-module
git clone git://github.com/ansible/ansible.git --recursive
source ansible/hacking/env-setup
chmod +x ansible/hacking/test-module
```

Here we cloned Ansible from GitHub, and source a file that manipulate the environment. This makes sure that when you run the Ansible commands it points to the downloaded version instead the globally installed version. The last step we did make the file test-module executable. You may also need to install some packages such as *pyyaml* and *jinja2*. This can be done via pip:

```
pip install pyyaml jinja2
```

6.4 Writing modules in Bash

You can write a simple module using bash. Most of the time you will use python for writing a module. However, bash is a good start to write your first simple module. Let's start with making a simple module that uses a file and convert it into uppercases. To do this we first need to make a file and then make it executable:

```
touch test-file
chmod +x test-file
```

After this we can create a module, first we are going to create a module that returns
data that gets used in making and running a module. Ansible expects a JSON string as return, for this example we write a module that returns a JSON-encoded string. To do this you have to add the following to *test-file:*

```
#! /bin/bash
cat <<EOF
{“content”:”Hello World”}
EOF
```

After you have done this you can run it with the following command:

```
Ansible/hacking/test-module -m test-file
```

You now have written and test your first module. This module doesn’t do anything yet. To make a module that converts a file into uppercases, you have to make the module accept a filename as parameter and then make all the characters in the file uppercase. To do this your script needs to look like this:

```
#!/bin/bash
source $1
content=$(cat $file | tr '[:lower:]' '[:upper:]')
echo $content > $file
cat << EOF
{"content":"$content"}
EOF
```

When you run this module, it will convert a file to uppercases. But you have to make sure that the file exists first.

```
Ansible/hacking/test-module -m test-file -a file=example_file.txt
```

The module will change the characters in example_file.txt to uppercase characters. So the module does what you would expect it to do, but for Ansible there are still missing some metadata that is required. This is because Ansible module should be idempotent. Which means that a module can be executed multiple time the output will always be the same. Ansible modules report back when changes are made using the changed attribute. You can do this by checking in your module, to see if the content is the same both before and after the execution:

```
#!/bin/bash
source $1
original=$(cat $file)
content=$(echo $original | tr '[:lower:]' '[:upper:]')

if [[ "$original" == "$content" ]]; then
 CHANGED="false"
else
 CHANGED="true"
 echo $content > $file
fi

cat <<EOF
{"changed":$CHANGED, "content":"$content"}
EOF
```

This file contains all of the information Ansible needs to run is as a standalone module. You can test this module, the first time you run it you will notice that changed is true. If you run the same file again the output of changed will be false. The module you created is now idempotent.

6.5 Writing modules in python

Writing a module in python is very similar to writing a module in bash. The script is called with a file path, this contains your arguments as the first parameter. Python can be used to read and parse this file and build it out of your module. Ansible is shipped with a module called *ansible.module_utils.basic*, this module provides a boilerplate that you'd otherwise have to yourself. In this chapter we will make a module that sets the system time. We are going to create this module the hard way, so without the boilerplate. This is because you can't use the boilerplates in other programming languages.
Let's get started, you first need to make a file named *time-test.py*. The content of this file is a python script and would look like the following:

```
  #!/usr/bin/python
import datetime
import json

date = str(datetime.datetime.now())
print(json.dumps({
    "time" : date
}))
```

This module can be tested in the same environment you created earlier. To run this script, you can use the following command:

```
ansible/hacking/test-module -m ./time-test.py
```

The output should look like this:

```
{"time": "2020-09-17 11:34:03.432951"}
```

We can modify this module to allow us to set the time. We will do this by passing a key value pair in the form of *time=<string>* into the module. Ansible saves arguments internally in an argument file. This file is just a string, so any form of argument is good. In this case we will do parsing to treat our input as key=value.
We will use the following as an example to set the time:

time time="July 17 12:55"

if you don't set the parameters the module will just leave the time as it is and return it. Your script would look like the following:

```
#!/usr/bin/python
import datetime
import sys
import json
import os
import shlex

args_file = sys.argv[1]
args_data = file(args_file).read()

arguments = shlex.split(args_data)
for arg in arguments:

    if "=" in arg:
        (key, value) = arg.split("=")

        if key == "time":
            rc = os.system("date -s \"%s\"" % value)
            if rc != 0:
                print(json.dumps({
                    "failed" : True,
                    "msg"    : "failed setting the time"
                }))
                sys.exit(1)
            date = str(datetime.datetime.now())
            print(json.dumps({
                "time" : date,
                "changed" : True
            }))
            sys.exit(0)

date = str(datetime.datetime.now())
print(json.dumps({
    "time" : date
}))
```

This module will look for the inserted key and if this key is time then it will change the time. If no parameters were sent, this module will just return the time.
To test this module, you have to use the following command:

```
ansible/hacking/test-module -m ./time-test.py -a "time=\"July 17 12:55\""
```

It would return something like this:

```
{"changed": true, "time": "2020-07-17 12:55:00.000405"}
```

As mentioned before, there are some shortcuts that you can use when you write a module in python. Modules are transferred as one file, so an argument file is no longer needed. This means that the code is getting shorter and the execution time is faster.

6.6 Writing modules in other programming language

To write a module you don’t have to use python or bash, you can use any programming language you want. But if you want to use the core modules, you need to write your module in python, otherwise you have to implement the argument handling yourself. We would recommend to write your modules in python, because of this core modules that handles the arguments.