redhat.

Ansible Hands-On Lab

Ansible Hands-on Lab

Piros
Gaston Geenslaan 11 B4
3001 Leuven

with special thanks to:

Red Hat
Dieter De Moitie
Ward De Rick
Alexander De Wispelaere

Chapter 1

Ansible Engine Exercises

Prepare your laptop

 Do you know your unique number and username? They have been assigned to you at the registration desk, and are printed on your badge.

 Connect to the wifi. Details are on the screen.

 Are you using Windows and don't have ssh? Install portable putty from http://portableapps.com/apps/internet/putty_portable

 Open a terminal session on your laptop and ssh to your student environment. Your username is printed on your badge. We have 3 tower servers set up for you in a Tower cluster configuration. Choose the correct Tower server, based on the information below:

- wsuser1 to wsuser 15 log in to Tower server **t1.piros.be**
- wsuser16 to wsuser30 log in to Tower server **t2.piros.be**
- wsuser31 and up log in to Tower server **t3.piros.be**


```
ssh wsuserX@tY.piros.be
```

The password is **PirosUser!**

Set up your ssh key

 Generate your ssh key-pair. For this lab, do **NOT** specify a passphrase and accept all defaults


```
ssh-keygen
```

 Copy your public key to your student server


```
ssh-copy-id <student ip>:
```

where <student ip> is the IP address you just provisioned using Ansible Tower in the Azure Public Cloud. It will ask for the password. This is the password you specified yourself when you provisioned the server. If you don't know your IP address, this is how you can find it:

1. Log in to Ansible Tower with your credentials
2. Go to JOBS in the top part of the screen, and find and click the job you executed earlier to create your vm (create_azure_vm_X)
    a. Download the output, or click the output to read it in your browser
    b. Find the privateIPAddress and remember it (write it down)

 Further learning: https://docs.ansible.com/ansible/intro_getting_started.html

Create an inventory for your server

The inventory file holds all your hosts grouped any way you want

- Create the directory $HOME/ansible


```
mkdir -p $HOME/ansible
```

- Create the file $HOME/ansible/hosts and give it the following content:

/$HOME/ansible

```
[testservers]
<student ip>
```

Where <student ip> is the IP address of the server you just provisioned using Ansible Tower (and who's IP address you wrote down earlier, right…?).

You need an editor for this. Both vi (vim) and nano are installed. If you are unfamiliar with Linux, use nano, as it is more user friendly for newbies.

If you did **NOT** read the instructions right and used a different username than the username you logged into Tower with (you know, wsuserX), then shame on you! To make it work anyway, you need to specify the user in the inventory, like so:

/$HOME/ansible

```
[testservers]
<student ip> ansible_user=<username>
```

You can add all kinds of extra vars in the inventory to make special transport requirements work.

Further learning: https://docs.ansible.com/ansible/intro_inventory.html

Ping Pong

The Ansible ping module is a great way to test if ansible is able to communicate with all your servers:

```
ansible all -m ping
```

If all is well, it will answer with a green colored "Pong". Note that the ansible ping module does not do a real ICMP ping.

If the returned output (log) is colored red, that means something went wrong. Yellow means it was successful and something on the target server changed. Green means successful and nothing changed.

Further learning: https://docs.ansible.com/ansible/ping_module.html

List Facts

Facts are key-value pairs detailing an enormous amount of metadata about your server(s). Each playbook run starts with getting your servers' facts to use them in your playbooks. They are not available when running ad-hoc commands. Facts are implemented by using the setup module.

- List all facts of your servers

```
ansible all -m setup
```

Have a good look at all the facts that ansible gets from your servers. You can use them all in your playbooks.

- List a single fact of your servers: the fqdn

```
ansible all -m setup -a "filter=ansible_fqdn"
```

You can also use wildcards in the filter argument

We use "all" as the server pattern here. This means all unique servers in the inventory.

 Further learning: https://docs.ansible.com/ansible/setup_module.html

Install apache

You can install packages, such as the Apache web server:


```
ansible testservers -m yum \
-a “name=httpd state=latest”
```

You will notice this takes a while and will fail, because the server we use doesn't have the correct privileges to install packages on the target server. However, this user is member of the wheel group on the target server, so we can sudo to root:


```
ansible testservers -m yum \
-a “name=httpd state=latest” \
--become --ask-become-pass
```

 Further learning: https://docs.ansible.com/ansible/yum_module.html

Variables and templates

Variables can be specified on multiple levels. In the inventory on both group and individual host level, and in playbooks (which we will introduce later). Although you can specify variables inside the inventory file, it's recommended to set them up in separate files, to be able to version control different sets of variables using for example git.

Let's create a group variable.

- First create the directory structure $HOME/ansible/group_vars/testservers:


```
mkdir -p $HOME/ansible/group_vars/testservers
```

- Now create a yaml formatted file myvars.yml (or any name you like, but it should end in .yml) in this directory with the following content:

/$HOME/ansible/group_vars/testservers/myvars.yml

```
---
student_name: <your name>
```


You can place as many files with as many variables in these group_var directories as you like.

You can use the powerful jinja2 templating engine that is built in in Ansible. You can use this to dynamically and programmatically convert template files into (configuration) files with the correct content for your server, replacing vars with the actual value.

- Create a directory templates in your home directory:


```
mkdir -p $HOME/templates
```

- Create a file called index.html.j2 in this directory with the following content:

/$HOME/templates/index.html.j2

```
<html>
<head><title>Ansible Workshop</title></head>
<body>
<div>This server belongs to {{ student_name }}</div>
</body>
</html>
```

Now install this template on the target server while replacing the variables with the correct values. We use the template module for this. The arguments are the source on the local machine and the destination on the target server(s):

```
ansible testservers -m template -a \
“src=$HOME/templates/index.html.j2 \
dest=/var/www/html/index.html” \
--become --ask-become-pass
```

Further learning: https://docs.ansible.com/ansible/playbooks_variables.html

Start the apache web server service

Using the service module you can control system services.

Let’s start and enable the Apache web server you just installed:

```
ansible testservers -m service -a \
“name=httpd enabled=yes state=started” \
--become --ask-become-pass
```

Since we are limited in the number of public IP addresses, your VM doesn’t have one. To test the web server, curl from the tower machine to the http server:

```
curl -kv http://<student ip>
```

Do you see the index.html you installed? No? So, what's wrong?

You forgot to open up the http service in the firewall! Read on to solve that.

 Further learning: https://docs.ansible.com/ansible/service_module.html

Firewall module

By now, you should see how ansible ad-hoc commands work. As a test, try to build the ad-hoc command to open up the http port in the firewalld firewall on the testservers. You can find the firewalld module documentation on https://docs.ansible.com/ansible/firewalld_module.html

 Usually, simply googling ansible module <modulename> brings you straight to the correct page.

Read the available arguments carefully and execute the correct ad-hoc command using this module to permanently and immediately open the http port.

Test again with curl. Does it work now? (hint: it should)

 Further learning: https://docs.ansible.com/ansible/modules.html

Vaulting

Vaulting is a great way to hide (configuration) data, such as passwords, to the people that execute Ansible playbooks. This way, you can easily have users execute Ansible playbooks that contain sensitive data without exposing this data to them.

Let's see how this works:

- First, create a directory for variable files specifically for your server:


```
mkdir -p $HOME/ansible/host_vars/<student ip>
```

 Make sure you use the exact same student server name or ip as you put in the inventoryfile $HOME/ansible/hosts.

- Next, create a vault. A vault is simply an AES encryoted file. You use a special command ansible-vault for this:

```
ansible-vault create \
$HOME/ansible/host_vars/<student ip>/mysecrets
```

Ansible-vault will ask you for a password. This password is used to unlock the contents of this file when used.

- Treat this file just like any other var file. Give this file the following content:

/$HOME/ansible/host_vars/<student ip>/mysecrets

```
---
my_secret_data: Ans1bl3=Gr3aT!
```

And save this file (include the ---, remember, it's yaml formatted). Try to look at the file's content. You will see it's encrypted:

```
cat $HOME/ansible/host_vars/<student ip>/mysecrets
```

You can view or edit it though:

```
ansible-vault view \ $HOME/ansible/host_vars/<student
ip>/mysecrets
```

```
ansible-vault edit \ $HOME/ansible/host_vars/<student
ip>/mysecrets
```

- Next, change the index.html.j2 template and add this super secret var:

/$HOME/templates/index.html.j2

```
<html>
<head><title>Ansible Workshop</title></head>
<body>
<div>This server belongs to {{ student_name }}</div>
<div>The secret data is {{ my_secret_data }}</div>
</body>
</html>
```

- Finally, to be able to execute an ad-hoc command or playbook with a vault involved, you need to specify the vault password:


```
ansible testservers -m template -a \
“src=$HOME/templates/index.html.j2 \
dest=/var/www/html/index.html” \
--become --ask-become-pass \
--ask-vault-pass
```

Now check with curl if the secret can be revealed.

Asking for passwords interactively is very safe, but not very automate-friendly. You can use a password file with the --vault-password-file argument, but you need to make really sure that file is very well protected. Another great method is using a python script to provide the password. Another way is to use Tower as the credential store for accessing secrets. Finally, Ansible has methods to access enterprise vaults.

Further learning: https://docs.ansible.com/ansible/playbooks_vault.html

Playbooks

Ad-hoc commands are nice for, you know, ad-hoc things. But the real super hero powers are unleashed when you use playbooks!

Playbooks are basically a bunch of ad-hoc commands put together in a file to be executed in a particular order on a particular set of servers.

Simply put, playbooks are the basis for a really simple configuration management and multi-machine deployment system, unlike any that already exist, and one that is very well suited to deploying complex applications.

Playbooks can declare configurations, but they can also orchestrate steps of any manual ordered process, even as different steps must bounce back and forth between sets of machines in particular orders. They can launch tasks synchronously or asynchronously.

We have prepared a simple playbook for you as an example to play with. Just copy it over to your home directory:


```
cp /workshop/install_cockpit.yml $HOME
```

And have a look at it:


```
cat $HOME/install_cockpit.yml
```

Can you understand the structure and the content of this playbook? What does it do? (we did not add comments in the playbook on purpose)

Note the difference in notation between the first two tasks and the third task. Both are possible, but the last notation format is prepared.

To execute a playbook, you use the ansible-playbook command. Let's execute this playbook on your student server:


```
ansible-playbook install_cockpit.yml \
--become --ask-become-pass \
--ask-vault-pass
```

As you might have noticed, you do not have to specify a server pattern nor sudo information, because these are now part of the playbook. You can also make the sudo password part of the playbook, but that's not very… smart, is it?

As you can see in the returned logging, the first task executed is always to run the setup module to gather the latest facts. That task is automatically added to the playbook.

Check if cockpit is installed and functioning. Since we don't have public ip's, this will have to be done from your tower:


```
curl -kv https://<student ip>:9090
```

You should get a web-page (in raw html) with login fields.

 Further learning: https://docs.ansible.com/ansible/playbooks_intro.html

Chapter 2

Ansible Tower Exercises

Delegation

Let's take another look at the Ansible Tower interface. Log in as wsuserX again

- "Projects" are where we pull in code
- "Inventories" are where we pull in lists of nodes
- "Job templates" are where we connect these two above

As you'll notice, everything is read-only for the user wsuserX

Go to Settings/**teams**/teamX/**permissions**

- Notice you have been delegated permissions to use and execute various objects.

Let's create some stuff!

- Log out.
- Log back in with the wsadminX user. Same password.
- Go to **projects** and create a new one
    - SCM URL: https://github.com/mcielen/ensure-cis-compliance.git
    - Assign permissions to use to teamX
    - Save it (saving it will sync it from Github)
- Assign **use** permissions to teamX
- Go to **inventories**, and create a new one
    - create a new inventory
    - add a new source of type Microsoft Azure Resource Manager and use azure_credsX as the cloud credentials
        - the region is Europe West
- Sync the group with the icon

- assign **use** permissions on the inventory to teamX
- go to **settings/credentials** and create a new one
    - make it a machine credential
    - enter the name/password you entered when creating your vm
    - set the **privilege escalation** to "sudo" and enter the same password as **privilege escalation password**
        - privilege escalation name = root
    - assign **use** permission to teamX
- wsuserX now has the required permissions to create a job template with this inventory/credentials/projects
- Log out
- Log in as user wsuserX again to create a template as wsuserX
    - With current permissions, wsuserX can create other projects and execute them with the credentials provided, or wsuserX could run the same project against other inventories and credentials. Alternatively, wsadminX could create the job template and assign execute permissions to wsuserX. In the latter case, wsuserX can only execute this specific job template and nothing more
- Go to **templates**, create a new job template
    - job type needs to be "Run"
    - Inventory needs to be the one you created earlier
    - Machine credential needs to be the one you created earlier
    - Project needs to be the one you created earlier
    - Playbook needs to be main.yml
- Assign **execute** permissions to teamX
- Add a survey with one field, variable name needs to be "vm_name"
    - ask for a host to run the playbook on
- Go to **templates**, run your newly created template
- Fill in the name of the machine you created earlier
- Watch how everything catches fire. I mean, completes flawlessly

 Recap: we have created a project, inventory and template and delegated usage rights to wsuserX and this user created a job template.

 Homework: try to think up a good use case for this in your own datacenter ;-)

Workflow templates (aka wickedly advanced stuff)

- As wsadminX, go to **templates**, create a new Workflow template
    - try and chain the create_azure_vm_X job and the job you created together
    - You'll need to add a survey to ask for
        - vm_name
        - admin_user
        - admin_pass
- Play around in this intercae a bit. See what it offers in terms of possibilities.
- Please only execute with the exact same vm_name as used before. Otherwise we will run out of cloud. I kid you not.

Chapter 3

Ansible Engine Advanced Exercises

All commands are to be executed on the Ansible host unless specified otherwise

Create a role template

You can create the folder structure for an Ansible role easily by issuing the following command at the cli:


```
ansible-galaxy init testrole
```

This command will create a directory with the name you specified after the init (testrole in this case) in your current directory. Inside the newly created folder a lot of subdirectories are created to put the various parts of an Ansible role in.

If you want to customize what files and directories are set up by the init command, you can specify a skeleton directory:


```
ansible-galaxy init \
--role-skeleton=/path/to/skeleton \
role_name
```

Import a role from Ansible Galaxy

You can search for roles hosted on Ansible Galaxy both via a webbrowser and via the cli:

```
ansible-galaxy search ntp --author geerlingguy
```

It's also possible to do a direct install of a role found on Ansible Galaxy from the cli:

```
ansible-galaxy install geerlingguy.ntp
```

Test the imported role

```
cd $HOME
vi ntp.yml
```

Make sure your file looks like this:

$HOME/ntp.yml

```
---
- hosts: testhosts
  roles:
    geerlingguy.ntp
```

You can copy the content from https://github.com/wardderick/ansible-advanced/blob/master/ntp.yml

Edit the ntp role so it uses the correct timezone:

$HOME/roles/geerlingguy.ntp/defaults/main.yml

```
---
ntp_enabled: true
ntp_timezone: Europe/Brussels

ntp_manage_config: false

# NTP server area configuration (leave empty for
'Worldwide').
# See:
http://support.ntp.org/bin/view/Servers/NTPPoolServer
s
ntp_area: ''
ntp_servers:
  - "0{{ '.' + ntp_area if ntp_area else ''
}}.pool.ntp.org iburst"
  - "1{{ '.' + ntp_area if ntp_area else ''
}}.pool.ntp.org iburst"
  - "2{{ '.' + ntp_area if ntp_area else ''
}}.pool.ntp.org iburst"
  - "3{{ '.' + ntp_area if ntp_area else ''
}}.pool.ntp.org iburst"

ntp_restrict:
  - "127.0.0.1"
  - "::1"
```

Now you can apply this role to all servers in the group testservers by running:


```
ansible-playbook ntp.yml -f 1
```

The -f parameter sets the parallelism to 1 so that only 1 server gets updated at the same time.

List installed roles

If you want to get a list of all roles that are installed in your Ansible role path, you can issue the following command:


```
ansible-galaxy list
```

 Further learning:
https://docs.ansible.com/ansible/latest/reference_appendices/galaxy.html

Dynamic inventories

Instead of manually creating inventories, you can also use dynamic inventories. On https://github.com/ansible/ansible/tree/devel/lib/ansible/plugins/inventory and https://github.com/ansible/ansible/tree/devel/contrib/inventory you can find a lot of different dynamic inventories.

Since we are using Azure for this lab, this will be the dynamic inventory we will be using in this lab. You can use as many inventories as you want, by placing all the desired inventories in the directory you configure in your Ansible config.

First you go to the directory where you will place your inventories directory.


```
cd $HOME
```

Then you create the directory where you will place all your inventories. The name can be anything you want as long as you use the same name in your Ansible config.


```
mkdir inventories
```

Afterwards you move the already existing static Ansible hosts file to the newly created directory.


```
cp /etc/ansible/hosts $HOME/inventories
```

Download the dynamic inventory into the directory

```
cd inventories
vi azure_rm.py
```

Copy the contents of https://raw.githubusercontent.com/ansible/ansible/devel/lib/ansible/plugins/inventory/azure_rm.py to the file azure_rm.py and make it executable. All your dynamic inventories should be executable as they are scripts that are being executed each time you run Ansible.

```
chmod +x azure_rm.py
```

Now you still have to reconfigure Ansible to make use of the created inventories directory.

```
cp /etc/ansible.cfg $HOME/.ansible.cfg
vi $HOME/.ansible.cfg
```

Look for the line inventory and make sure it is "inventory=$HOME/ansible/inventories".

Look for the line roles_path and make sure it is "roles_path = $HOME/roles

Facts setting

In Ansible you already get a lot of facts by default when you run a playbook or an ad-hoc command against a host. By running `ansible <hostname> -m setup` you can get a list of what facts you would get by default and their values for this host.
If you want to define your own facts, there are 2 possibilities. The first one is capturing output and results from commands you ran previously. The second one is to set them using the set_fact module.

In the below example you will be shown how to do both.

To capture output, you can use register and the name of the variable you want to use to capture the output. You can then use this created variable to display it or use it as input for something else.

facts.yml

```
---
- hosts: testhosts
  vars:
  tasks:
  - name: Install telnet
    yum:
      name: telnet
      state: installed
    register: output
  - debug:
      msg: "Output of yum command {{ output }}"
  - set_fact:
      username: <your name>
      calc_fact: "{{ 400 % 6 }}"
  - debug:
      msg: "Your set username is {{ username }} and
the calculated modulo is {{ calc_fact }}"
```

The entire playbook can be found at https://github.com/wardderick/ansible-advanced/blob/master/facts.yml

Error handling

One way of doing error handling in Ansible is via the usage of blocks.

The tasks in the **block** would execute normally, if there is any error the **rescue** section would get executed with whatever you need to do to recover from the previous error.

The **always** section runs no matter whether the previous error did or did not occur in the block and rescue sections. It should be noted that the play continues if a rescue section completes successfully as it "erases" the error status (but not the reporting). This means it won't trigger max_fail_percentage nor any_errors_fatal configurations, but will appear in the playbooks statistics.

It should also be noted that both the rescue and always sections are optional and you can have a block without them. Below is an example of a block with all 3 sections defined.

```
name: Attempt and graceful roll back demo
  block:
    - debug:
        msg: 'I execute normally'
    - name: I force a failure
      command: /bin/false
    - debug:
        msg: 'I never execute, due to the above task
              failing'
  rescue:
    - debug:
        msg: 'I caught an error'
    - name: I force a failure in middle of recovery
      command: /bin/false
    - debug:
        msg: 'I also never execute'
  always:
    - debug:
        msg: 'This always executes'
```

It's also possible to ignore any errors a task may return and just continue with the next task. To do so you add ignore_errors: yes to your task.

```
- name: this will not be counted as a failure
  command: /bin/false
  ignore_errors: yes
```

To see what you have learned so far on error handling in action, you can run the playbook found on https://github.com/wardderick/ansible-advanced/blob/master/block.yml

By default, when a task fails and previously notified handlers will not be run. You want to change this. There are three possible ways of doing this.

1. adding the --force-handlers command line option
2. adding force_handlers: True inside your play
3. adding force_handlers = True in ansible.cfg

You can also define what defines a failure of a task. Below an example on how to fail a task based on output and one based on return code.

```
- name: Fail task when the command error output \
     prints FAILED
  command: /usr/bin/example-command -x -y -z
  register: command_result
  failed_when: "'FAILED' in command_result.stderr"

- name: Fail task when both files are identical
  raw: diff foo/file1 bar/file2
  register: diff_cmd
  failed_when: diff_cmd.rc == 0 or diff_cmd.rc >= 2
```

 Further learning:
https://docs.ansible.com/ansible/latest/user_guide/playbooks_error_handling.html

Templating and filters

With the combination of filters and templates you can for example create a file that is filled in using variables and where some calculations and verifications are made on some variables.

In the below example you will create an index.html file that contains some information about the server it is located on. The final result of the template will look like this:

```
My hostname is: testvm.localdomain
I have 1 cores and 1 threads per core
I have a total of "0.9677734375" GB of RAM
My owner is unknown
My operating system is: RedHat 7 x86_64
My IP address is 192.168.1.47
sh is a link to /usr/bin/bash
```

If you want to try to solve this without walkthrough, here are some references that can get you started:

- http://www.mydailytutorials.com/ansible-arithmetic-operations
- https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html
- https://docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html

Line 3 should contain a calculation. Line 4 sets a default value of unknown. Line 6 should have a validation for a valid IPv4 address and line 7 a lookup of a link.

To check if your template is correct, you can use the playbook at https://github.com/wardderick/ansible-advanced/blob/master/httpd.yml to test. The solution can be found at https://github.com/wardderick/ansible-advanced/blob/master/index.html.j2

If you want to follow the walkthrough, skip to the next page.

To get started, create the template file:

```
cd /etc/ansible
vi index.html.j2
```

The first 2 lines of the template are just plain text and some variables found in the output of ansible -m setup <hostname>. The double curly brackets are so that your template knows what's between them is a variable.

index.html.j2

```
My hostname is: {{ ansible_nodename }}
I have {{ ansible_processor_cores}} cores \
and {{ ansible_processor_threads_per_core }} \
threads per core
```

To make the 3rd line display the amount of RAM of your server a small calculation is needed as the standard fact about RAM is set in MB.

index.html.j2

```
I have a total of "{{ ansible_memtotal_mb / 1024 }}" \
GB of RAM
```

On the 4th line we will display the owner of the server, and in case the referenced variable is empty default it to unknown.

index.html.j2

```
My owner is {{ username || default('unknown') }}
```

The fifth line is again just plain text and facts from the setup module and checks if the IP is returned is a valid IPv4 address.

index.html.j2

```
My operating system is: \
{{ ansible_distribution_file_variety }} \
{{ ansible_distribution_major_version }} \
{{ ansible_architecture }}
```

The 6th line gets the standard IPv4 address from the setup module and checks if the IP returned is a valid IPv4 address.

index.html.j2

```
My IP address is \
{{ ansible_default_ipv4.address | ipv4 }}
```

And the last line gets the file referenced by the /bin/sh link

index.html.j2

```
sh is a link to {{ 'bin/sh' | realpath }}
```

To put this template on a server you can use the playbook at https://github.com/wardderick/ansible-advanced/blob/master/httpd.yml

Nesting

Now that you learned about templating, guess what this does:

```
- name: Guess what this does?
    azure_rm_networkinterface:
      name: "{{ item[0] }}-{{ item[1] }}-2"
      resource_group: "{{ resource_group_name }}"
      virtual_network: "{{ virtual_network_name }}{{
item[1] }}"
      subnet_name: "{{ subnet_name }}2{{ item[1] }}"
      subscription_id: "{{ subscription_id }}"
      secret: "{{ secret }}"
      tenant: "{{ tenant }}"
      client_id: "{{ client_id }}"
      ip_configurations:
        - name: ipconfig1
          primary: no
    with_nested:
      - [ 'rhv1', 'rhv2', 'rhv3' ]
      - "{{ range(1, nr_of_participants +1) | list
}}"
```

Thank you

We hope you got a good sense of the basics (and some more advanced concepts) around Ansible. We do invite you to continue learning Ansible. Here are some great resources you can use to further develop your Ansible skills:

Quick Start Video: https://www.ansible.com/quick-start-video

Ansible Documentation: https://docs.ansible.com/

Ansible Galaxy: https://galaxy.ansible.com/

Integrating Satellite 6 and Ansible Tower:
https://www.youtube.com/watch?v=SXhfgJjeaGc

Happy automating (with Ansible. What else?)