Ansible Workshop - Exercises

Projects

Use your Ansible skills to complete a couple of small projects.

Project - AWS Automation

Automating Cloud infrastructure is getting more and more important. Tools like *Terraform* are well suited for provisioning infrastructure in public cloud environments.
When dealing with immutable infrastructure, Terraform works well and is great at provisioning cloud resources and applications for AWS, Azure, Docker, GCP, and others. However, there is more to IT operations than automated infrastructure provisioning and this is why Ansible is extremely popular as well.

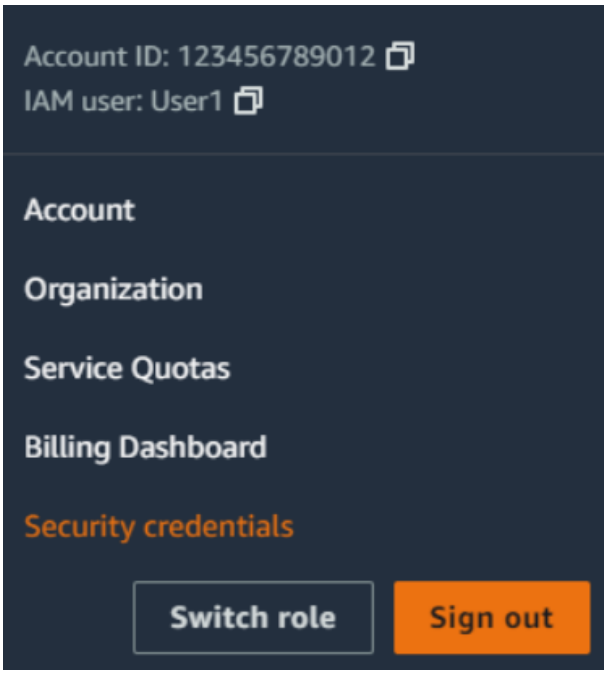
Terraform is an excellent cloud provisioning and de-provisioning tool for infrastructure as code. Ansible is a great all-purpose, cross-domain automation solution.
Together, they perform in harmony to create a better experience for developers and operations teams. Still, this workshops focus is Ansible, let us do provisioning **and** configuration in the cloud with the tool we learned.

Objective

Get to know cloud automation with Ansible.
Cloud automation uses the basic Ansible concepts, but there are some differences in how the modules work.
From a user's point of view, cloud modules work like any other modules. They work with ad hoc commands, playbooks, and roles. Behind the scenes, however, cloud modules use a different methodology than the other (Linux/Unix and Windows) modules use. As we are communicating with an API endpoint, but Ansible and most of its modules are written and executed in Python, you need to use the Python interpreter on the Ansible control node.

Requirements

For doing the following exercises, you will need an AWS Account and an AWS Access key (consists of an *Access Key ID* and a *Secret Access Key*).



After creating your AWS Account, go to the navigation bar on the upper right, choose your user name, and then choose *Security credentials*.
In the Access keys section, choose Create access key. On the Access key best practices & alternatives page, choose your use case to learn about additional options which can help you avoid creating a long-term access key. Mark the checkbox and click *Create Access Key*. On the *Retrieve access keys* page, choose either Show to reveal the value of your user's secret access key, or Download .csv file. This is your only opportunity to save your secret access key. After you've saved your secret access key in a secure location, choose Done.

Tip

Most modules need the region set, use the region `eu-central-1` throughout your playbook.

Guide

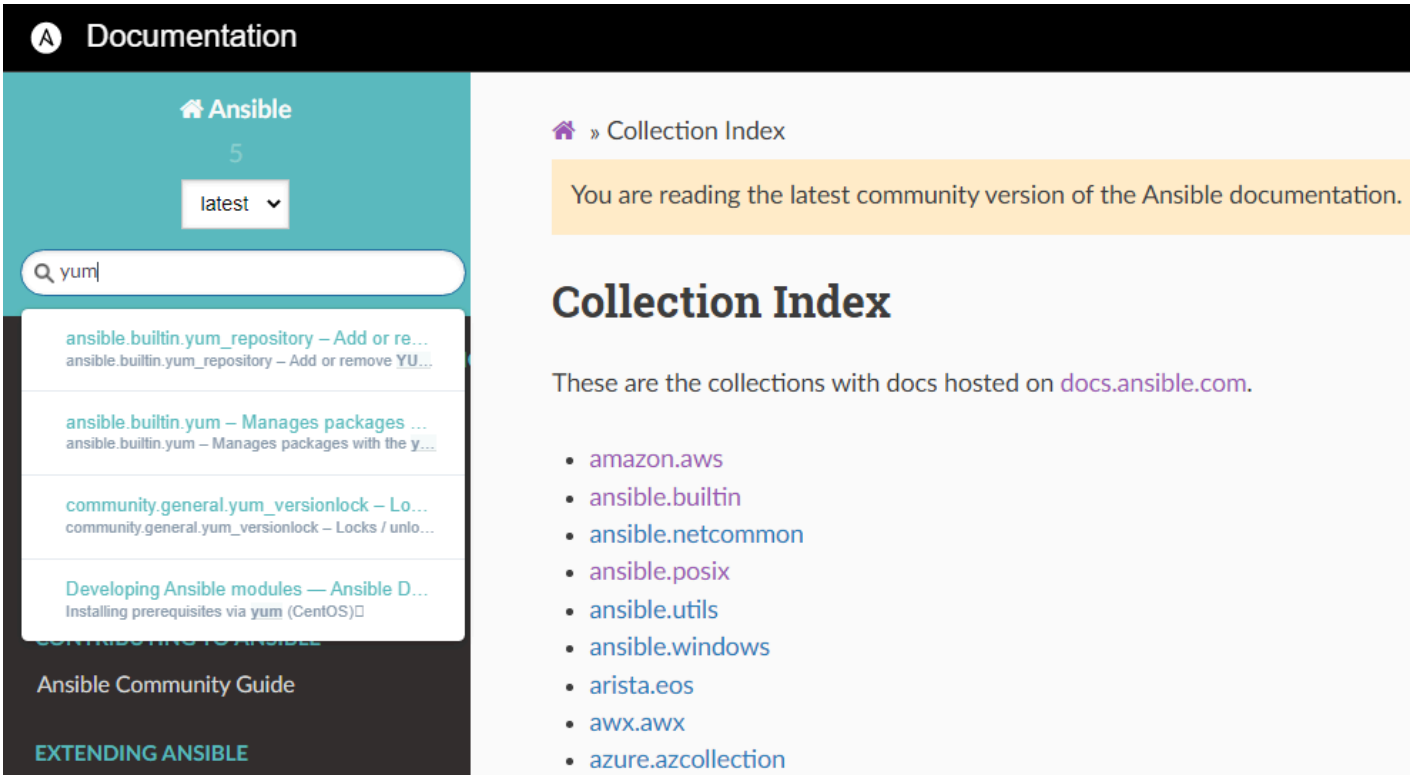
The following steps explain and train you how to use the modules and inventory scripts to automate your AWS resources with Ansible.

Step 1 - Prepare controller

Today, you will need additional Ansible modules. In the first part of the workshop, we only used a handful of modules which are all included in the `ansible-core` binary. With *ansible-core* only 69 of the most used modules are included:

```
[student@ansible-1 ~]$ ansible-doc -l
add_host               Add a host (and alternatively a group) to the ansible-playbook in-memory
inventory
apt                    Manages apt-packages
apt_key                Add or remove an apt key
apt_repository         Add and remove APT repositories
assemble               Assemble configuration files from fragments
assert                 Asserts given expressions are true
async_status           Obtain status of asynchronous task
blockinfile            Insert/update/remove a text block surrounded by marker lines
command                Execute commands on targets
copy                   Copy files to remote locations
...
```

Additional modules are installed through *collections*, search the Collection Index in the Ansible documentation for a suitable collection or use the search field.



Once you found the appropriate collection, install it with the `ansible-galaxy` CLI command:

```
ansible-galaxy collection install provider.collection
```

Requirements for the AWS modules are minimal, you will need an additional Python package. Install the package with this command:

```
pip3.9 install boto3 --user
```

Note

Note the version of the Python package manager utility (`pip3.9`)!
Your Ansible control node might have multiple Python versions installed, install necessary dependencies for the Python version that Ansible uses.
You can check for the Python interpreter of Ansible with the `ansible --version` command:

```
[student@ansible-1 ~]$ ansible --version
ansible [core 2.14.0]
config file = /etc/ansible/ansible.cfg
configured module search path = ['/home/student/.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
ansible python module location = /usr/lib/python3.9/site-packages/ansible
ansible collection location = /home/student/.ansible/collections:/usr/share/ansible/collections
executable location = /usr/bin/ansible*
python version = 3.9.13 (main, Nov  9 2022, 13:16:24) [GCC 8.5.0 20210514 (Red Hat 8.5.0-15)]
(/usr/bin/python3.9)
jinja version = 3.1.2
libyaml = True
```

Do not use *sudo* when installing Python packages. If you get a *Permission denied*, add `--user` , this installs the dependencies to `~/.local/lib` .

Achieve the following tasks:

- Find appropriate collection for AWS automation in the documentation
- Collection installed
- Python requirements installed

You can view the installed collections with this command:

```
[student@ansible-1 aci-automation]$ ansible-galaxy collection list
# /home/student/.ansible/collections/ansible_collections
Collection        Version
----------------- -------
ansible.posix     1.4.0
community.docker  2.7.0
community.general 5.3.0
```

Step 2 - Prepare project

Create a new project folder in your home directory:

```
[student@ansible-1 ~]$ mkdir aws-automation
```

Within your newly created project folder, create a playbook file.

Tip

You have to instruct Ansible to communicate with the AWS API, per default Ansible would try to communicate via SSH. This will not work. Set the target of your playbook to your *local machine*.

The documentation provides an extensive Guide for AWS automation which can help you setting up everything. For successful communication with the AWS API, you need to authenticate yourself, this is where your previously created Access key is needed.

You can either specify your credentials as module arguments (you'll need to repeat them with **every** module) or as *environment variables*. The first variant would require you to set the credentials in variables (which need to be encrypted, this can be achieved with *ansible-vault*). Let's use the method with environment variables, this eases the first steps and is also applicable if you would run your playbook in the Ansible Automation Platform.

Set the environment variables on the CLI:

```
export AWS_ACCESS_KEY_ID='AK123'
export AWS_SECRET_ACCESS_KEY='abc123'
```

Warning

Environment variables are only set in the current session, if you close your terminal, you'll need to set them again.

To remember setting the variables, you could include this optional task as the first in your playbook which asserts that the variables are set. If the variables are missing, it will fail the playbook with a hint on what to do:

```
- name: Ensure AWS credentials are set
  ansible.builtin.assert:
  that:
    - ansible_env.AWS_ACCESS_KEY_ID is defined
    - ansible_env.AWS_SECRET_ACCESS_KEY is defined
  quiet: true
  fail_msg: |
    No environment variables with AWS credentials found!
    Set the variables with:
      export AWS_ACCESS_KEY_ID='AK123'
      export AWS_SECRET_ACCESS_KEY='abc123'
```

Danger

Your bash history reveals the CLI input and your credentials!
You can delete the respective entry with `history -d <position>`.

Alternative solution

You can set your credentials in a hidden file `~/.aws/credentials` in your home directory in an *ini* file:

```
[workshop]
aws_access_key_id = YOUR_AWS_ACCESS_KEY_ID
aws_secret_access_key = YOUR_AWS_SECRET_ACCESS_KEY
```

The *section* represents a credential profile which needs to be added to every module with the key-value-pair `aws_profile: profile_name`, in our example with `aws_profile: workshop`.

Note, this solution also does not store the credentials in an encrypted way! Everybody with access to your home directory would be able to read your credentials!
In production, its best to use an external credential provider. In the Ansible Automation platform you can store your variables in an encrypted database or use multiple credential provider plugins.

Testing the successful communication with the API could be done by querying information about an EC2 AMI Image. Find an appropriate module, create your playbook and add a task. Try to gather information about the following AMI, you can copy the content with a button:

```
ami-06c39ed6b42908a36
```

The AMI is available in the *eu-central-1* region, you may to define this in the module you've chosen.

Run your playbook, if it returns a green *ok* status, communication is established. For now, the gathered information about the AMI is not relevant for us, still, you could store the output in a variable and output it with an appropriate module, if you are curious.

Achieve the following tasks:

- Playbook created
- Successful communication with AWS established

Step 3 - Create SSH key-pair

In a later step, we will create EC2 instances. To be able to login to these hosts, we need a SSH key-pair. Let's create a dedicated key, this can be achieved with the module `openssh_keypair`. The module is not part of the *ansible.builtin* collection, try to find the collection where the module is stored (Tip: Use the search field in the documentation). When you found the correct collection, install it with the `ansible-galaxy collection install` command.

Add a task to your playbook which creates a key-pair in the default folder in your home directory (`~/.ssh`). The key should be called `workshop`, the module will create a private key with this name and a public key with the name `workshop.pub`. The home directory of the user running the playbook is stored in the fact `ansible_env.HOME`, use this as a variable and append `/.ssh/workshop`.

Success

Use a key size of **2048 bits**!

Now, lets create the EC2 key-pair named `workshop` in AWS with our playbook.
Find the correct module and provide the **public** key created by the previous task.
You can access the content of the public key with a *lookup* plugin:

```
"{{ lookup('file', ansible_env.HOME + '/.ssh/workshop.pub') }}"
```

Achieve the following tasks:

- Collection with module `openssh_keypair` found and installed
- Added task to create key pair with 2048 bits
- Added task to create new AWS EC2 keypair using public key of previously created local keypair

Step 4 - Get default VPC

A AWS Virtual Private Cloud should already be configured for you, lets use this for our workshop. We need to get the ID of the default VPC net, this can be achieved with Ansible as well.
Find the correct module to gather information about EC2 VPCs and add it to your playbook. Add the following parameters:

```
region: eu-central-1
filters:
  "is-default": true
```

Store the output of the module in a variable, e.g. `vpc_info`. Afterwards, add the following task which sets a fact/variable with the ID of your default VPC:

```
- name: Set variable with ID of default VPC
  set_fact:
    default_vpc_id: "{{ vpc_info.vpcs.0.vpc_id }}"
```

The variable `vpc_info` contains a list `vpcs`. As we filtered for the default VPC, the list only contains one element, therefore we can access the list item with `0`. The list item contains a key `vpc_id`, the value is what we are looking for.

Achieve the following tasks:

- Module for gathering VPC info identified and used
- *set_fact* Task returns green "ok" status

If you are curious, add another task which debugs the variable to *stdout*.

Step 5 - Create Security group

We need to create a security group and add a rule for incoming SSH access to be able to login to our EC2 instance later. Find the correct module and add a task, provide the following parameter:

Parameter	Value	Description

name	*workshop-sg*	The name of the Security group
description	*Security group created by Ansible*	Short description
vpc_id	*"{{ default_vpc_id }}"*	The value of your variable `default_vpc_id`
region	*eu-central-1*	The region we used in all other tasks

The `rules` parameter must hold a list, in our case a single rule is enough. Find the correct rule parameters and use the following values:

- Protocol: *TCP*
- From: *22*
- To: *22*
- CIDR: *0.0.0.0/0*

Run your playbook.

Achieve the following tasks:

- Module for maintaining security groups identified and used
- Security group successfully created

Step 6 - Create EC2 instance

Now it's finally time to create a virtual machine in AWS.
Find the appropriate module and add a task to your playbook, your instance should have the following configuration (this time it is up to you to find the correct key-value-pairs):

- Must be called `workshop-instance1`
- Must be created in `eu-central-1`
- Must have a public IP address
- Must have the `workshop` key assigned
- Must have the size `t2.micro`
- Must be in the security group `workshop-sg`
- Must use the AMI `ami-06c39ed6b42908a36`
- Should have the tag `Environment: Testing` attached

Choose the right value for the `state` parameter, your playbook should wait for a *running* instance!

Achieve the following task:

- Running EC2 instance

Step 7 - Get DNS name and login

Find a module to gather information about your EC2 instances in your region, use the filter `"tag:Name": workshop-instance1` to only get this single instance.
Store the output of the module into a variable and use the variable in another task which *debugs* only the *public DNS name* of your previously created EC2 instance.

```
TASK [Output public DNS name of workshop-instance1]
*******************************************************************************************
***************
ok: [localhost] =>
  msg: ec2-3-70-238-39.eu-central-1.compute.amazonaws.com
```

Copy the output of your task and login to your EC2 instance with SSH. Provide the *private* key and use the user `ec2-user`, for example:

```
[student@ansible-1 ~]$ ssh -i ~/.ssh/workshop ec2-user@ec2-3-70-238-39.eu-central-
1.compute.amazonaws.com
Last login: Sat Feb 11 13:27:56 2023 from ec2-3-71-15-149.eu-central-1.compute.amazonaws.com

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
16 package(s) needed for security, out of 16 available
Run "sudo yum update" to apply all updates.
```

Achieve the following tasks:

- Added task to gather information about EC2 instances
- Added task to output public DNS name of instance
- Successful SSH login to EC2 instance

Success

Awesome, you created a virtual machine in the Cloud and are able to login!

Optional

Step 1 - Create multiple EC2 instances

In *Step 5* you created a single EC2 instance, adjust your task to create multiple instances in a loop. The *name* of every instance must differ, as well as the `Environment` tag.
Create three instances with the single task, with the instance being in the given *Environment* :

Name	Environment (Tag)
workshop-instance1	*Testing*

workshop-instance2	*Testing*
workshop-instance3	*Production*

Run your playbook, you should see two more instances being created.

Achieve the following task:

- Adjusted task to create three EC2 instances `workshop-instance[1-3]`

Step 2 - Create dynamic inventory

When using Ansible with AWS, inventory file maintenance will be a hectic task as AWS frequently changes IPs, autoscaling instances, and more. Once your AWS EC2 hosts are spun up, you'll probably want to talk to them again. With a cloud setup, it's best not to maintain a static list of cloud hostnames in text files. Rather, the best way to handle this is to use the `aws_ec2` dynamic inventory plugin.

Create a file `workshop.aws_ec2.yml`

The inventory should have two additional groups `test_stage` and `prod_stage`. The hosts have a tag `Environment` with either `Testing` or `Production`, ensure that they are part of the correct group.
You can test your inventory with the `ansible-inventory` CLI utility, it outputs a JSON representation of how Ansible sees your provided inventory.

```
[student@ansible-1 aws-automation]$ ansible-inventory -i demo.aws_ec2.yml --list
{
    "_meta": {
        [..Cut for better readability..]
    }
    "all": {
        "children": [
            "aws_ec2",
            "prod_stage",
            "test_stage",
            "ungrouped"
        ]
    },
    "aws_ec2": {
        "hosts": [
            "ec2-18-185-94-35.eu-central-1.compute.amazonaws.com",
            "ec2-3-126-92-75.eu-central-1.compute.amazonaws.com",
            "ec2-3-70-238-39.eu-central-1.compute.amazonaws.com"
        ]
    },
    "prod_stage": {
        "hosts": [
            "ec2-18-185-94-35.eu-central-1.compute.amazonaws.com"
        ]
    },
    "test_stage": {
        "hosts": [
            "ec2-3-126-92-75.eu-central-1.compute.amazonaws.com",
            "ec2-3-70-238-39.eu-central-1.compute.amazonaws.com"
        ]
    }
}
```

You need to set some Ansible connection variables, remember, direct SSH connection also only worked when providing the SSH private key and the target user.

Tip

The documentation has a typo, the variable for the SSH private key file is **not** `ansible_private_ssh_key_file` but `ansible_ssh_private_key_file`!

When you finished your inventory, use this playbook to test the connection:

test-connection.yml

```yaml
---
- name: Playbook targeting hosts from dynamic inventory
  hosts: test_stage
  tasks:
    - name: Try to reach hosts
      ansible.builtin.ping:
```

Running the playbook (and providing the inventory!) results in the following output:

```
[student@ansible-1 aws-automation]$ ansible-playbook -i workshop.aws_ec2.yml test.yml

PLAY [Playbook targeting hosts from dynamic inventory]
*******************************************************************************************
**************

TASK [Gathering Facts]
*******************************************************************************************
*******************************************
ok: [ec2-3-70-238-39.eu-central-1.compute.amazonaws.com]
ok: [ec2-3-126-92-75.eu-central-1.compute.amazonaws.com]

TASK [Try to reach hosts]
*******************************************************************************************
*****************************************
ok: [ec2-3-126-92-75.eu-central-1.compute.amazonaws.com]
ok: [ec2-3-70-238-39.eu-central-1.compute.amazonaws.com]

PLAY RECAP
*******************************************************************************************
*********************************************************
ec2-3-126-92-75.eu-central-1.compute.amazonaws.com : ok=2    changed=0    unreachable=0
failed=0    skipped=0    rescued=0    ignored=0
ec2-3-70-238-39.eu-central-1.compute.amazonaws.com : ok=2    changed=0    unreachable=0
failed=0    skipped=0    rescued=0    ignored=0
```

Help wanted?

Authentication is done here with the credentials stored in `~/.aws/credentials`

```
# demo.aws_ec2.yml
plugin: amazon.aws.aws_ec2

aws_profile: workshop

regions:
  - eu-central-1

groups:
  test_stage: "'Testing' in tags.Environment"
  prod_stage: "'Production' in tags.Environment"

filters:
  instance-state-name: running

compose:
  ansible_host: public_dns_name
  ansible_ssh_private_key_file: ~/.ssh/workshop
  ansible_user: ec2-user
```

Cleanup

Warning

When you are done, remember to **clean up all created resources** in AWS to prevent incurring costs!

You created the following resources in AWS:

- EC2 Instance(s)
- Security Group
- SSH Keypair