4/26/2023

Caldera Automation

How to automate Caldera deployment using AWS

Jacob Levenant, Candan Martin, Marwin Kralemann

CIS 4355 CLOUD COMPUTING
SPRING 2023

EXECUTIVE SUMMARY

Objective

The main objective of this project is to automate the deployment and management of the Caldera platform on AWS EC2 instances using Ansible and CloudFormation, ultimately improving the efficiency, maintainability, and security of the system.

Background

Caldera is a powerful cybersecurity platform designed to simulate adversarial tactics, techniques, and procedures. By automating its deployment and management on AWS EC2 instances, we can streamline the process, minimize manual intervention, and enhance overall security.

Methodology

- Develop Ansible scripts and CloudFormation templates to automate the deployment of web servers and the provisioning of EC2 instances.
- Create an Ansible playbook to install Caldera on a VM in AWS.
- Enhance security by designing a configuration file template for Caldera using Ansible, incorporating user credentials.
- Implement a dynamic inventory-enabled Ansible server for efficient Caldera deployment and management.
- Leverage CloudFormation and Ansible to automate Caldera deployment on created EC2 instances.

Key Findings

- Automated deployment and management of Caldera on AWS EC2 instances resulted in faster and more efficient operations.
- Improved security configurations for Caldera through the use of Ansible-managed user credentials.
- Ansible server with dynamic inventory enabled better control and tracking of Caldera deployments across various instances.
- Integration of CloudFormation and Ansible scripts streamlined the entire process, reducing manual intervention and human error.

Recommendations

- Ensure that the Ansible and CloudFormation scripts are kept up-to-date with the latest changes in the Caldera platform and AWS environment.
- Prioritize security when designing and deploying solutions by incorporating secure configuration management techniques.
- Leverage dynamic inventory and automated processes to manage deployments more effectively and ensure consistency across instances.

Cloud Architecture

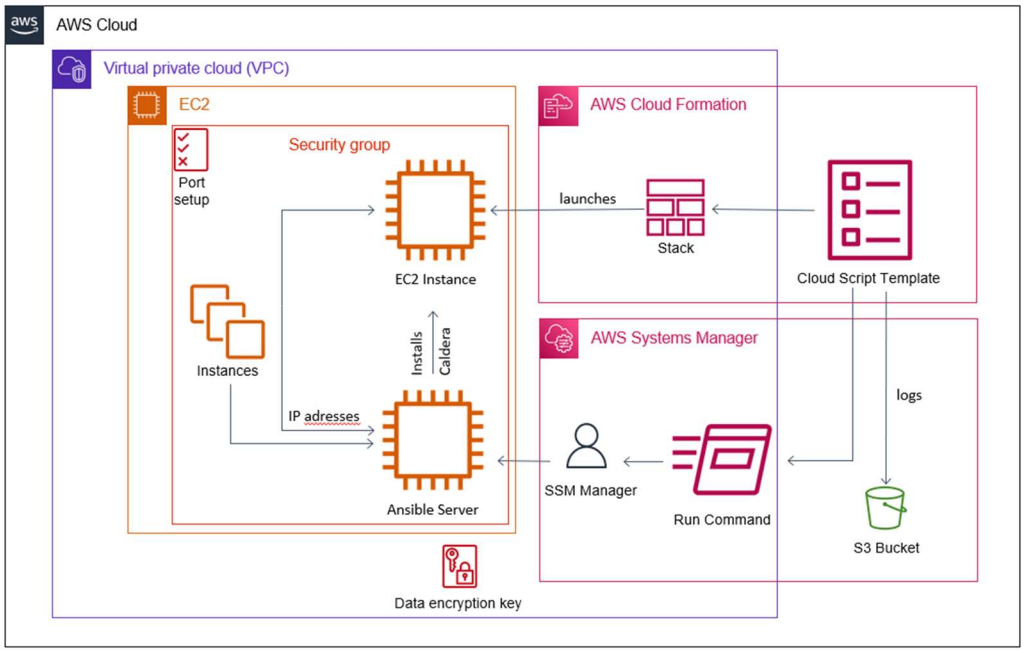

Figure 1: Project Architecture

Conclusion

Through the successful integration of Ansible and CloudFormation, we have effectively automated the deployment and management of Caldera on AWS EC2 instances. This approach not only streamlines the process but also enhances the overall security and maintainability of the system. By combining these methodologies, we have created a more efficient, secure, and maintainable infrastructure that paves the way for improved management and monitoring of cybersecurity tools in cloud environments.

Project Milestones

Milestone 1

- Create Ansible script to launch a web server & create cloud formation script to launch ec2 instances

Milestone 2

- Develop an Ansible playbook to install Caldera on a VM in AWS

Milestone 3

- Create a configuration file template for Caldera using Ansible to improve security by adding user credentials.

Milestone 4

- Set up an Ansible server with dynamic inventory to manage the deployment of Caldera.

Milestone 5

- Set up cloud formation script that will create ec2 instances and run the caldera-install ansible script on the ansible server to deploy caldera on the created ec2 instances

Materials List:

1. AWS
2. Visual Studio Code
3. Terminal/CLI Interfaces

Deliverables:

1. Report
2. Deployed Architecture
3. Scripts

Professional Accomplishments:

1. Learn and use Cloud Formation
2. Learn and use Ansible
3. Set up and configure Caldera

PROJECT SCHEDULE MANAGEMENT

Gantt Chart

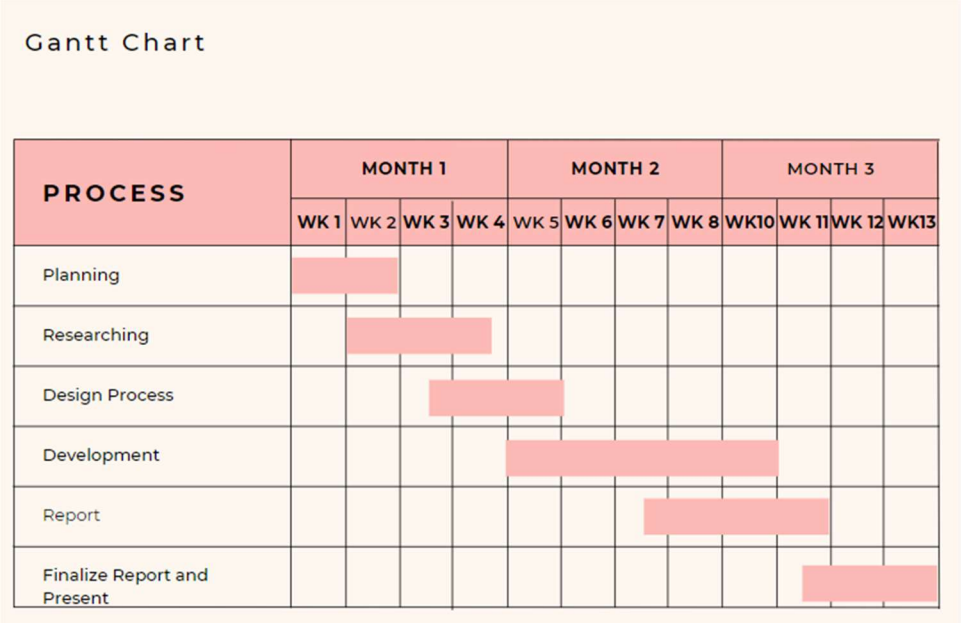

Figure 2: Gantt Chart

Trello Board

Figure 3: Trello QR Code

GitHub Project Repository
/ccmarti1/CSS-CLOUD-PROJECT (github.com)

Milestone 1: Intro to Ansible and Cloud Formation

Set up Ansible on local Machine

The first step to take was to set up Ansible on a local machine to start developing scripts and running them. Ansible is developed to run on machines with a unix kernel, but the local machine used to create and run the scripts was running Windows. In order to get a working Ansible installed, a program called Cygwin was used (Fig. 4). Cygwin is a console application that allows native Linux apps to run on Windows, which includes ansible. The guide to the Cygwin and ansible installation can be found here:
How to Install and Configure Ansible on Windows {3 Methods Explained} (phoenixnap.com)

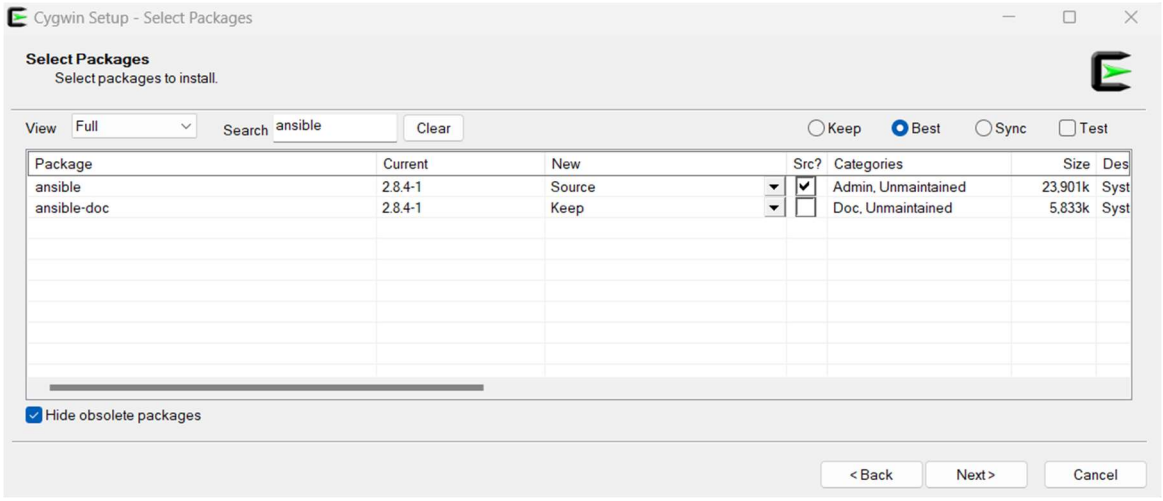

Figure 4: Cygwin packages install

It is very important that the ansible packages are selected as shown above during the installation.

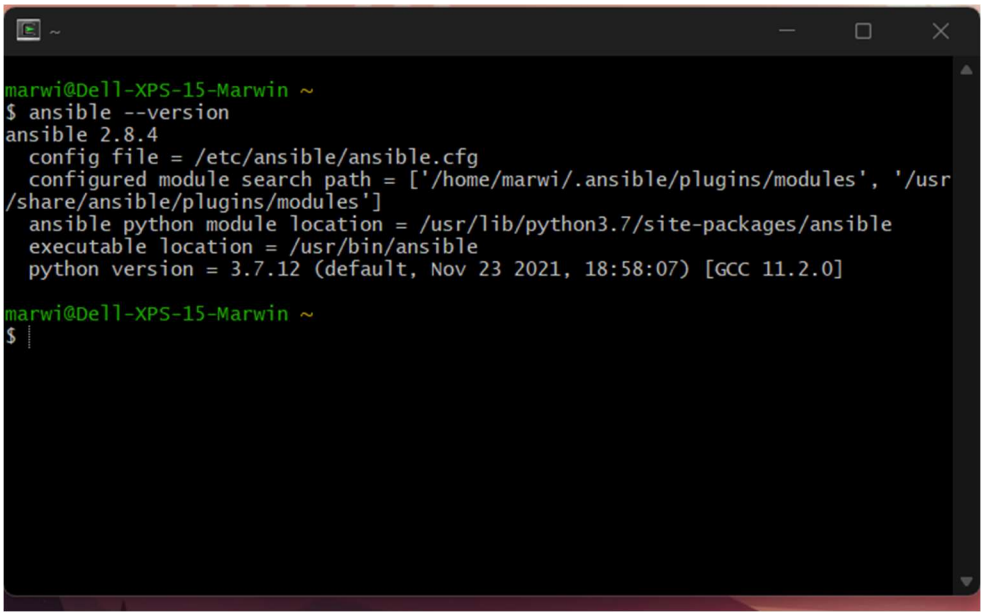

Figure 5: Cygwin console with ansible installed

This is the cygwin console (Fig. 5) and as can be seen ansible was installed correctly. In order for ansible to work the best way with aws a couple of files need to be changed. Firstly, the ansible.cfg file in the /etc/ansible folder needed to be changed.

The default values (Fig. 6) need to be changed to the following:

```
[defaults]

host_key_checking = false
remote_user = ec2-user
ask_pass = false
private_key_file = ../../home/marwi/Cloud-Computing-Project/Ansible/ansible.pem
roles_path = ../../home/marwi/Cloud-Computing-Project/Ansible/roles
```

Figure 6: Ansible.cfg file default values

The remote_user needs to be ec2-user as that allows the connection to the instances. Then, a private key file needs to be provided that is set up for access to the ec2 instances with said key configured. That is why we created a specific ansible.pem key file only for the purpose of use in ansible.

Next, the privilege escalation (Fig. 7) needs to be changed to the following:

```
[privilege_escalation]
become=True
become_method=sudo
become_user=root
become_ask_pass=False
```

Figure 7: Ansible.cfg file privilege escalation

The method needs to be set to sudo, so whenever ansible performs any task, it is done with superuser privileges. The user needs to be root for the same reasons.

Lastly, the host file needs to be modified and the public IP addresses of the instances that want to be addressed by the playbooks need to be inputted (Fig. 8).

```
[webservers]
# 54.197.129.83
# 100.25.149.138
# 100.25.157.170
# 44.203.7.250
35.168.7.222
```

Figure 8: Host file webservers group

Create Ansible Playbook to test SSH connection to

The next step was to create an ansible playbook to test the connection to the ec2 instances listed in the host file (Fig. 9). The script is very simple and looks as follows:

```
1  - name: Test SSH connection to EC2 instance
2    hosts: webservers
3    gather_facts: no
4    tasks:
5      - name: Ping the EC2 instance
6        ping:
```

Figure 9: Test-SSH-Connection playbook script

The host part declares which group is going to be addressed in the host files. The tasks are what is being performed on the specified instance or instances. In this case that means pinging the EC2 instane.

In order to run the script the following command needs to be run.

```
$ ansible-playbook -u ec2-user -b test-ssh-connection.yaml --private-key=ansible.pem
```

Figure 10: Ansible-playbook test-ssh-connection command

Ansible-playbook specifies that a playbook will be run. The -u part specifies the user that will be used to perform the tasks. The -b part specifies the playbook and the –private-key part specifies the private key necessary for the connection to the instance.
Executing the script will yield the following result if everything is setup correctly.

```
ec2-54-204-250-152.compute-1.amazonaws.com : ok=1    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
 ignored=0
```

Figure 11: Successful SSH test playbook execution

Apache Server Ansible Script

As a first script that would perform tasks on an ec2 instance we created a script that mimicked the task from the class to launch an httpd apache server on it which will output the IP address as a text.
To accomplish this the following script was created:

C: > Users > marwi > OneDrive > Dokumente > Cloud-Computing > Cloud-Computing-Project > Ansible > install-apache-server.yaml

```yaml
- name: Install hhpd server, create index.html file, and start httpd
  hosts: webservers
  become: true

  tasks:
    - name: install httpd
      package:
        name: httpd
        state: present

    - name: Create index.html file
      copy:
        content: "<h1>Hello World from $(hostname -f)</h1>"
        dest: /var/www/html/index.html

    - name: start httpd
      service:
          name: httpd
          state: started
```

Figure 12: Apache Server install ansible playbook

The first task is to install httpd, which will then install the httpd apache server. The second task creates the index file with the required text in the “content” variable and the file destination in the “dest” field. Lastly, the server needed to be started so the task for starting the httpd service was created.

Cloud Formation

To get familiar with Cloud Formation, we started off by creating a template that will create an EC2 instance with an Apache web server installed.

```
Resources:
  EC2I1V5E2:
    Type: 'AWS::EC2::Instance'
    Properties:
      InstanceType: t2.micro
      ImageId: ami-0dfcb1ef8550277af
      KeyName: ansible
      UserData:
        'Fn::Base64': !Sub |
          #!/bin/bash
          yum update -y
          yum install httpd -y
          systemctl start httpd
          systemctl enable httpd
          echo "Hello World" > /var/www/html/index.html
      SecurityGroups:
        - Ref: EC2InstanceSecurityGroup
    Metadata:
  EC2InstanceSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable SSH and HTTP access
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
```

Figure 13: Apache Server Cloud Formation Script

Milestone 2: Caldera Deployment Playbook for AWS VMs

Manual Installation:

We initially installed Caldera manually in order to help us understand the process of installing caldera so that we can use our ansible script to automate the process. This involves connecting to the instance via SSH and following the official Caldera installation documentation. The step-by-step process is detailed below.

Created and Connect to the EC2 instance using SSH:

```
>ssh -i /Downloads/ansible.pem ec2-user@ec2-12-34-56-78.compute-1.amazonaws.com
```

Figure 14: SSH connection to ec2 instance

Install Dependencies:

We installed the following dependencies:

- Python 3.6 or later
- Pip for Python 3
- Git

Clone the Repository:

Clone the CALDERA repository from GitHub using the following command:

```
git clone https://github.com/mitre/caldera.git --recursive --branch 3.0.0
```

Figure 15: Clone git repository command

This command clones the CALDERA repository and its submodules, targeting the 3.0.0 branch.

Install Python Requirements:

Navigate to the cloned CALDERA directory and install the required Python packages using the following command:

```
pip3 install -r requirements.txt
```

Figure 16: Pip3 install command

Start the CALDERA Server:

Ran the CALDERA server using the following command:

```
>ansible-playbook -u ec2-user -b install-caldera.yaml --private-key=ansible.pem
```

Figure 17: Start caldera command with

The manual installation process sets the foundation for further automation, enabling seamless deployment and management of Caldera on AWS EC2 instances.

Automating caldera installation on EC2 Instance

This Ansible script is designed to automate the installation and deployment of Caldera on an EC2 instance. The script is divided into four main tasks, each performing a specific step in the installation process.

Install Dependencies:

The first task in the script is to install the necessary dependencies for Caldera (Fig. 18). The **`dnf`** module is used to manage packages on the target host, ensuring that Python3, pip3, gcc, openssl-devel, python3-devel, and git-all are installed and up to date. The **`become`** and **`become_method`** directives ensure that the commands are run with elevated privileges (sudo).

```
tasks:
- name: Install dependencies
  dnf:
    name:
      - python3
      - python3-pip
      - gcc
      - openssl-devel
      - python3-devel
      - git-all
    state: present
  become: yes
  become_method: sudo
```

Figure 18: Install dependencies task

Clone Caldera Repository:

The second task clones the Caldera repository from GitHub to the target EC2 instance using the **`git`** module (Fig. 19). The **`recursive`** flag is set to ensure that all submodules are cloned as well. Again, elevated privileges are used with the **`become`** and **`become_method`** directives.

```
- name: Clone Caldera repository
  git:
      repo: https://github.com/mitre/caldera.git
      dest: /home/caldera
      recursive: yes
  become: yes
  become_method: sudo
```

Figure 19: Clone git repository task

Install Caldera Requirements:

The third task (Fig. 20) installs the Python packages required for Caldera by running the **`pip`** module. The requirements are specified in the **`requirements.txt`** file located in the cloned repository. The **`executable`** flag ensures that the pip3 version is used, and elevated privileges are utilized as before.

```
- name: Install Caldera requirements
  pip:
    requirements: /home/caldera/requirements.txt
    executable: pip3
  become: yes
  become_method: sudo
```

Figure 20: Install Caldera requirements task

Start Caldera:

The final task (Fig. 8) starts the Caldera server using the **`shell`** module. The command **`sudo nohup python3 server.py --insecure &`** is used, where **`nohup`** allows the process to continue running in the background even if the terminal is closed, and the **`&`** symbol runs the process asynchronously, preventing it from blocking other tasks. The **`async`** and **`poll`** options ensure that the task is non-blocking and runs in the background.

```
- name: Start Caldera
  shell:
    cmd: sudo nohup python3 server.py --insecure &
    chdir: /home/caldera
  async: 1
  poll: 0
```

Figure 21: Start Caldera task

The script was initially developed and tested by manually installing Caldera on an EC2 instance via SSH, following the Caldera installation documentation. Once the process was fully understood, the steps were automated using the Ansible script, making the deployment process more efficient and maintainable.

Milestone 3: Secure Caldera Configuration with Ansible-Managed Credentials

Modify Configuration File

The given Ansible script comprises four tasks focused on configuring the local.yml file for Caldera. This configuration file is essential for setting up user credentials and managing access to the system.

Create local.yml configuration file:

This task (Fig. 22) copies the default configuration file (default.yml) and creates a new file called local.yml in the /home/caldera/conf directory. This new file will store the customized configuration settings.

```
- name: Create local.yml configuration file
  shell:
    cmd: sudo cp default.yml local.yml
    chdir: /home/caldera/conf
```

Figure 22: Create local yaml file task

Set admin password in local.yml:

This task (Fig. 23) modifies the local.yml file to replace the default 'admin' password with a secure, custom password 'CloudProject2023'. The 'ansible.builtin.replace' module searches for the pattern 'admin: admin' and replaces it with 'admin: CloudProject2023'.

```
- name: Set admin password in local.yml
  ansible.builtin.replace:
    path: /home/caldera/conf/local.yml
    regexp: 'admin: admin'
    replace: 'admin: CloudProject2023'
  become: yes
  become_method: sudo
```

Figure 23: Set admin password task

Set red password in local.yml:

Similar to the previous task, this step changes the 'red' user's password in the local.yml file (Fig. 24). It replaces the default 'red: admin' with the custom password 'red: CloudProject2023Red'.

```
- name: Set red password in local.yml
  ansible.builtin.replace:
    path: /home/caldera/conf/local.yml
    regexp: 'red: admin'
    replace: 'red: CloudProject2023Red'
  become: yes
  become_method: sudo
```

Figure 24: Set red password task

Set blue password in local.yml:

This task also modifies the local.yml file, updating the 'blue' user's password (Fig. 25). It searches for the pattern 'blue: admin' and replaces it with the custom password 'blue: CloudProject2023Blue'.

```
- name: Set blue password in local.yml
  ansible.builtin.replace:
    path: /home/caldera/conf/local.yml
    regexp: 'blue: admin'
    replace: 'blue: CloudProject2023Blue'
  become: yes
  become_method: sudo
```

Figure 25: Set blue password task

Conclusion:

By performing these tasks, the Ansible script ensures that the local.yml configuration file contains updated and secure user credentials, enhancing the overall security of the Caldera deployment.

Milestone 4: Dynamic Inventory-Enabled Ansible Server for Caldera Management

Install Ansible on EC2 Instance

In order to set up an ansible server to use the first step was to create an EC2 instance. Since it is going to host the ansible server that will be used to run the playbooks for all necessary caldera deployments it shouldn't be terminated and thus termination protection was enabled.

Next, an SSH connection to the EC2 instance was used in order to install ansible. The original Ansible documentation was used to install it with the necessary commands. As shown in Fig. 26

```
[ec2-user@ip-172-31-80-23 ~]$ ansible --version
ansible [core 2.14.4]
  config file = None
  configured module search path = ['/home/ec2-user/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /home/ec2-user/.local/lib/python3.9/site-packages/ansible
  ansible collection location = /home/ec2-user/.ansible/collections:/usr/share/ansible/collections
  executable location = /home/ec2-user/.local/bin/ansible
  python version = 3.9.16 (main, Feb 23 2023, 00:00:00) [GCC 11.3.1 20221121 (Red Hat 11.3.1-4)] (/usr/bin/python3)
  jinja version = 3.1.2
  libyaml = True
```

Figure 26: Install ansible on ec2 instance

Since aws-linux uses yum to install ansible it does not install some necessary files right away. Normally, an ansible folder under the root/etc is created with the ansible.cfg and host file. In order to set up ansible properly these files needed to be created manually using the cli. As show in Fig. 27

```
[ec2-user@ip-172-31-80-23 etc]$ sudo mkdir ansible
[ec2-user@ip-172-31-80-23 etc]$ sudo vi ansible.cfg
[ec2-user@ip-172-31-80-23 etc]$ vi ansible.cfg
[ec2-user@ip-172-31-80-23 etc]$ vi hosts
```

Figure 27: Creating directories and config files

The files were then created using vi.

By mistake they were in the wrong directory at first, so they were moved to the correct one.

```
[ec2-user@ip-172-31-80-23 etc]$ sudo mv /etc/ansible.cfg /etc/ansible/ansible.cfg
```

Figure 28: Move files into correct directory

Set Up Dynamic Inventory

The Dynamic Inventory allows the Ip addresses from the ec2 instances to automatically be passed to ansible for any playbooks that are being executed. In order to do that multiple steps are necessary.

```
[ec2-user@ip-172-31-80-23 ansible]$ sudo mkdir inventory
[ec2-user@ip-172-31-80-23 ansible]$ cd inventory
[ec2-user@ip-172-31-80-23 inventory]$ ls -a
.  ..
[ec2-user@ip-172-31-80-23 inventory]$ sudo vi aws_ec2.yaml
```

Figure 29: Creating inventory folder and plugin file

First, I need to create a plugin file that will have access to the ec2 instances and automatically get their public IP addresses (Fig. 30). That is the aws_ec2.yaml file and will be created in a new inventory folder within the /etc/ansible directory (Fig. 29).

```
---
plugin: aws_ec2
aws_access_key: AKIA3NTJREOBUB6BY5UU
aws_secret_key: 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
regions:
  - us-east-1

keyed_groups:
  - key: tags
    prefix: tag
```

Figure 30: Plugin script

This .yaml file specifies that it is the aws_ec2 plugin. It needs an AWS access key to gain access to the ec2 instances. One was created for that specific purpose and provided here. Then the regions of the ec2 instances are identified under the regions part and lastly the instances are going to be grouped by their tags, which will allow playbooks to address specific instances based on specific tags only.
Since it is accessing an AWS service the boto3 plugin for python also needs to be installed, which is done using the following command.

```
[ec2-user@ip-172-31-80-23 ~]$ pip install boto3
Defaulting to user installation because normal site-packages is not writeable
Collecting boto3
  Downloading boto3-1.26.118-py3-none-any.whl (135 kB)
     |████████████████████████████████| 135 kB 6.2 MB/s
Collecting botocore<1.30.0,>=1.29.118
  Downloading botocore-1.29.118-py3-none-any.whl (10.7 MB)
     |████████████████████████████████| 10.7 MB 37.8 MB/s
Collecting s3transfer<0.7.0,>=0.6.0
  Downloading s3transfer-0.6.0-py3-none-any.whl (79 kB)
     |████████████████████████████████| 79 kB 15.7 MB/s
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in /usr/lib/python3.9/site-packages (from boto3) (0.10.0)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/lib/python3.9/site-packages (from botocore<1.30.0,>=1
.29.118->boto3) (2.8.1)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /usr/lib/python3.9/site-packages (from botocore<1.30.0,>=1.29.11
8->boto3) (1.25.10)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages (from python-dateutil<3.0.0,>=2.1->botocore<
1.30.0,>=1.29.118->boto3) (1.15.0)
Installing collected packages: botocore, s3transfer, boto3
Successfully installed boto3-1.26.118 botocore-1.29.118 s3transfer-0.6.0
```

Figure 31: Installing boto3

Lastly, the plugin needs to be enabled (Fig. 32 & 33) and the plugin file needs to be set as the default host directory in the ansible.cfg file

```
[defaults]

host_key_checking = false
remote_user = ec2-user
ask_pass = false
private_key_file = ../../home/marwi/Cloud-Computing-Project/Ansible/ansible.pem
roles_path = ../../home/marwi/Cloud-Computing-Project/Ansible/roles
inventory= /etc/ansible/inventory/aws_ec2.yaml
```

Figure 32: Changing ansible.cfg default values

```
[inventory]

# enable inventory plugins, default: 'host_list', 'script', 'auto', 'yaml', 'ini', 'toml'
#enable_plugins = host_list, virtualbox, yaml, constructed
enable_plugins = aws_ec2
```

Figure 33: Enable plugin in ansible.cfg

Import Necessary Files

As the server is set up and fully functional, the rest of the files were brought into the ec2 instance using vi. Those files include the ansible.pem key file as it is necessary to establish a connection to the ec2 instances, as well as the different playbooks.

```
[ec2-user@ip-172-31-80-23 ~]$ vi ansible.pem
```

Figure 34: Creating ansible.pem

As mentioned before, vi was used to create the ansible.pem by manually typing in all the characters so that the key didn't have to be stored in any publicly accessible storage such as a git repository.

The same goes for the install-caldera.yaml file. However, this file needed to be changed slightly as the hosts were still configured for a manual input of the ip-addresses. Now that the created plugin automatically inputs and groups the ec2 instances by tags it is simply necessary to provide the group tag name for the hosts variable.

```
- name: Install Caldera on EC2 instance
  hosts: tag_caldera_
  become: yes
```

*Figure 35: Changing hosts group to tag_caldera_*

Milestone 5: Automated Caldera Deployment on EC2 Instances via CloudFormation & Ansible

AWS Systems Manager

To have the ability to run shell script in our CloudFormation, we had to use Systems Manager. It allows ec2 instances to be managed from within the interface itself and provides a functionality called “Run Command”. This functionality will be used to run the command on our server and that will deploy caldera as mentioned before.

First, AWS Systems Manager needed to be set up which required the Default Host Management Configuration to be enabled as seen below (Fig. 38).

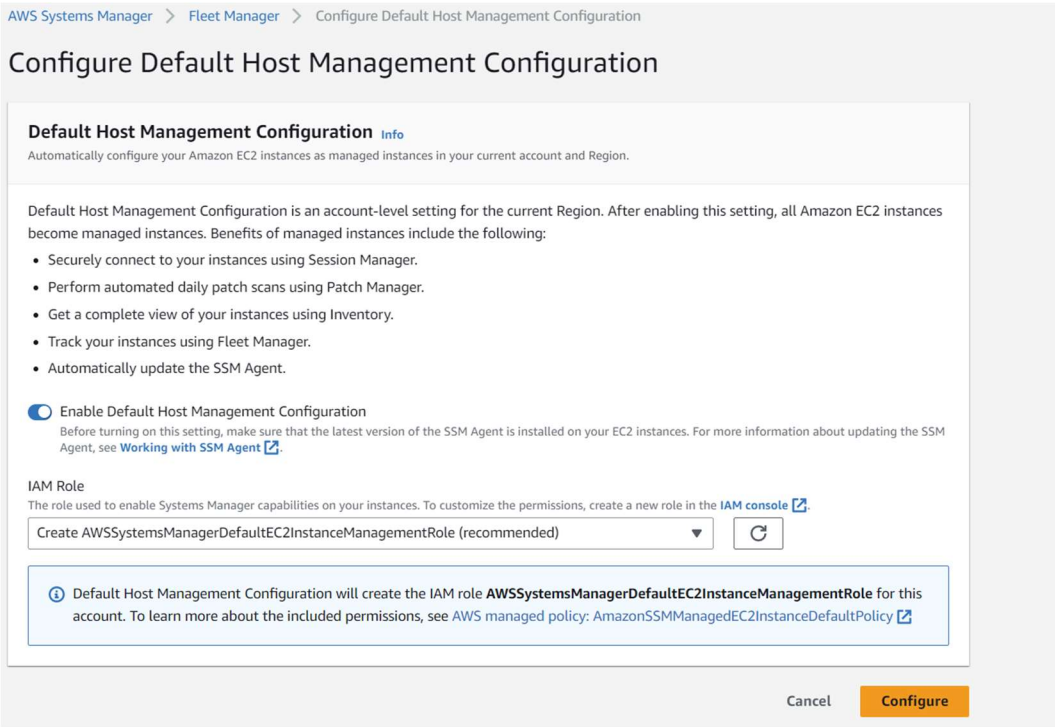

Figure 36: Configure default host management configuration

Next, an SSM Agent needed to be installed on the ansible server (Fig. 39). This allows the AWS Systems Manager to pick up on the ansible server, manage it, and most importantly run commands on it.

```
[ec2-user@ip-172-31-80-23 ~]$ sudo yum install -y https://s3.amazonaws.com/ec2-downloads-windows/SSMAgent/latest/linux_amd64/amazon-ssm-agent.rpm
Last metadata expiration check: 20:21:35 ago on Sat Apr 22 22:43:18 2023.
amazon-ssm-agent.rpm                                                    78 MB/s |  25 MB     00:00
Dependencies resolved.
================================================================================================================================
 Package                    Architecture          Version                  Repository                  Size
================================================================================================================================
Upgrading:
 amazon-ssm-agent           x86_64                3.2.815.0-1              @commandline                25 M

Transaction Summary
================================================================================================================================
Upgrade  1 Package

Total size: 25 M
Downloading Packages:
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
```

Figure 37: Installing SSM agent on Ansible Server

Systems Manager Challenges

After successfully installing the SSM Agent, multiple attempts to use the Run Command to run the ansible playbook were unsuccessful with the error message “ansible command not found”. Troubleshooting attempts from adding ansible to path, as well as other failed. The last idea was to install ansible using the run command itself (Fig. 40), which then finally started working. Since ansible would address the files in the /etc/ansible location again there would be no further configuration necessary.

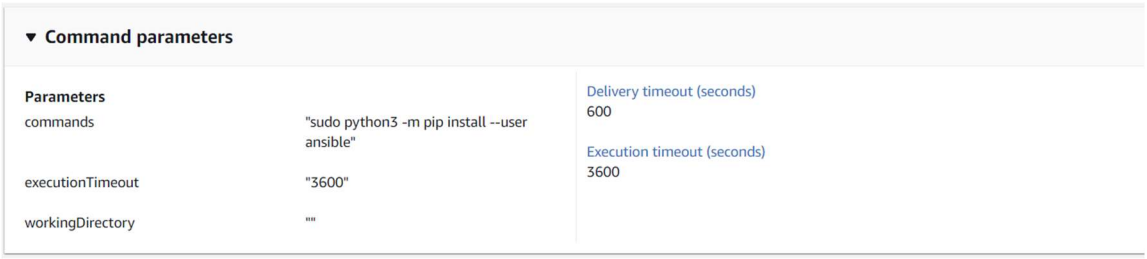

Figure 38: Installing ansible using run command

The next error was that boto3 was missing (Fig. 41), so just like ansible was installed using the run command, now boto3 was installed with its help.

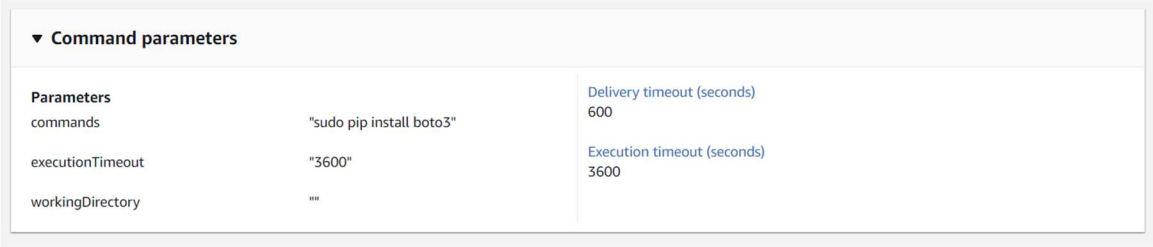

Figure 39: Installing boto3 using run command

The last error had to do was with the rights assigned to the ansible.pem files, so a simple chmod solved the last issues (Fig. 42).

```
[ec2-user@ip-172-31-80-23 sbin]$ PATH=$PATH:/root/.local/bin
[ec2-user@ip-172-31-80-23 sbin]$ cd /home/ec2-user
[ec2-user@ip-172-31-80-23 ~]$ chmod 400 ansible.pem
chmod: changing permissions of 'ansible.pem': Operation not permitted
[ec2-user@ip-172-31-80-23 ~]$ sudo chmod 400 ansible.pem
[ec2-user@ip-172-31-80-23 ~]$
```

Figure 40: Changing ansible.pem permissions

After this, the shell ran command
sudo ansible-playbook -u ec2-user -b /home/ec2-user/install-caldera.yaml --private-key=/home/ec2-user/ansible.pem

worked without further issues.

Cloud Formation

In this milestone, we deployed Caldera on an EC2 instance using CloudFormation. The deployment process was automated using AWS Systems Manager Run Command (RunShellScript) to execute an Ansible playbook in our Ansible Server.

CloudFormation Script

Task One: NewEC2

```
NewEC2Instance:
  Type: 'AWS::EC2::Instance'
  Properties:
    InstanceType: t2.micro
    ImageId: ami-02396cdd13e9a1257
    KeyName: ansible
    SecurityGroupIds:
      - sg-013e6b6c8233a38c8
      - sg-066fbedc9cb965bf9
    Tags:
      - Key: caldera
        Value: ''
```

Figure 41: EC2 creation cloud script part

The script (Fig. 32) starts off with creating an EC2 Instance and defining the properties:

- SecurityGroups(Ports):
  - 22: Allow SSH access for Ansible
  - 8888: Web-Port for Caldera
- Assigning Tag 'caldera' so the Ansible Server can target the created instance.

Task Two: RunShellScript

In the cloud formation script one of the necessary tasks is to have the ansible server run the install-caldera.yaml playbook so that caldera gets deployed on the created ec2 instances.

```
RunShellScript:
  Type: 'AWS::SSM::Association'
  Properties:
    Name: AWS-RunShellScript
    InstanceId: i-0427b0598ed16123c
    Parameters:
      commands:
        - sleep 300
        - sudo ansible-playbook -u ec2-user
        -b /home/ec2-user/install-caldera.yaml
        --private-key=/home/ec2-user/ansible.pem
```

Figure 42: Run command cloud formation script part

In Task Two (Fig. 37), we use AWS Systems Manager to run a couple of commands using 'AWS-RunShellScript' on our Ansible Server. The commands used:

- **sleep 300** – This gives the created EC2 instance enough time to setup and initialize before the next command it runs.
- **ansible-playbook** – Executes ansible playbook that then targets the created EC2 instance through the tag 'caldera'.

References

Installing ansible□. Installing Ansible - Ansible Documentation. (2023, March 30). Retrieved April 24, 2023, from https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#installing-and-upgrading-ansible

Pejsar, R. J. (1975). *The systems manager: A study in conflict*. Amazon. Retrieved April 24, 2023, from https://docs.aws.amazon.com/systems-manager/latest/userguide/what-is-systems-manager.html

Welcome to caldera's documentation!¶. Welcome to CALDERA's documentation! - caldera documentation. (n.d.). Retrieved April 24, 2023, from https://caldera.readthedocs.io/en/latest/

AWS::SSM::Document - AWS cloudformation . (n.d.). Retrieved April 26, 2023, from https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ssm-document.html

AWS Systems Manager Run Command - AWS Systems Manager. (n.d.). AWS Systems Manager Run Command - AWS Systems Manager. https://docs.aws.amazon.com/systems-manager/latest/userguide/run-command.html

AWS CloudFormation Designer interface overview - AWS CloudFormation. (n.d.). AWS CloudFormation Designer Interface Overview - AWS CloudFormation. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/working-with-templates-cfn-designer-overview.html