Ansible:

Session 1: Introduction and Installation

Today we will be covering the first session of the Ansible together. Ansible is the one the cool tools of DevOps. I like the way it handles jobs in network environment.

We will be learning what is Ansible, what kind of tool is it, what it does, what kind of skills, capability we will be having with using ansible, at the end of the class with a hands-on session we will be finishing today’s session. We will those issues related with ansible.

It’s an open-source automation tool. What do I men by saying automation tool? It is a tool that we can manage, install, remove software from servers and computers, change settings of running system, even provision resources from Cloud environment too.

What does it do? Basically, It goes to machines that either it would be virtual or tangible assets, login them, after successful login run some commands on those machines.   
Since it’s an open-source software the tool being updated with the ones who makes contributions. As new software and releases are published, their configuration and deployment setting are being updated as well. Constantly.

Slayt 3: What we can do with ansible, we can make deployment with it. What is deploy mean? Anyone has any idea of it?

You have just learnt with Terraform how to provision servers from cloud environment right? Now we will be configuring them.

Deployment is one of essential object in Dev&Ops. How do you define deploy? To make sure of a software is running on that particular point. If I say that I deployed something it means that I installed a software or code into particular area and run it.

Slide about ansible with data center:

İt goes those one of the machine, open the gate and makes changes that we told it to do. With scritps. We will tell ansible which machines will be deployed, with what kind of software and and with configurations.

Slayt 5: Advantages:

İts simple, its powerful and agentless. What does it mean agentless?

İt has human readable format, easy to understand, not a big deal, that does not any translation. Once you look at it, you will understand what it is related with and what changes will be done when you run that particular file for ansible. More secure, more efficient. Think of situation with agent you have to install that small program into any managed node you would like to control and it would be different versions of programs according to operating system of running nodes.

It support not only computers and servers but also support any devices that allows connection from ssh port such as routers, network devices and containers too.

It supports both static and dynamic inventories.

It’s in the yaml format. That does not need any other programming language. It performs in sequence.

Powerful because it makes deployment. Application deployment. It is a configuration management tool.

There must be a connection between ansible server and the machines that will be configured.

Configuration management systems are designed to streamline the process of controlling large numbers of servers, for administrators and operations teams. They allow you to control many different systems in an automated way from one central location. While there are many popular configuration management tools available for Linux systems, such as Chef and Puppet, these are often more complex than many people want or need. Ansible is a great alternative to these options because it offers an architecture that doesn’t require special software to be installed on nodes, using SSH to execute the automation tasks and YAML files to define provisioning details.

Slayt 9: Control Node

A control node is a system where Ansible is installed and set up to connect to your server. The Ansible control node is the machine we’ll use to connect to and control the Ansible hosts over SSH. Your Ansible control node can either be your local machine or a server dedicated to running Ansible. You can have multiple control nodes, and any system capable of running Ansible can be set up as a control node, including personal computers or laptops running a Linux or Unix based operating system. For the time being, Ansible can’t be installed on Windows hosts, but you can circumvent this limitation by setting up a virtual machine that runs Linux and running Ansible from there.

Managed Nodes

The systems you control using Ansible are called managed nodes. Ansible requires that managed nodes are reachable via SSH, and have Python 2 (version 2.6 or higher) or Python 3 (version 3.5 or higher) installed. Ansible supports a variety of operating systems including Windows servers as managed nodes.

Inventory

An inventory file contains a list of the hosts you’ll manage using Ansible. Although Ansible typically creates a default inventory file when installed, you can use per-project inventories to have a better separation of your infrastructure and avoid running commands or playbooks on the wrong server by mistake. Static inventories are usually created as .ini files, but you can also use dynamically generated inventories written in any programming language able to return JSON.

Playbook

A playbook contains an ordered list of tasks, and a few other directives to indicate which hosts are the target of that automation, whether or not to use a privilege escalation system to run those tasks, and optional sections to define variables or include files. Ansible executes tasks sequentially, and a full playbook execution is called a play. Playbooks are written in YAML format.

Tasks

In Ansible, a task is an individual unit of work to execute on a managed node. Each action to perform is defined as a task. Tasks can be executed as a one-off action via ad-hoc commands, or included in a playbook as part of an automation script.

Modules

The units of code Ansible executes. Each module has a particular use, from administering users on a specific type of database to managing VLAN interfaces on a specific type of network device. Modules are kept on the managed nodes to run the content of the code, then when the job is finished is deleted.

Conclusion

Ansible is a minimalist IT automation tool that has a gentle learning curve, thanks in part to its use of YAML for its provisioning scripts. It has a great number of built-in modules that can be used to abstract tasks such as installing packages and working with templates. Its simplified infrastructure requirements and accessible syntax can be a good fit for those who are getting started with configuration management.

Before going over to implementation section lets create a machines from AWS console.

excalidraw

Installation of Ansible:

# sudo yum update -y

Put command into chat section

Arrange installation so that give a break

for ubuntu systems # sudo apt update -y

#sudo amazon-linux-extras install ansible2 -y

Talk about ansible installation output

For ubuntu systems # sudo apt install ansible -y

Talk about installation methods

Use official pages

# ansible –version

Modules were included inside ansible but now only base modules, now the other packages can be used as collections learn collections

Now lets go to console again lets create two centos machine from console:

ansible\_ssh\_private\_key\_file=/home/ec2-user/<pem file>

chmod 400 firstkey.pem

$ sudo su

$ cd /etc/ansible

$ ls

$ vim hosts

[webservers]

node1 ansible\_host=<node1\_ip> ansible\_user=ec2-user

node2 ansible\_host=<node2\_ip> ansible\_user=ec2-user

[all:vars]

ansible\_ssh\_private\_key\_file=/home/ec2-user/<pem file>

show aws console connect command. Explain the paradigm.

Explain public ad private ip difference. Vpc, change in shutdown with new IP

Node>:/home/ec2-user

$ ansible all --list-hosts

$ ansible webservers --list-hosts

$ ansible all -m ping

$ ansible webservers -m ping

$ ansible node1 -m ping

Ansible localhost -a “hostname”

$ ansible-doc ping

Ansible-doc -l

$ ansible all -m ping -o

/etc/ansible/ansible.cfg

[defaults]

interpreter\_python=auto\_silent

private\_key\_file = ~/firstkey.pem

inventory = inventory.txt

host\_key\_checking = False

$ ansible --help

-a, -m, -i, -o, -v, -vvvv,--list-hosts, --become-user

$ ansible webservers -m shell -a “uptime”

ansible webservers -a "uptime"

$ ansible webservers -a "uptime"

Ansible webservers -a “date”

If you give more than one argument put into quotes

ansible webservers -m shell -a "systemctl status sshd"

shell variables are valid, directions > >> < can be used

ansible localhost -m shell -a "sudo cat /etc/shadow | grep ec2-user"

Command module is the default module. There is no need to put -m, just put -a

ansible webservers -m command -a 'df -h'

ansible localhost -a "df -h"

ansible localhost -a "uptime"

there is also raw module no python environment, network components

be aware of those shell command and raw modules are not idempotent and its not best used case scenario, as they don’t check status of remote nodes, no test, if they are in correct state, idempotence

$ vi testfile    # Create a text file name "testfile"

  "This is a test file."

```

```bash

$ ansible webservers -m copy -a "src=/etc/ansible/testfile dest=/home/ec2-user/testfile"

$ ansible node1 -m shell -a "echo Hello All > /home/ec2-user/testfile2 ; cat testfile2"

Now lets create another machine but this time let the machine be on ubuntu operating system.

$ cd etc/ansible

vim hosts

[ubuntuserver]

node3 ansible\_host=<node3\_ip> ansible\_user=ubuntu

$ ansible all --list-hosts

$ ansible all -m ping -o

$ ansible all -m shell -a "echo Hello TuranCyberHub > /home/ubuntu/testfile3"

$ ansible node3 -m shell -a "echo Hello TuranCyberHub > /home/ubuntu/testfile3"

$ ansible node1:node2 -m shell -a "echo Hello TuranCyberHub > /home/ec2-user/testfile3"

Ansible node3 -a “cat /home/ubuntu/testfile3”

Desired state

ansible webservers -b -m shell -a "amazon-linux-extras install -y nginx1 ; systemctl start nginx ; systemctl enable nginx ; systemctl start nginx"

```

Check whether nginx is running or not.

For ubuntu machine

$ ansible node3 -b -m shell -a "apt update -y ; apt-get install -y nginx ; systemctl start nginx; systemctl enable nginx"

$ ansible webservers -b -m shell -a "yum -y remove nginx"

$ ansible-doc yum

Ansible-doc user

- Run the command below ```twice```.

```bash

$ ansible webservers -b -m yum -a "name=git state=present"

-  Explain the difference of the standard outputs. Emphasize the changes in color and ```changed``` property together with idempotency.

71 ansible webservers -b -m shell -a "systemctl start nginx"

72 ansible webservers -b -m package -a "name=nginx state=absent"

$ ansible -b -m package -a "name=nginx state=present" all

ansible -b -m package -a "name=nginx state=absent" all

Yum apt package

$ vim inventory

```

```bash

[webservers]

node1 ansible\_host=<node1\_ip> ansible\_user=ec2-user

[webservers:vars]

ansible\_ssh\_private\_key\_file=/home/ec2-user/<YOUR-PEM-FILE-NAME>.pem

ansible -i inventory -m ping all

ansible -i inventory -b -m yum -a "name=httpd state=present" node1

ansible -i inventory -b -m shell -a "systemctl start httpd" all

ansible -i inventory -b -m yum -a "name=httpd state=absent" node

sudo nano etc/ansible/ansible.cfg

inventory file location

interpreter\_python=auto\_silent

default module name = command

Session 2 Playbooks

Idempotence

Shell command ansible localhost -m shell -a “useradd ec2-user” with shell command you can give example of touch file1 & rm file1 and run more time rm file1 and observe the error output

ansible localhost -m lineinfile -a "path=/home/ec2-user/file1 state=present line=hello2"

ansible localhost -m lineinfile -a "path=/home/ec2-user/file1 state=present line=hello3"just to add another line to the file

ansible localhost -m lineinfile -a "path=/home/ec2-user/file1 regexp='^hello3' line=hello4"

ansible localhost -b -m yum -a "name=httpd state=present"

systemctl status httpd

ansible localhost -b -m yum -a "name=httpd state=present"

ansible localhost -b -m yum -a "name=httpd state=absent"

**-** **name:** Replace a localhost entry with our own

**ansible.builtin.lineinfile:**

**path:** /etc/hosts

**regexp:** '^127\.0\.0\.1'

**line:** 127.0.0.1 localhost

**owner:** root

**group:** root

**mode:** '0644'

sudo cat /etc/passwd

Verbosity: result, task configuration, connection and script level

syntax-check

ansible-playbook –syntax-check webserver.yml similar to terraform validate

ansible-doc -l

ansible-doc module name

copy:

dest: /etc/resolv.conf  
 content: “nameserver 192.168.0.2\n”

yaml | new line, multiline

---

- name: sample

become: yes

hosts: localhost

tasks:

- name: yum install

yum:

name: httpd

state: latest

- name: package

package:

name: git

state: present

- name: ikinci

become: yes

hosts: all

tasks:

- name: package

package:

name: git

state: absent

---

- name: ilk

hosts: localhost

tasks:

- name: file copy

copy:

src: /home/ec2-user/file1

dest: /home/ec2-user/test/file2

- name: Install a list of packages

yum:

name:

- nginx

- postgresql

- postgresql-se

Session 3:

Handlers

Handlers are used to perform actions on a service, such as restarting or stopping a service that is actively running on the managed node’s system. Handlers are typically triggered by tasks, and their execution happens at the end of a play, after all tasks are finished. This way, if more than one task triggers a restart to a service, for instance, the service will only be restarted once and after all tasks are executed. Although the default handler behavior is more efficient and overall a better practice, it is also possible to force immediate handler execution if that is required by a task.

Roles

A role is a set of playbooks and related files organized into a predefined structure that is known by Ansible. Roles facilitate reusing and repurposing playbooks into shareable packages of granular automation for specific goals, such as installing a web server, installing a PHP environment, or setting up a MySQL server.

Session 4:

Session 5: