DevOps – Lecture 6

Automation

11 Oct 2022

Chinmaya Dehury

Chinmaya.Dehury@ut.ee
Where are we now?

1. SDLC + Why DevOps
2. DevOps Phases + Cloud Computing – Basics
3. VCS – GitLab
4. Containerization – Kubernetes
5. Microservices

6. Automation – Ansible

7. CI/CD – GitLab
8. Testing
9. Monitoring
10. Application Deployment Modeling
11. DataOps
12. DevOps + Serverless
13. Challenges + Future Scope
A quick Recap...

Automation
- What can be automated in your business?
- Key features
- Why automation?
- Benefits

Automation in DevOps
- Build Automation
- Test Automation
- Deployment Automation
- Infrastructure Automation
- Automation in Resource provisioning

Infrastructure as Code (IaC)
- IaC & Scripts
- Benefits
- Declarative vs Imperative
- Tools

Ansible Automation tool
- Basic architecture
- Inventories
- tasks
- modules
- collections
- Playbooks
- Roles
Notes on submissions/deadline

• Regarding active participation
  • You may also PM lecture (over Zulip) regarding your previous active participations

• Start lab exercises early
  • At least 4 contact hours with TA/lecture to fix the issues.

• Maximum score for any submission is 12, not 13.

• Improvements made by students:
Notes on Final Exam Dates

• You can appear final exam either on

  - Week 15: 6 Dec 2022, **Tuesday**, 4-6PM  OR
  - Week 16: 14 Dec 2022, **Wednesday**, 4-6PM

• Pick a date here (link will be made available in Zulip):
  - [https://forms.gle/FtVtPRbmAfEKrZL19](https://forms.gle/FtVtPRbmAfEKrZL19)
Automation

...performs tasks with reduced human assistance

https://www.redhat.com/en/topics/automation/what-is-devops-automation
What can be automated in your business?

• IT automation
• business automation
  • Business process management
  • Business process automation
• robotic process automation
• industrial automation
• AI/ML/Deep Learning
  • Adaptive algorithms that use predictive models to perform tasks without explicit instructions
  • E.g. *predictive maintenances*
  • Automatically detect repetitive manual tasks and later automate them
• Etc....

https://www.redhat.com/en/topics/automation
Key features of automation technologies

• Unified automation platforms
  • promote **consistency** and let you **standardize automated processes** and content across development, test, and production environments.

• Easy, straightforward operations
  • allow more team members to participate and contribute.

• Overall simplicity
  • promotes greater adoption of automation throughout your organization.
  • E.g. Winery GUI, AWS datapipline,

• Integration with other tools and products
  • allow you to more easily automate a broader range of tasks and components.
  • making it **highly pluggable**

• Platform scalability
  • allows you to simply expand the capacity of your pipeline as adoption grows.
  • E.g. scale according to the requests coming from mobile devices and laptop devices
...but why automation?

- Blue Cross and Blue Shield of North Carolina saved*
  - > US$850,000
  - 70,000 work hours.
  - by automating VM provisioning
  - executed 200,000 Ansible Playbooks

- The City of Denver, without Ansible Automation Platform
  - 20 minutes to create an individual Microsoft Teams environment
  - *Less than a min* with automation platform

- Creating ‘Autonomous’ AI

* https://www.redhat.com/en/resources/bcbsnc-case-study
...but why automation? - Benefits

• Consistency
  • Consistent infra for both dev and ops teams

• Scalability
  • Automatic scaling up/down

• Speed

• Flexibility

• Standardization
  • Both dev and ops team will follow the same standard toolstack

* https://www.redhat.com/en/resources/bcbsnc-case-study
Automation in DevOps

...use of technology to perform tasks with reduced human assistance [src].

helps you
- accelerate processes
- scale environments
- build continuous integration
- Continuous Testing
- build continuous delivery
- build continuous deployment (CI/CD) workflows

Src: https://www.redhat.com/en/topics/automation/what-is-devops-automation
Build Automation in DevOps

• Automatic source code retrieval
• Source code compilation into binary
• Packaging compiled files (maybe into a compressed one)

Key metrics
• Average build time
• Frequency of builds
• Success rate
  • x% success/failed

Source: https://www.redhat.com/en/topics/automation/what-is-devops-automation
Test Automation in DevOps

• Unit tests -> code analytics
• Integration tests
• System tests
• Installation test
• User experience tests
• UI tests

**Key metrics**
• Test execution time
• Test code coverage
• Code coverage criteria
  • **Function coverage** – has each function in the program been called?
  • **Statement coverage** – has each statement in the program been executed?
  • **Edge coverage** – has every edge in the control-flow graph been executed?
  • **Branch coverage** - *e.g. if and case statements*
  • **Condition coverage** – *e.g. true – false condition*
Deployment Automation in DevOps

- Model the environment
- Deploy the application
  - Orchestration of the workflow

Key metrics
- % of failed/success deployments
- Avg. deployment time
- Deployment frequency
- Environment provisioning time

Src: https://www.redhat.com/en/topics/automation/what-is-devops-automation
Infrastructure Automation in DevOps

- Create the required infrastructure
  - For development environment
  - For testing environment
  - For production environment

Key metrics
- Environment provisioning time
- Average deployment time
- Resource configuration
- Frequency of environment provisioning

Src: https://www.redhat.com/en/topics/automation/what-is-devops-automation
Automation in Resource provisioning

• What to provision? What does resource mean here?

• Server:
  • Provision the servers
    • Set up the physical servers
    • Install the necessary system software
  • configure the servers
  • Make it ready to access over network

• Network

• Service
  • Service set up
  • Data management
  • User management

• User
  • Create user
  • Assign to group(s)
  • Assign roles (viewer, commenter, editor, administrator, owner, etc.)
  • Assign permissions
Infrastructure as Code (IaC)

• Not a product

• A methodology
  • for rapid deployment of application
  • Preparing the environment
  • Configuration
Infrastructure as Code (IaC)

• No manual work to provision the infrastructure:
  • Create VM
  • Prepare specific OS image
  • Create specific network interface
  • Create storage server
  • Manually assigning computing resource to each VM

• E.g. setting up 1000 VMs for an application deployment may need more than 15 weeks and >10 dedicated staffs

• IaC:
  • No more manual work
  • Write a code ->
    • Provision the required infrastructure and configure
Infrastructure as Code (IaC) – Examples

- No manual work to provision the infrastructure:
  - Create VM
  - Prepare specific OS image
  - Create specific network interface
  - Create storage server
  - Manually assigning computing resource to each VM
  - E.g. setting up 1000 VMs for an application deployment may need more than 15 weeks and >10 dedicated staffs

- IaC:
  - No more manual work
  - Write a code ->
    - Provision the required infrastructure and configure

```python
@name: Create VM
os_server:
  state: present
  name: "{{ vm_name }}"
  image: "{{ image }}"
  key_name: "{{ key_name }}"
  flavor: "{{ flavor }}"
  network: "{{ network }}"
  security_groups: default
  boot_from_volume: true
  volume_size: 20
  timeout: 200
```

```python
@name: Delete VM
os_server:
  state: absent
  name: "{{ id }}"
```

```python
@name: provision instance
ec2:
  key_name: "{{ ssh_key_name }}"
  instance_type: "{{ instance_type }}"
  image: "{{ image }}"
  region: "{{ region }}"
  wait: true
  vpc_subnet_id: "{{ vpc_subnet_id }}"
  assign_public_ip: true
```

```python
@name: terminate instance
ec2:
  state: absent
  region: "{{ region }}"
  instance_id: "{{ id }}"
```
IaC & Scripts

• Scripts
  • Not suitable for large scale system
  • Hard to keep track of
  • Hard to maintain
  • Used to automate a static step
  • Not flexible

• IaC
  • It is also a script-like environment -> high-level scripting
  • Provides versatility of code
  • More human readable
  • Platform-agnostic
Infrastructure as Code (IaC) – Benefits

• Cost reduction
• Increase in speed of deployments
• Reduce errors -> risk
• Infrastructure/configuration consistency
  • Development and Operation teams are aligned
• Reduced management overhead
IaC: Declarative vs Imperative

2 ways to approach IaC—*declarative* or *imperative*

e.g. *Tea preparation*: “prepare black tea” vs “how to prepare black tea?”

**Declarative approach**

- Defines the desired state of the system
- Keeps a list of the current state of system objects

**Imperative approach**

- Defines the specific commands needed to achieve the desired configuration
- Order of execution of those commands
Infrastructure as Code (IaC) – Tools

• Ansible
• Puppet
• Chef
• Terraform
• SaltStack
• AWS CloudFormation
• Azure Resource Manager
• Google Cloud Deployment Manager
Infrastructure as Code (IaC) – Tools

• **Ansible**
  • Puppet
  • Chef
  • Terraform
  • SaltStack
  • AWS CloudFormation
  • Azure Resource Manager
  • Google Cloud Deployment Manager
Ansible Automation tool

- An open source *infrastructure automation* tool
- A *configuration* management tool
- Gives the ability to control and configure multiple servers from one single location
  - Avoid doing repeated tasks manually
  - E.g. Setting up the database server, web server, etc
- **Agentless**
  - Require only ssh access
- Over 1,300 modules
- Easy to learn
  - Because of so high level descriptive languages (using YAML)
- Founded in 2013
- Bought by Red Hat in 2015.
- Alternatives:
  - Puppet
  - Chef
  - Salt
Why Ansible?

• Agentless
  • Target system requires only sshd and python
  • No daemons or agents to install

• Security
  • Relies on ssh

• Easy to get started, compared to the others!
  • Human readable, YAML format

Basic Ansible Architecture

VM1
Master Node
Installed Ansible

Control node

ssh

Managed nodes

VM2

VM3
Ansible Inventories

VM1
Master Node
*Installed Ansible*

*ssh*

Inventory

VM2

VM3
Ansible Inventories

VM1
Master Node
Installed Ansible

ssh

Inventory 1
- VM2
- VM3

Inventory 2
- VM4
- VM5

Inventory 3
Ansible Inventories

Sample Inventory file (*ini format*)

```ini
[web-servers]
server1 ansible_host=192.168.0.1 ansible_port=1600
server2 ansible_host=192.168.0.2 ansible_port=1800

[offsite]
server3 ansible_host=10.160.40.1 ansible_port=22 ansible_user=root
server4 ansible_host=10.160.40.1 ansible_port=4300 ansible_user=root

[backup-servers]
server2 ansible_host=192.168.0.2 ansible_port=1800
server3 ansible_host=10.160.40.1 ansible_port=22 ansible_user=root
```
Ansible Inventories

The default inventory/host file is in (if you have not installed using pip)

/etc/ansible/hosts
Ansible tasks

Now, you want to test if all the servers are accessible from **VM1**

```bash
$ ansible all -m ping
```
Ansible tasks

Now, You want to test if all the servers are accessible from **VM1**

```bash
$ ansible web-servers -m ping
```
Ansible tasks

Now, you want to test if all the servers are accessible from **VM1**

```
$ ansible webervers -m ping
```

Host-group

Web-servers

Server1

Server2

Server3

Server4

Offsite

Backup-servers

LTAT.06.015 : Lec-06 : Automation 36
Ansible **modules**

Now, You want to test if all the servers are accessible from **VM1**

```bash
$ ansible web-servers -m ping
```

- **Web-servers**
  - `Server1`
  - `Server2`
  - `Server3`
  - `Server4`

- **Backup-servers**
  - `Server1`
  - `Server2`

- **Offsite servers**
  - `Server3`
  - `Server4`
Ansible modules

Now, you want to test if all the servers are accessible from **VM1**

```
$ ansible web-servers -m ping
```

**But what is Module?**
Ansible modules - Example

```bash
1 $ ansible web-servers -m service -a "name=apache2 state=running"
2 $ ansible offsite -m command -a "pwd"
3 $ ansible offsite -m command -a "ls"
4 $ ansible backup-servers -m copy -a "src=/tmp/mydummyfile dest=/home/centos/"
```
Ansible **modules**

• The units of code Ansible executes.
• Each module has a particular use.
• You can invoke
  • a single module with a task.
  • several different modules in a **playbook**
• modules are grouped in **collections**
Ansible modules

- The units of code Ansible executes.
- Each module has a particular use.
- You can invoke
  - a single module with a task.
  - several different modules in a playbook
- modules are grouped in collections

The List of collections

- amazon.aws
- ansible.builtin
- ansible.netcommon
- ansible.posix
- ansible.utils
- ansible.windows
- arista.eos
- awx.awx
- azure.azcollection
- check_point.mgmt
- chocolatey.chocolatey
- cisco.aci
- cisco.asa
- cisco.ios
- cisco.iosxr
- cisco.meraki
- cisco.mso
- cisco.nso
- cisco.nxos
- cisco.ucs
- cloudscale_ch.cloud
- community.aws
- community.azure
- community.crypt
- community.digitalocean
- community.docker
- community.fortios
- community.general
- community.google
- community.grafana
- ibm.qradar
- infiniidat.infinibox
- inspur.sm
- junipernetworks.junos
- kubernetes.core
- Mellanox.onyx
- netapp.aws
- netapp.azure
- netapp.cloudmanager
- netapp.elementsw
- netapp.ontap
- netapp.um_info
- netapp_eseries.santricity
- netbox.netbox
- nginxIo-cloudstack
- nginxIo-exoscale
- nginxIo.vultr
- openstack.cloud
- openshift.openvswitch
- ovirt.ovirt
- purestorage.flasharray
- purestorage.flashblade
- sensu.sensu_go
- servicenow.servicenow
- splunk.es
- t_systems_mms.icinga_director
- theforeman.foreman
- vyon.vyon
- wti.remote
- community.hashi_vault
- community.hrobot
- community.kubernetes
- community.libvirt
- community.mongodb
- community.mysql
- community.network
- community.okd
- community.postgresql
- community.proxy
- community.rabbitmq
- community.rivers
- community.skydive
- community.sops
- community.vmware
- community.windows
- community.zabbix
- containers.podman
- cyberark.conjur
- cyberark.pas
- dell EMC.enterprise_sonic
- dell EMC.openmanage
- dell EMC.os10
- dell EMC.os6
- dell EMC.os9
- f5 Networks.f5_modules
- fortinet.fortimanager
- fortinet.fortios
- frr.frr
- gluster.gluster
- google.cloud
- hetzner.hcloud
- hpe.nimble
Ansible collections

• Ansible.Builtin is a collections

The List of modules in Ansible.Builtin collection

- 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
- asyncc_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
- cron – Manage cron.d and crontab entries
- debconf – Configure a .deb package
- debug – Print statements during execution
- dnf – Manages packages with the dnf package manager
- dpkg_selections – Dpkg package selection selections
- expect – Executes a command and responds to prompts
- fail – Fail with custom message
- fetch – Fetch files from remote nodes
- file – Manage files and file properties
- find – Return a list of files based on specific criteria
- gather_facts – Gather facts about remote hosts
- get_url – Downloads files from HTTP, HTTPS, or FTP to node
- getent – A wrapper to the unix getent utility
- git – Deploy software (or files) from git checkouts
- group – Add or remove groups
- group_by – Create Ansible groups based on facts
- hostname – Manage hostname
- import_playbook – Import a playbook
- import_role – Import a role into a play
- import_tasks – Import a task list
- include – Include a play or task list
- include_role – Load and execute a role
- include_tasks – Dynamically include a task list
- include_vars – Load variables from files, dynamically within a task
- iptables – Modify iptables rules
- known_hosts – Add or remove a host from the known_hosts file
- lineinfile – Manage lines in text files
- meta – Execute Ansible ‘actions’
- package – Generic OS package manager
- package_facts – Package information as facts
- pause – Pause playbook execution
- ping – Try to connect to host, verify a usable python and return pong on success
- pip – Manages Python library dependencies
- raw – Executes a low-down and dirty command
- reboot – Reboot a machine
- replace – Replace all instances of a particular string in a file using a back-referenced regular expression
- rpm_key – Adds or removes a gpg key from the rpm db
- script – Runs a local script on a remote node after transferring it
- service – Manage services
- service_facts – Return service state information as fact data
- set_fact – Set host variable(s) and fact(s).
- set_stats – Define and display stats for the current ansible run
- setup – Gathers facts about remote hosts
- shell – Execute shell commands on targets
- slurp – Slurps a file from remote nodes
- stat – Retrieve file or file system status
- subversion – Deploys a subversion repository
- systemd – Manage systemd units
- sysvinit – Manage SysV services.
- tempfile – Creates temporary files and directories
- template – Template a file out to a target host
- tempfile – Creates temporary files and directories
- template – Template a file out to a target host
- unarchive – Unpacks an archive after (optionally) copying it from the local machine
- uri – Interacts with webservers
- user – Manage user accounts
- validate_argument_spec – Validate role argument specs.
- wait_for – Waits for a condition before continuing
- wait_for_connection – Waits until remote system is reachable/usable
- yum – Manages packages with the yum package manager
- yum_repository – Add or remove YUM repositories
Ansible collections

- **Ansible.Builtin** collections -> **ansible.builtin.file** is a module

Some functionalities of *file* module

- Change file ownership, group and permissions
- Give insecure permissions to an existing file
- Create a symbolic link
- Touch a file
- Create a directory if it does not exist
- Remove file (delete file)
- Recursively remove directory
Ansible **Playbooks**

- Ordered lists of tasks, saved so you can run those tasks in that order repeatedly.
- Include *variables* as well as *tasks*
- Written in YAML and are easy to read, write, share and understand.

https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html#about-playbooks
Ansible **Playbooks – an Example**

**Playbook1.yaml**

```yaml
---
- name: Update web servers
  hosts: web_servers
  remote_user: root

  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest
  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

  - name: Update db servers
  hosts: DATABASE
  remote_user: root

  tasks:
  - name: Ensure postgresql is at the latest version
    ansible.builtin.yum:
      name: postgresql
      state: latest
  - name: Ensure that postgresql is started
    ansible.builtin.service:
      name: postgresql
      state: started
```

**Execute the playbook:**

```bash
ansible-playbook Playbook1.yaml
```
Ansible **Playbooks**

- Ansible executes
  - each task in order
    - with specific arguments.
  - one at a time
  - against all machines matched by the host pattern.
- By Default:
  - When a task has executed on all target machines, Ansible moves on to the next task
  - Ansible runs in parallel against **all** the hosts in the pattern you set in the `hosts:` field of each play
- Instead of all, you can manage only a few machines at a time.
- The order of execution can also be changed

---

```yaml
- name: Update web servers
  hosts: webservers
  remote_user: root
  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest
  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.conf
      dest: /etc/httpd.conf

- name: Update db servers
  hosts: databases
  remote_user: root
  tasks:
  - name: Ensure postgresql is at the latest version
    ansible.builtin.yum:
      name: postgresql
      state: latest
  - name: Ensure that postgresql is started
    ansible.builtin.service:
      name: postgresql
      state: started
```

https://docs.ansible.com/ansible/latest/user_guide/playbooks_strategies.html#playbooks-strategies
https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html#about-playbooks
Ansible **Roles**

• A way of organizing tasks and related files to be later called in a playbook.
• Roles are a collection of tasks (not to confuse with Ansible collections),
  • moved from one playbook to another,
  • can be run independently but only through a playbook file.
• Roles can be considered as something similar to libraries
Ansible **Roles – An example**

- A way of organizing tasks and related files to be later called in a playbook.
- Roles are a collection of tasks,
  - moved from one playbook to another,
  - can be run independently but only through a playbook file.
- Roles can be considered as something similar to libraries
Idempotency in IaC

• Idempotent behavior:
  • Repeating the task does not change the final state

• Compare the “current state” with “desired state” of the system

• E.g. below “Create VM” task will not create multiple VMs if you run the task more than once

• Most Ansible modules are “Idempotent”

Source: https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html#desired-state-and-idempotency
Pull Model vs Push Model

**Push Model**
- Controller push the configurations/tasks to worker nodes
- You don’t need any client tool/agent on worker nodes

**Pull Model**
- Worker nodes pull the configuration/tasks from the controller/repository
- An agent is needed to regularly check and pull the configuration/tasks

Pull Model vs Push Model – An illustration

Summary

Automation
What can be automated in your business?

Key features

why automation?

Benefits

Automation in DevOps
Build Automation

Test Automation

Deployment Automation

Infrastructure Automation

Automation in Resource provisioning

Infrastructure as Code (IaC)
IaC & Scripts

Benefits

Declarative vs Imperative

Tools

Ansible
Basic architecture

Inventories

tasks

modules

collections

Playbooks

Roles

Ansible
Automation tool
Lab Sessions

• Installation and basic configuration of Ansible v2.9.
• Basic Ansible commands
• Working with Ansible Playbook
• Deploy flask application on k8s-cluster
• Working with gitlab, Ansible and k8s-cluster
References

- https://www.redhat.com/en
- https://docs.ansible.com/ansible/latest/user_guide/playbooks_strategies.html
- https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html
Any Question?

THANK YOU