DevOps Workflows with Ansible Engine/Tower

Robert Zahradníček
Agenda

- What is DevOps workflow about?
Agenda

● What is DevOps workflow about?
● What it means to scale DevOps workflow?
Agenda

- What is DevOps workflow about?
- What it means to scale DevOps workflow?
- Where can you utilise Ansible Engine and Ansible Tower in it?
Agenda

- What is DevOps workflow about?
- What it means to scale DevOps workflow?
- Where can you utilise Ansible Engine and Ansible Tower in it?
- Why should consider it?
What is DevOps workflow based on?

- **Culture** - brings people and processes together
What is DevOps workflow based on?

- **Culture** - brings people and processes together
- **Automation** - creates a natural fabric for all DevOps practices
What is DevOps workflow based on?

- **Culture** - brings people and processes together
- **Automation** - creates a natural fabric for all DevOps practices
- **Measurement** - permits continuous improvements
What is DevOps workflow based on?

- **Culture** - brings people and processes together
- **Automation** - creates a natural fabric for all DevOps practices
- **Measurement** - permits continuous improvements
- **Sharing** - enables collaboration and feedback
What is DevOps workflow about?

- It’s about efficiently producing useful outcomes
What is DevOps workflow about?

- It’s about efficiently producing useful outcomes
- It’s about how you deliver the changes from development to production
What is DevOps workflow about?

- It’s about efficiently producing useful outcomes
- It’s about how you deliver the changes from development to production
- It’s about continuously improving the next cycle of work from development to production
What is DevOps workflow about?

- It’s about efficiently producing useful outcomes
- It’s about how you deliver the changes from development to production
- It’s about continuously improving the next cycle of work from development to production
- It’s about enablement of new opportunities at each step to provide both value and allow experimentation
What is DevOps workflow about?

- It’s about efficiently producing useful outcomes
- It’s about how you deliver the changes from development to production
- It’s about continuously improving the next cycle of work from development to production
- It’s about enablement of new opportunities at each step to provide both value and allow experimentation
- It’s definitely not just about the velocity
What is DevOps workflow about?

- It’s about efficiently producing useful outcomes
- It’s about how you deliver the changes from development to production
- It’s about continuously improving the next cycle of work from development to production
- It’s about enablement of new opportunities at each step to provide both value and allow experimentation
- It’s definitely not just about the velocity
- It’s your framework for implementation of DevOps practices
What is DevOps workflow about?

- It’s about efficiently producing useful outcomes
- It’s about how you deliver the changes from development to production
- It’s about continuously improving the next cycle of work from development to production
- It’s about enablement of new opportunities at each step to provide both value and allow experimentation
- It’s definitely not just about the velocity
- **It’s your framework for implementation of DevOps practices**
- There is no one size fits all
What is DevOps workflow about?

- It’s about efficiently producing useful outcomes
- It’s about how you deliver the changes from development to production
- It’s about continuously improving the next cycle of work from development to production
- It’s about enablement of new opportunities at each step to provide both value and allow experimentation
- It’s definitely not just about the velocity
- It’s your framework for implementation of DevOps practices
- There is no one size fits all
- You can choose and integrate wide variety of tools in its implementation
DevOps Workflow:

- Dev: code, build, plan, test
- Ops: deploy, release, monitor, Operate

[1]
Drivers of DevOps workflows

- Eliminate repetitive tasks and time sinkholes
Drivers of DevOps workflows

- Eliminate repetitive tasks and time sinkholes
- Reduce friction, enable rapid iterations
Drivers of DevOps workflows

- Eliminate repetitive tasks and time sinkholes
- Reduce friction, enable rapid iterations
- Apply automation wherever possible (CI/CD, IaaS, ...)

1. Write a new code based on your Version Control flow.
DevOps workflow example

1. Write a new code based in Version Control flow.
2. CI tool detects the new changes, pulls them and runs them through build pipeline.
DevOps workflow example

1. Write a new code based in Version Control flow.
2. CI tool detects the new changes, pulls them and runs them through build pipeline.
3. Required testing environments are provisioned and the new build is deployed to them.
DevOps workflow example

1. Write a new code based in Version Control flow.
2. CI tool detects the new changes, pulls them and runs them through build pipeline.
3. Required testing environments are provisioned and the new build is deployed to them.
4. All the required testing is done and results are reported back to the CI.
DevOps workflow example

1. Write a new code based in Version Control flow.
2. CI tool detects the new changes, pulls them and runs them through build pipeline.
3. Required testing environments are provisioned and the new build is deployed to them.
4. All the required testing is done and results are reported back to the CI.
5. **Once the code is tested, changes are approved and merged, CI detects it and initiates release and deployment pipeline to the production server.**
DevOps workflow example

1. Write a new code based in Version Control flow.
2. CI tool detects the new changes, pulls them and runs them through build pipeline.
3. Required testing environments are provisioned and the new build is deployed to them.
4. All the required testing is done and results are reported back to the CI.
5. Once the code is tested, changes are approved and merged, CI detects it and initiates release and deployment pipeline to the production server.
6. **New version is continuously monitored and maintained in desired state.**
Scaling

- The majority of the workflow is dependent on automation (CI/CD, IaasC)
The majority of the workflow is dependent on automation (CI/CD, IaaC)
Tools can keep up with infrastructure and organization needs
Scaling

- The majority of the workflow is dependent on automation (CI/CD, Iaas)
- Tools can keep up with infrastructure and organization needs
- Collaboration and reusability of automation fabric is present
Scaling

- The majority of the workflow is dependent on automation (CI/CD, IaasC)
- Tools can keep up with infrastructure and organization needs
- Collaboration and reusability of automation fabric is present
- Enable self-service to others
<table>
<thead>
<tr>
<th>Ansible Engine</th>
<th>Ansible Tower</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Universal automation language</td>
<td>● Automation glue for integration with different services</td>
</tr>
<tr>
<td>● Human-readable format</td>
<td>● Transparent interface for executing automation defined via Ansible playbooks</td>
</tr>
<tr>
<td>● Built in features that can help us in collaboration and reusability</td>
<td>● Solution for automation scaling</td>
</tr>
<tr>
<td></td>
<td>● Solution for enabling self-service</td>
</tr>
</tbody>
</table>
Ansible building blocks

- **Module** - the units of code Ansible executes
- **Tasks** - the smallest unit of work by executing a module
  - Stop/start services and monitoring
  - Add/remove the server from our load balancers
  - Verify application state—is it ready to serve requests?
- **Roles** - shareable logical collection of tasks *(ansible-galaxy + molecule)*
  - Install and configure a web-server
  - Configure monitoring
  - Deploy application
- **Playbook** - ordered collection of tasks for the desired high-level action
  - Provision and deploy to production
  - Create testing environment
- **Inventory** - list of targets for the playbook
# Provision EC2 instances as defined in the variable `aws_ec2_instances`

- name: Provision EC2 instances

  ec2:
    aws_access_key: "{{ iam.aws_access_key }}"  # comes from group_vars/host_vars/extra_vars
    aws_secret_key: "{{ iam.aws_secret_key }}"  # comes from group_vars/host_vars/extra_vars
    region: "{{ aws_ec2_region }}"
    tenancy: "{{ aws_ec2_tenancy }}"
    vpc_subnet_id: "{{ aws_ec2_vpc_subnet_id }}"
    group: "{{ aws_ec2_group }}"
    assign_public_ip: "{{ aws_ec2_assign_public_ip }}"
    key_name: "{{ aws_ec2_key_name }}"
    count: "{{ aws_ec2_count }}"
    image: "{{ aws_ec2_image }}"
    instance_type: "{{ aws_ec2_instance_type }}"
    instance_tags: "{{ aws_ec2_instance_tags }}"
    state: "{{ aws_ec2_state }}"

  register: aws_ec2_provisioned_instances
ansible-aws-ec2-role

- defaults
- meta
- molecule
  - default
  - smoke
    - tests
      - INSTALL.rst
      - molecule.yml
      - playbook.yml
- tasks
  - main.yml
  - preflight.yml
  - provision.yml
  - .gitlab-ci.yml
- .yamllint
- config.toml
- README.md
dependency:
    name: galaxy
driver:
    name: docker
lint:
    name: yamllint

platforms:
- name: ${MOLECULE_DISTRO:-centos7}
  image: "geerlingguy/docker-${MOLECULE_DISTRO:-centos7}-ansible:latest"
  command: ${MOLECULE_DOCKER_COMMAND:-""}
  volumes:
    - /sys/fs/cgroup:/sys/fs/cgroup:ro
    - /var/run/docker.sock:/var/run/docker.sock
  privileged: true
  pre_build_image: true
provisioner:
    name: ansible
    log: true
    lint:
      name: ansible-lint
scenario:
  name: smoke
test_sequence:
  - lint
  - destroy
  - dependency
  - syntax
  - create
  - prepare
  # - converge
  # - idempotence
  # - side_effect
  - verify
  - destroy

verifier:
  name: testinfra
  lint:
    name: flake8
# Manage EC2 instances

- hosts: "{{ _hosts | default('localhost') }}"
gather_facts: false
become: false
vars:
  testing: false

vars_files:
  - "{{ varsfile }}"

tasks:
- include_role:
    name: aws-ec2

vars:
  aws_ec2_region: "{{ item.aws_ec2_region }}"
  aws_ec2_tenancy: "{{ item.aws_ec2_tenancy }}"
  aws_ec2_vpc_subnet_id: "{{ item.aws_ec2_vpc_subnet_id }}"
  aws_ec2_group: "{{ item.aws_ec2_group }}"
  aws_ec2_assign_public_ip: "{{ item.aws_ec2_assign_public_ip }}"
  aws_ec2_key_name: "{{ item.aws_ec2_key_name }}"
  aws_ec2_count: "{{ item.aws_ec2_count }}"
  aws_ec2_state: "{{ item.aws_ec2_state }}"
  aws_ec2_image: "{{ item.aws_ec2_image }}"
  aws_ec2_instance_type: "{{ item.aws_ec2_instance_type }}"
  aws_ec2_instance_tags: "{{ item.aws_ec2_instance_tags }}"

loop: "{{ aws_instances }}"
stage('Deploy') {
    dir('ansible') {
        ansiblePlaybook playbook: 'provision_ec2.yml', extraVars: 'varsfiles=./vars/awx_instances.yml'
        ansiblePlaybook playbook: 'rolling_deploy.yml'
    }
}

stage('Deploy') {
    ansibleTower(
        towerServer: 'Production',
        templateType: 'workflow',
        jobTemplate: '10',
        importTowerLogs: true,
        inventory: '2',
        jobTags: '',
        skipJobTags: '',
        limit: '',
        removeColor: false,
        verbose: true,
        credential: '2',
        extraVars: ''
    )
}
Ansible Tower building blocks

- Credentials
- Inventories
- Playbooks/Roles (Projects)
- Job templates
- Jobs
Ansible Tower definition of playbook run
- Job templates, Project Updates and Inventory updates can be included in **Workflows** to form arbitrary pipeline
- Job templates, Project Updates and Inventory updates can be included in **Workflows** to form arbitrary pipeline
- Job templates can **prompt on launch** for additional information (e.g. extra variables)
• Job templates, Project Updates and Inventory updates can be included in **Workflows** to form arbitrary pipeline
• Job templates can **prompt on launch** for additional information (e.g. extra variables)
• Job templates can be triggered via **Callbacks**
● Job templates, Project Updates and Inventory updates can be included in **Workflows** to form arbitrary pipeline
● Job templates can **prompt on launch** for additional information (e.g. extra variables)
● Job templates can be triggered via **Callbacks**
● All actions and job events can sent to external logging tools like Splunk or ELK
• Job templates, Project Updates and Inventory updates can be included in **Workflows** to form arbitrary pipeline
• Job templates can **prompt on launch** for additional information (e.g. extra variables)
• Job templates can be triggered via **Callbacks**
• All actions and job events can sent to external logging tools like Splunk or ELK
• Tower can scale horizontally
  ○ by creating a **cluster**
  ○ or adding **Isolated Nodes**
● Job templates, Project Updates and Inventory updates can be included in **Workflows** to form arbitrary pipeline
● Job templates can **prompt on launch** for additional information (e.g. extra variables)
● Job templates can be triggered via **Callbacks**
● All actions and job events can sent to external logging tools like Splunk or ELK
● Tower can scale horizontally
  ○ by creating a **cluster**
  ○ or adding **Isolated Nodes**
● You can dedicate resources by using **Instance Groups**
<table>
<thead>
<tr>
<th>Instance</th>
<th>Instances</th>
<th>Running Jobs</th>
<th>Total Jobs</th>
<th>Used Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>dev</td>
<td>3</td>
<td>9</td>
<td>89</td>
<td>61.8%</td>
</tr>
<tr>
<td>prod</td>
<td>4</td>
<td>6</td>
<td>26</td>
<td>27.3%</td>
</tr>
<tr>
<td>test</td>
<td>3</td>
<td>6</td>
<td>44</td>
<td>55.8%</td>
</tr>
<tr>
<td>tower</td>
<td>8</td>
<td>0</td>
<td>33</td>
<td>43.6%</td>
</tr>
</tbody>
</table>
• Job templates, Project Updates and Inventory updates can be included in **Workflows** to form arbitrary pipeline
• Job templates can **prompt on launch** for additional information (e.g. extra variables)
• Job templates can be triggered via **Callbacks**
• All actions and job events can be sent to external logging tools like Splunk or ELK
• Tower can scale horizontally
  ○ by creating a **cluster**
  ○ or adding **Isolated Nodes**
• You can dedicate resources by using **Instance Groups**
• Individual job execution can be scaled by using “**slicing**”
Ansible automation

**Ansible Tower**
- Role-Based Access Control
- Knowledge & Visibility
- Scheduled & Centralized Jobs

**Ansible Engine**
- Open Source Module Library
- Plugins
- Python Codebase

**Transport**
- SSH, WinRM, etc.

**Automate Your Enterprise**
- Infrastructure
  - Linux
  - Windows
  - UNIX
- Networks
  - Arista
  - Cisco
  - Juniper
- Containers
  - Docker
  - LXC
- Cloud
  - AWS
  - Google Cloud
  - Azure
- Services
  - Databases
  - Logging
  - Source Control Management

**Use Cases**
- Provisioning
- Configuration Management
- App Deployment
- Continuous Delivery
- Security & Compliance
- Orchestration
● We need to run the tests in some environment
● We want to have all the environments identical
You will need to “release” your build somewhere.

Usually you do it directly via your CI tool.

Writing it in your automation language makes it independent on the CI tool.

Interesting modules (shell, uri, docker_image, ...)
---

# Promote build artifact between repositories
# artifactory_promotion_name - handles where we are going to promote (defined in Artifactory promotion plugin)
#

- name: Ensure custom promotion plugin is present
copy:
    src: "${artifactory_promotion_plugin_name}.groovy"
    dest: "${artifactory_plugins_directory}"# ${ARTIFACTORY_HOME}/etc/plugins

- name: Reload plugins
  uri:
    url: "${artifactory_base}/api/plugins/reload"
    method: POST
    status_code: 201
    username: "${artifactory_user}"
    password: "${artifactory_password}"
    register: plugins
    failed_when: "artifactory_promotion_plugin_name not in plugins.content"

- name: Promote release in Artifactory
  uri:
    url: "${artifactory_base}/api/plugins/build/promote/${artifactory_promotion_name}/${artifactory_build}/${artifactory_build_number}"
    body_format: form-urlencoded
    body:
      status: "${promotion_status}"
    method: POST
    status_code: 201
    username: "${artifactory_user}"
    password: "${artifactory_password}"
• Instead to tight your deployment to your CI tool write it as a playbook
• This will detach these tasks away from particular CI
• This way you can deploy to arbitrary infrastructure in repeatable way
Interesting modules for deployment

- Monitoring (e.g., nagios, zabbix_maitenance, datadog_monitor, sensu_silence ...)
- Load balancers (e.g., haproxy, elb_application_lb, azure_rm_loadbalancer, gce_lb ...)
- Provisioning (e.g. ec2_intance, azure_rm_virtual_machine, gce, os_server ...)
- Services (e.g., service, command, file)
- Deployment (e.g., copy, unarchive, get_url)
- Programmatic verifications (e.g., command, Uniform Resource Identifier)
Deployment (e.g., copy, unarchive, get_url)

```yaml
---
# Retrieve builded artifact from
# Artifactory with following repository structure
# "http://artifactory.example.com/artifactory/releases/build-<version>.tgz"
# artifactory_base: "http://artifactory.example.com/artifactory/"
# artifactory_repository: "releases"
# artifact_name: "build-<version>.tgz"
- name: Get release from Artifactory
  get_url:
    url: "{{ artifactory_base }}/artifactory/{{ artifactory_repository }}/{{ artifact_name }}"
    dest: "{{ artifact_deploy_path }}/{{ artifact_name }}"
    url_username: "{{ artifactory_user }}"
    url_password: "{{ artifactory_password }}"
    force: True
```
Programmatic verifications (e.g., command, Uniform Resource Identifier)

```yaml
---
# Check if service healthcheck reports correct status
- name: Get /health data
  uri:
    url: "{{ service_url }}/health"
    method: GET
    status_code: 200
    return_content: yes
    register: healthcheck
  failed_when: "'RUNNING' not in healthcheck.content"
```
# Rolling update deployment of my app

- hosts: "{{ _hosts | default('app')}}"
  serial: 1

pre_tasks:
  
  - import_playbook: remove_lb_pool_member.yml
    vars:
      member: "{{ hostvars[inventory_hostname]['ansible_default_ipv4']['address'] }}"
    delegate_to: localhost

  - name: Wait for application endpoint to be drained
    wait_for:
      host: "{{ hostvars[inventory_hostname]['ansible_default_ipv4']['address'] }}"
      port: "{{ application_port }}"
      state: drained
      timeout: 60
      delegate_to: localhost

roles:
  - { role: app, postgres: "{{ hostvars[groups['dbs'][0]]['postgres'] }}" }

post_tasks:
  
  - name: Wait for application endpoint to come up
    wait_for:
      host: "{{ hostvars[inventory_hostname]['ansible_default_ipv4']['address'] }}"
      port: "{{ application_port }}"
      state: started
      timeout: 60
      delegate_to: localhost

  - include_playbook: add_lb_pool_member.yml
    vars:
      member: "{{ hostvars[inventory_hostname]['ansible_default_ipv4']['address'] }}"
    delegate_to: localhost
Day 2 Operations
# Deploy elasticsearch-curator to Graylog Elasticsearch cluster

- hosts: graylog
  user: ansible
  become: yes
  vars:
    curator_host: "{{ curator_host }}"
    curator_actions: ["graylog"]
    curator_logfile: ""
    curator_cron_jobs:
      - name: "Free Fast storage from old indices"
        user: root
        job: "{{ curator_exec_path }} --config {{ curator_config_path }}/config.yml {{ curator_config_path }}/free_fast_storage.yml"
        cron_min: 0
        cron_hour: 3
        file: "free_fast_storage"
      - name: "Move indices from SSD to Fast storage"
        user: root
        job: "{{ curator_exec_path }} --config {{ curator_config_path }}/config.yml {{ curator_config_path }}/ssd_to_fast_storage.yml"
        cron_min: 0
        cron_hour: 4
        file: "ssd_to_fast_storage"

roles:
- elasticsearch-curator
Database slave promotion example

```yaml
- hosts: "{{ _hosts }}"
  user: "{{ _user | default('ansible') }}"
  become: "{{ _become | default(True) }}"
  tasks:
    - name: Rename recovery.conf file
      when: ansible_hostname|string == "{{ _postgresql_slave_replica }}"
    - name: Turn off hot_standby option
      lineinfile:
        path: /etc/postgresql/9.6/main/postgresql.conf
        regexp: '^hot_standby = on'
        line: 'hot_standby = off'
        when: ansible_hostname|string == "{{ _postgresql_slave_replica }}"
    - name: Promote slave
      shell: pg_ctlcluster 9.6 main promote
      when: ansible_hostname|string == "{{ _postgresql_slave_replica }}"
    - name: Restart postgresql service
      service:
        name: postgresql
        state: restarted
      when: ansible_hostname|string == "{{ _postgresql_slave_replica }}"
```
Summary

- DevOps workflow is your framework for implementing DevOps practices
Summary

- DevOps workflow is your framework for implementing DevOps practices
- Ansible Engine + Tower gives you scalable transparent automation
Summary

- DevOps workflow is your framework for implementing DevOps practices
- Ansible Engine + Tower gives you scalable transparent automation
- Utilise Ansible Engine to provide the common language
Summary

- DevOps workflow is your framework for implementing DevOps practices
- Ansible Engine + Tower gives you scalable transparent automation
- Utilise Ansible Engine to provide the common language
- Utilise Ansible Tower to delegate usage to others
Summary

- DevOps workflow is your framework for implementing DevOps practices
- Ansible Engine + Tower gives you scalable transparent automation
- Utilise Ansible Engine to provide the common language
- Utilise Ansible Tower to delegate usage to others
- Utilise Ansible Tower to scale your automation
Summary

- DevOps workflow is your framework for implementing DevOps practices
- Ansible Engine + Tower gives you scalable transparent automation
- Utilise Ansible Engine to provide the common language
- Utilise Ansible Tower to delegate usage to others
- Utilise Ansible Tower to scale your automation
- DevOps workflow will always heavily depend on automation and therefore scale according based on the tools chosen
Thank you!