Automation I

Ansible Introduction & Ad-hoc mode

DE HOGESCHOOL MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

Managing Servers with SSH and Shell Scripts

- SSH is commonly used to make changes to servers.
- Inconsistent server environments may result from undocumented changes.
- Manual changes to multiple servers are time-consuming and error-prone.
- Most servers have complicated configurations and running multiple applications.
- **Snowflake servers** are hand-configured servers without documentation.
- Setting up a new server like an existing one requires significant time and effort.
- Shell scripts can automate server management, but complex scripts may not handle all edge cases and may not be scalable.
- Configuration management tools can streamline server management, improve efficiency, and reduce errors.

Introduction to Ansible for Configuration Management

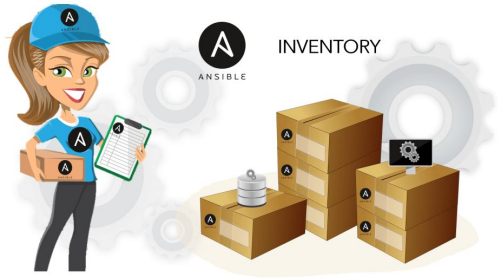
- CFEngine, Puppet, and Chef were popular configuration management tools in the mid-to-late 2000s.
- Many prefer shell scripting for its simplicity and familiarity.
- Ansible is a repeatable and centrally managed configuration management tool built by command line users.
- Ansible can run regular shell commands verbatim, making it easy to convert existing scripts into idempotent playbooks.
- Ansible pushes changes out to all servers by default and doesn't require extra software installation on servers (except for python!).
- Ansible is a versatile tool beyond server management for those involved in DevOps.

Ansible's Idempotent Configuration Management

- Idempotence ensures the same result regardless of how many times an operation is run.
- Ansible's idempotent configuration management maintains the same configuration regardless of how many times it's run.
- Many shell scripts have unintended consequences if run multiple times.
- Almost every aspect of Ansible modules and commands is idempotent.
- For non-idempotent commands, users can define when to run them and what constitutes a change or failure.
- Ansible simplifies maintaining an idempotent configuration on all servers.

On/*Off* buttons of a train's destination sign control panel. Pressing the *On* button (green) is an idempotent operation, since it has the same effect whether done once or multiple times. Likewise, pressing *Off* is idempotent.

Ansible inventory file



- Ansible uses an inventory file to communicate with servers, matching IP addresses or domain names to groups.
- Create a basic inventory file with one server by creating a file named `hosts.ini`.
  - `[example]`
  - `www.pxl.be`
- Edit the hosts file and add the group of servers you're managing and the domain name or IP address of a server in that group.
- If not using port 22 for SSH, add it to the address, e.g., `www.pxl.be:2222`.
- Inventory files can be placed in Ansible's global inventory file, but it requires sudo permissions, so it's better to maintain inventory alongside Ansible projects.

Inventory

- Determines which hosts you want to run against.
- Described in a configuration file
  - INI or YAML format
  - Default location is /etc/ansible/hosts.
  - IP address or hostname
  - Nodes can be assigned to groups.
  - dynamic (external source of truth) or static (files)

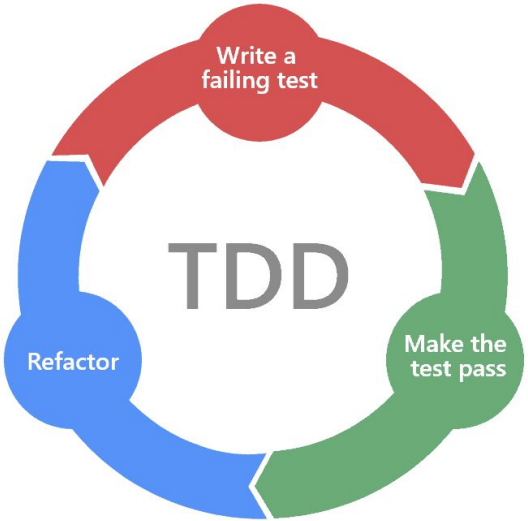
```
[lamp-varnish]
192.168.2.2

[lamp-www]
192.168.2.3
192.168.2.4

[a4d.lamp.db.1]
192.168.2.5

[lamp-db]
192.168.2.5 mysql_replication_role=server
192.168.2.6 mysql_replication_role=client

[lamp-memcached]
192.168.2.7
```

Running your first Ansible command

- `ansible -i hosts.ini example -m ping -u [username]`
- If successful, you should see a message with the result of your ping. If not, try running the command with "-vvvv" for verbose output.
- Ansible assumes key-based login for SSH
- to see memory usage on all servers in the example group:
  - `ansible -i hosts.ini example -a "free -h" -u [username]"`
- Use commands like "free -h" and "df -h" to check server statistics quickly.

Testing Infrastructure Development with Ansible

- Test-Driven Development (TDD) has become the norm for much of the software industry.
    - Write the test first, then make sure the test passes.
- Best practices dictate thorough testing for Infrastructure development.
- Changes to software are tested manually or automatically, and systems now integrate with Ansible and other tools for infrastructure testing.
- Testing infrastructure development is becoming increasingly important, and even testing changes locally before applying to production is better than "cowboy coding".
    - Cowboy coding is working directly in a production environment without documenting changes in code or having a way to roll back to a previous version.



Vagrant

- Vagrant is an open-source tool by HashiCorp, used for building and managing virtual machine environments.
- Vagrant can be used with various virtualization technologies such as VirtualBox, VMware, and Docker.
- Developers can use Vagrant to automate the setup and configuration of their virtual machines with provisioning tools like Ansible, Chef, and Puppet.
- Vagrantfiles are used to define the configuration of a virtual machine environment, including the operating system, software packages, and networking settings.
- Vagrant simplifies the process of setting up and managing development environments, reducing the time and effort required to get up and running.

Streamlining Server Management with Ansible

- Managing servers has become more complex due to virtualization and cloud usage
- Traditional sysadmins have many tasks to manage, such as:
  - Applying patches and updates
  - Checking resource usage and logs
  - Managing users, groups, DNS, and files
  - Deploying and maintaining applications
  - Rebooting servers and managing cron jobs
- Automation can handle many of these tasks, but some still require human touch
- Ansible allows for ad-hoc commands to be run on one or multiple servers simultaneously

Lab

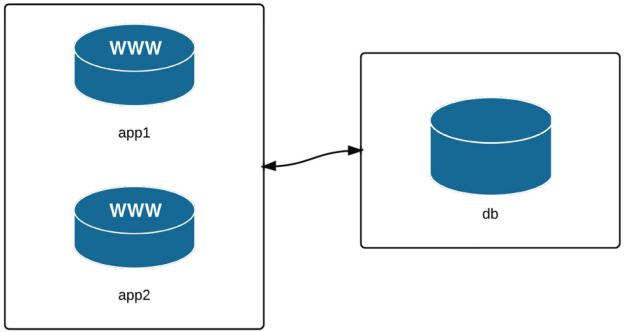
LAB set-up

- Managing three VMs: two app servers and a database server.
- This architecture is common for many simple web applications and websites.
- Specifying the path to the inventory file on the command line can be avoided by creating an `ansible.cfg` file in the root directory of your project.
- Use the github template https://github.com/PXLAutomation/automation-2223.git
    - Adjust number of hosts
    - Define the following inventory groups
        - app (2 hosts)
        - db (1 host)
        - multi (contains app en db)
    - Copy and adjust '`inventory`' to '`hosts.ini`'
    - Create an ansible.cfg file with the following contents if you want a default inventory file setting.

      ```
      [defaults]
      inventory = hosts.ini
      ```

    - Test with

      ```
      ansible multi -m ping
      ```

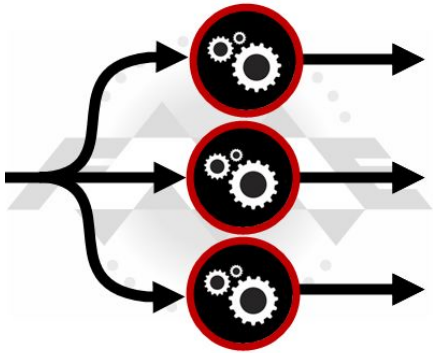


Ansible Ad-Hoc mode

Immediate status checks with Ad-Hoc commands

- To check disk space available:
  - `ansible multi -a "df -h"`
- To check available memory:
  - `ansible multi -a "free -m"`
- To check date and time on each server:
  - `ansible multi -a "date"`
- It's a good idea to use Network Time Protocol to ensure servers have synchronized time, which can be easily configured.

Ansible parallelism



- Use Ansible with the -a argument 'hostname' to run hostname against all servers
  - `ansible multi -a "hostname"`
- By default, Ansible runs commands in parallel using multiple process forks
- Using parallelism enables dramatic speedup even when managing 5-10 servers
- Add the argument `-f 1` to tell Ansible to use only one fork (default: 5)
  - `ansible multi -a "hostname" -f 1`
  - running the same command with -f 1 will always return results in the same order
- Increase the value of forks to speed up the process of running commands on multiple servers.

Ansible modules

- Ansible's modules offer abstraction and idempotency, making it easier to manage servers.
- If no module is provided with -m, the module ansible.builtin.command is used.
- Use Ansible's dnf module to install chrony daemon on servers to sync time.
    - Run this command on all three servers to install chrony:
    - `ansible multi -b -m dnf -a "name=chrony state=present"`
- Verify that chrony daemon is started and set to run on boot using Ansible's service module.
    - Run this command on all three servers to start and enable chrony daemon:
    - `ansible multi -b -m service -a "name=chronyd state=started enabled=yes"`
- Check server time synchronization with official time using:
    - `ansible multi -b -a "chronyc tracking"`

Chrony is a network time synchronization software designed for Linux systems. A modern alternative to the older NTP (Network Time Protocol) implementation that is commonly used for time synchronization.

Documentation

- https://docs.ansible.com/
- Ansible provides inbuilt help documentation for each module, which can be accessed by running the following command:
  - `ansible-doc <module-name>`
- This command will display the help documentation for the specified module, including all available options, examples, and usage.

Configure groups of servers, or individual servers

- Target commands to specific groups of servers in the inventory file
- Configure the Application servers:
  - Install Python package manager using dnf module
    - `ansible app -b -m dnf -a "name=python3-pip state=present"`
    - `ansible app -b -m pip -a "executable=pip3 name=django<4 state=present"`
  - Install Django using pip module
- Check if Django is installed and working correctly
  - `ansible app -a "python3 -m django --version"`

```
[app]
192.168.2.3
192.168.2.4
```

Configure the Database servers

- Install MariaDB and start it:
  - `ansible db -b -m dnf -a "name=mariadb-server state=present"`
  - `ansible db -b -m service -a "name=mariadb state=started enabled=yes"`
- Configure the system firewall to ensure only the app servers can access the database
  - `ansible db -b -m dnf -a "name=firewalld state=present"`
  - `ansible db -b -m service -a "name=firewalld state=started enabled=yes"`
  - `ansible db -b -m firewalld -a "zone=database state=present permanent=yes"`
  - `ansible db -b -m firewalld -a "source=192.168.56.0/24 zone=database state=enabled permanent=yes"`
  - `ansible db -b -m firewalld -a "port=3306/tcp zone=database state=enabled permanent=yes"`
- Install PyMySQL module on the managed server
  - `ansible db -b -m dnf -a "name=python3-PyMySQL state=present"`
- Allow MySQL access for one user from the app servers
  - `ansible db -b -m mysql_user -a "name=django host=% password=12345 priv=*.*:ALL state=present"`
- After completing these steps, you should be able to create or deploy a Django application on the app servers, then point it at the database server with the username django and password 12345.

Make changes to just one server

- Check the status of chronyd with the command:
  - `ansible app -b -a "systemctl status chronyd"`
- Restart the service on the affected app server with the command:
  - `ansible app -b -a "service chronyd restart" --limit "ip_addr"`
- Use the `--limit` argument to limit the command to a specific host in the group
  - `--limit` can match either an exact string or a regular expression (prefixed with *)

Manage users and groups with Ansible's ad-hoc commands

- Ansible's user and group modules make user and group management simple and standard across any Linux flavor.
- Add a group using the group module
  - `ansible app -b -m group -a "name=admin state=present"`
- Add a user with the user module and group parameter
  - `ansible app -b -m user -a "name=johndoe group=admin createhome=yes"`
- Additional parameters
  - UID with `uid=<uid>`
  - user's shell with `shell=<shell>`
  - user's password with `password=<encrypted-password>`
- To delete an account, use the user module with the state=absent and remove=yes parameters
  - `ansible app -b -m user -a "name=johndoe state=absent remove=yes"`
- Ansible's user module offers the same functionality as useradd, userdel, and usermod. see User module documentation.

Manage packages

- Ansible has a variety of package management modules for any flavor of Linux.
- There's also a generic package module for easier cross-platform Ansible usage.
- To install a generic package like git on any system, use the command:
  - `ansible app -b -m package -a "name=git state=present"`
- The package module works similarly to `dnf`, `apt`, and other package management modules.
- In multi-platform environments, package names may differ between OSes, but this can be addressed in Ansible.

Manage Files

- Ansible makes it easy to copy files from your host to remote servers, create directories, manage file and directory permissions and ownership, and delete files or directories.
- Use Ansible's stat module to get information about a file, such as its permissions, MD5, or owner, by running:
  - `ansible multi -m stat -a "path=/etc/environment"`
- Copy a file or a directory to remote servers with Ansible's copy module:
  - `ansible multi -m copy -a "src=/etc/hosts dest=/tmp/hosts"`
- The copy module is best suited for single-file copies and small directories. For large file copies, use Ansible's unarchive module or synchronize/rsync modules.
- Use Ansible's fetch module to retrieve files from remote servers:
  - `ansible multi -b -m fetch -a "src=/etc/hosts dest=/tmp"`
  - Fetch will, by default, put the `/etc/hosts` file from each server into a folder in the destination with the name of the host (in our case, the three IP addresses), then in the location defined by src. So, the db server's hosts file will end up in
    - `/tmp/[ip_addr]/etc/hosts`
  - Add the parameter `flat=yes` and set the dest to `dest=/tmp/` to copy files directly into the `/tmp` directory. However, filenames must be unique for this to work, so it's not as useful for copying files from multiple hosts.

Manage Directories

- Use the file module to create files and directories, manage permissions, and create symlinks.
- To create a directory
  - `ansible multi -m file -a "dest=/tmp/test mode=644 state=directory"`
- To create a symlink
  - `ansible multi -m file -a "src=/src/file dest=/dest/symlink state=link"`
- To delete a file, directory, or symlink, set the state to absent
  - `ansible multi -m file -a "dest=/tmp/test state=absent"`

Update servers asynchronously with asynchronous jobs

- Some operations take a long time (minutes or even hours).
  - For example, when you run `dnf update` or `apt-get update && apt-get dist-upgrade`, it could be a few minutes before all the packages on your servers are updated.
- You can tell Ansible to run the commands asynchronously and poll the servers to see when the commands finish.
- To run a command in the background, you set the following options:
  - `-B <seconds>`: the maximum amount of time (in seconds) to let the job run.
  - `-P <seconds>`: the amount of time (in seconds) to wait between polling the servers for an updated job status.
- Update servers asynchronously with asynchronous jobs:
  - Run `dnf -y update` on all servers asynchronously:
    - `ansible multi -b -B 3600 -P 0 -a "dnf -y update"`
  - Check on the status using Ansible's `async_status` module: (use `ansible_job_id iso. xxxxxxxxxxxxx.xxxxx`)
    - `ansible multi -b -m async_status -a "jid=xxxxxxxxxxxxx.xxxxx"`

Check log files

- Common log file operations (e.g. `tail`, `cat`, `grep`) work through the `ansible` command.
- Caveats:
  - Operations like `tail -f` don't work with Ansible.
  - Not recommended to run commands returning huge amount of data via stdout.
  - Redirect and filter output using the shell module instead of command module.
- View last lines of messages log file on each server:
  - `ansible multi -b -a "tail /var/log/messages"`
- Filter messages log using grep:
  - `ansible multi -b -m shell -a "tail /var/log/messages | grep ansible-command | wc -l"`

Manage cron jobs

- Periodic tasks run via cron are managed by the system's crontab.
- Ansible makes managing cron jobs easy with its cron module.
- To add a cron job, use the `cron` module with the name, hour, and job parameters.
  - If you want to run a shell script on all the servers every day at 4 a.m., add the cron job with:
    - `ansible multi -b -m cron -a "name='daily-cron-all-servers' hour=4 job='/path/to/daily-script.sh'"`
    - Ansible assumes * for all values you don't specify, but you can specify special time values using `special_time=[value]`, like `reboot`, `yearly`, or `monthly`
  - You can also set the user the job will run under via `user=[user]`, and create a backup of the current crontab by passing `backup=yes`.
- To remove a cron job, use the same cron command, and pass the name of the cron job you want to delete, and state=absent.
  - `ansible multi -b -m cron -a "name='daily-cron-all-servers' state=absent"`
- You can also use Ansible to manage custom crontab files by specifying the location to the cron file with `cron_file=cron_file_name`.
- Ansible denotes Ansible-managed crontab entries by adding a comment on the line above the entry like
  - `#Ansible: daily-cron-all-servers.`

Deploy a version-controlled application

- For proper deployments, use Ansible playbooks and rolling update features.
- Assume that we're running a simple application on one or two servers in the directory /opt/myapp
- Update the git checkout to the application's new version branch, 1.2.4, on all the app servers with the git module
  - `ansible app -b -m git -a "repo=git://example.com/path/to/repo.git dest=/opt/myapp update=yes version=1.2.4"`
- Run the application's update.sh shell script with the command module
  - `ansible app -b -a "/opt/myapp/scripts/update.sh"`
- You should use Ansible's more powerful and flexible application deployment features (see later)

Oefening ansible-adhoc-1

- Create an ubuntu-based instance.
- Deploy this application with the 'ansible' command:
  - *hello.py*

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
return 'PXL App!'

app.run(host='0.0.0.0', port=80)
```

Oefening ansible-adhoc-2 (AWS)

- Create and launch 3 EC2 linux instances, reachable through ssh keys and reachable via http.
- Log in to test ssh connectivity.
- Update packages.
- Install and start the latest nginx version in 1 of the 3 instances.
- Test by browsing to the ip address of the host.

Oefening ansible-adhoc-2 (AWS)

- Create an ansible inventory file
  - Place the 3 hosts in "webservers" group
- Ping to all servers.
- Using ad-hoc commands, check hostname, free ram, free disk space, last 10 log messages
  - max 1 host in parallel
- Check using the 'ansible' command if the nginx package is installed on all hosts.

Oefening ansible-adhoc-2 (AWS)

- Subdivide the servers into 2 groups
  - webservers (2 hosts)
  - nodes (1 host)
- Installeer docker on the machines in **nodes**.
- Install nginx exclusively in the machines in **webservers**.

end

Herhaling:
SSH-connecties met private/public keypair

- SSH keypair - Private key beveiligd met wachtwoord
- Passwordless connecting over ssh
- Indien je slechts éénmaal je private-key wilt unlocken en vervolgens meerdere malen gebruiken voor verscheidene ssh-connecties
    - `ssh-agent bash` - start een nieuwe shell met de agent running
    - `ssh-add ~/.ssh/id_ed25519` - houdt de private key(s) in het geheugen
- We moeten dus niet telkens opnieuw de passphrase opgeven als we een nieuwe ssh-connectie starten

```
student@ubuntu-server:~$ ssh-agent bash
student@ubuntu-server:~$ ssh-add ~/.ssh/id_ed25519
Enter passphrase for /home/student/.ssh/id_ed25519:
Identity added: /home/student/.ssh/id_ed25519 (student@ubuntu-server)
student@ubuntu-server:~$ ssh student@192.168.246.129
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-47-generic x86_64)
```