**Projects**

* [Projects](https://darey.io/project-listings/)
* [Docs](https://www.darey.io/projects/)
* [(STEP 23) Project 13 Ansible Dynamic Assignments (Include) and Community Roles](https://www.darey.io/docs/ansible-dynamic-assignments-include-and-community-roles/)

**ANSIBLE DYNAMIC ASSIGNMENTS (INCLUDE) AND COMMUNITY ROLES**

[Mark as Completed](https://www.darey.io/docs/ansible-dynamic-assignments-include-and-community-roles/?markcomplete=yes)Submit Project for Review

**IMPORTANT NOTICE:** Ansible is an actively developing software project, so you are encouraged to visit [Ansible Documentation](https://docs.ansible.com/) for the latest updates on modules and their usage.

Last 2 projects have already equipped you with some knowledge and skills on Ansible, so you can perform configurations using playbooks, roles and imports. Now you will continue configuring your UAT servers learning and practicing new Ansible concepts and modules.

In this project we will introduce [dynamic assignments](https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse.html#includes-dynamic-re-use) by using include module.

Now you may be wondering, what is the difference between **static** and **dynamic** assignments?

Well, from [Project 12](https://professional-pbl.darey.io/en/latest/project12.html), you can already tell that static assignments use import Ansible module. The module that enables dynamic assignments is include.

Hence,

import = Static

include = Dynamic

When the **import** module is used, all statements are pre-processed at the time playbooks are [**parsed**](https://en.wikipedia.org/wiki/Parsing). Meaning, when you execute site.yml playbook, Ansible will process all the playbooks referenced during the time it is parsing the statements. This also means that, during actual execution, if any statement changes, such statements will not be considered. Hence, it is static.

On the other hand, when **include** module is used, all statements are processed only during execution of the playbook. Meaning, after the statements are **parsed**, any changes to the statements encountered during execution will be used.

Take note that in most cases it is recommended to use static assignments for playbooks, because it is more reliable. With dynamic ones, it is hard to debug playbook problems due to its dynamic nature. However, you can use dynamic assignments for environment specific variables as we will be introducing in this project.

Instructions On How To Submit Your Work For Review And Feedback

To submit your work for review and feedback – follow [**this instruction**](https://darey.io/docs/how-to-submit-your-work-for-review-and-feedback-2/).

**INTRODUCING DYNAMIC ASSIGNMENT INTO OUR STRUCTURE**

Introducing Dynamic Assignment Into Our structure

In your https://github.com/<your-name>/ansible-config-mgt GitHub repository start a new branch and call it dynamic-assignments.

Create a new folder, name it dynamic-assignments. Then inside this folder, create a new file and name it env-vars.yml. We will instruct site.yml to include this playbook later. For now, let us keep building up the structure.

Your GitHub shall have following structure by now.

**Note:** Depending on what method you used in the previous project you may have or not have roles folder in your GitHub repository – if you used ansible-galaxy, then roles directory was only created on your Jenkins-Ansible server locally. It is recommended to have all the codes managed and tracked in GitHub, so you might want to recreate this structure manually in this case – it is up to you.

├── dynamic-assignments

│ └── env-vars.yml

├── inventory

│ └── dev

└── stage

└── uat

└── prod

└── playbooks

└── site.yml

└── roles (optional folder)

└──...(optional subfolders & files)

└── static-assignments

└── common.yml

Since we will be using the same Ansible to configure multiple environments, and each of these environments will have certain unique attributes, such as **servername**, **ip-address** etc., we will need a way to set values to variables per specific environment.

For this reason, we will now create a folder to keep each environment’s variables file. Therefore, create a new folder env-vars, then for each environment, create new **YAML** files which we will use to set variables.

Your layout should now look like this.

├── dynamic-assignments

│ └── env-vars.yml

├── env-vars

└── dev.yml

└── stage.yml

└── uat.yml

└── prod.yml

├── inventory

└── dev

└── stage

└── uat

└── prod

├── playbooks

└── site.yml

└── static-assignments

└── common.yml

└── webservers.yml

Now paste the instruction below into the env-vars.yml file.

---

- name: collate variables from env specific file, if it exists

hosts: all

tasks:

- name: looping through list of available files

include\_vars: "{{ item }}"

with\_first\_found:

- files:

- dev.yml

- stage.yml

- prod.yml

- uat.yml

paths:

- "{{ playbook\_dir }}/../env-vars"

tags:

- always

Notice 3 things to notice here:

1. We used include\_vars syntax instead of include, this is because Ansible developers decided to separate different features of the module. From Ansible version **2.8**, the include module is deprecated and variants of include\_\* must be used. These are:

* [include\_role](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/include_role_module.html#include-role-module)
* [include\_tasks](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/include_tasks_module.html#include-tasks-module)
* [include\_vars](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/include_vars_module.html#include-vars-module)

In the same version, variants of **import** were also introduces, such as:

* [import\_role](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/import_role_module.html#import-role-module)
* [import\_tasks](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/import_tasks_module.html#import-tasks-module)

1. We made use of a [special variables](https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html) { playbook\_dir } and { inventory\_file }. { playbook\_dir } will help Ansible to determine the location of the running playbook, and from there navigate to other path on the filesystem. { inventory\_file } on the other hand will dynamically resolve to the name of the inventory file being used, then append .yml so that it picks up the required file within the env-vars folder.
2. We are including the variables using a loop. with\_first\_found implies that, looping through the list of files, the first one found is used. This is good so that we can always set default values in case an environment specific env file does not exist.

# UPDATE SITE.YML WITH DYNAMIC ASSIGNMENTS

#### Update site.yml with dynamic assignments

Update site.yml file to make use of the dynamic assignment. (At this point, we cannot test it yet. We are just setting the stage for what is yet to come. So hang on to your hats)

**site.yml** should now look like this.

---

- hosts: all

- name: Include dynamic variables

tasks:

import\_playbook: ../static-assignments/common.yml

include: ../dynamic-assignments/env-vars.yml

tags:

- always

- hosts: webservers

- name: Webserver assignment

import\_playbook: ../static-assignments/webservers.yml

#### Community Roles

Now it is time to create a role for MySQL database – it should install the MySQL package, create a database and configure users. But why should we re-invent the wheel? There are tons of roles that have already been developed by other open source engineers out there. These roles are actually production ready, and dynamic to accomodate most of Linux flavours. With Ansible Galaxy again, we can simply download a ready to use ansible role, and keep going.

#### Download Mysql Ansible Role

You can browse available community roles [here](https://galaxy.ansible.com/home)

We will be using a [MySQL role developed by geerlingguy](https://galaxy.ansible.com/geerlingguy/mysql).

**Hint:** To preserve your your GitHub in actual state after you install a new role – make a commit and push to master your ‘ansible-config-mgt’ directory. Of course you must have git installed and configured on Jenkins-Ansible server and, for more convenient work with codes, you can configure Visual Studio Code to work with this directory. In this case, you will no longer need webhook and Jenkins jobs to update your codes on Jenkins-Ansible server, so you can disable it – we will be using Jenkins later for a better purpose.

On Jenkins-Ansible server make sure that git is installed with git --version, then go to ‘ansible-config-mgt’ directory and run

git init

git pull https://github.com/<your-name>/ansible-config-mgt.git

git remote add origin https://github.com/<your-name>/ansible-config-mgt.git

git branch roles-feature

git switch roles-feature

Inside roles directory create your new MySQL role with ansible-galaxy install geerlingguy.mysql and rename the folder to mysql

mv geerlingguy.mysql/ mysql

Read README.md file, and edit roles configuration to use correct credentials for MySQL required for the tooling website.

Now it is time to upload the changes into your GitHub:

git add .

git commit -m "Commit new role files into GitHub"

git push --set-upstream origin roles-feature

Now, if you are satisfied with your codes, you can create a Pull Request and merge it to main branch on GitHub.

**LOAD BALANCER ROLES**

Load Balancer roles

We want to be able to choose which Load Balancer to use, Nginx or Apache, so we need to have two roles respectively:

1. Nginx
2. Apache

With your experience on Ansible so far you can:

* Decide if you want to develop your own roles, or find available ones from the community
* Update both static-assignment and site.yml files to refer the roles

***Important Hints:***

* Since you cannot use both **Nginx** and **Apache** load balancer, you need to add a condition to enable either one – this is where you can make use of variables.
* Declare a variable in defaults/main.yml file inside the Nginx and Apache roles. Name each variables enable\_nginx\_lb and enable\_apache\_lb respectively.
* Set both values to false like this enable\_nginx\_lb: false and enable\_apache\_lb: false.
* Declare another variable in both roles load\_balancer\_is\_required and set its value to false as well
* Update both assignment and site.yml files respectively

loadbalancers.yml file

- hosts: lb

roles:

- { role: nginx, when: enable\_nginx\_lb and load\_balancer\_is\_required }

- { role: apache, when: enable\_apache\_lb and load\_balancer\_is\_required }

site.yml file

- name: Loadbalancers assignment

hosts: lb

- import\_playbook: ../static-assignments/loadbalancers.yml

when: load\_balancer\_is\_required

Now you can make use of env-vars\uat.yml file to define which loadbalancer to use in UAT environment by setting respective environmental variable to true.

You will activate load balancer, and enable nginx by setting these in the respective environment’s env-vars file.

enable\_nginx\_lb: true

load\_balancer\_is\_required: true

The same must work with apache LB, so you can switch it by setting respective environmental variable to true and other to false.

To test this, you can update inventory for each environment and run Ansible against each environment.

Congratulations!

You have learned and practiced how to use Ansible configuration management tool to prepare UAT environment for Tooling web solution.



Next project

Next project is a capstone project for this part of your **Project Based Learning** journey – it will require all previously gained knowledge and skills, and introduce more new and exciting concepts and DevOps tools!

Get ready for new challenges ahead! Full Speed Forward!

