Table of Contents

Pre-Class Assignment

Introduction

The purpose of the pre-class assignment is to refresh your Red Hat Ansible Engine knowledge and make sure that you are comfortable with the fundamental Ansible Engine skills necessary for successful completion of the course. The assignment involves working with:

- Inventory
- Playbooks
- Modules
- Tasks
- Roles
- Variables
- Templates

You build on this assignment throughout the course, and it is essential that you arrive with a GitHub repository or similar publicly available repository of your work.

Overview

In this scenario, you are a consultant assigned to FinanceTech, a FinTech company. FinanceTech develops software for on-premise deployment and has started to offer their platform as a SaaS offering initially deployed from their datacenters. But they are planning to move to third-party cloud providers such as Amazon Web Services (AWS).

An internal team with limited Ansible Engine knowledge has downloaded Ansible Engine and used it to deploy a three-tier Java-based application on-premise as part of their SaaS offering. However, their approach has resulted in a functional but inefficient Ansible Engine implementation with manual steps, redundancy, poor and inconsistent style, and limited use of advanced features. The poor structure hampers reuse, and already teams have started to use their own customized versions of the playbook.

Management recognizes the potential benefits of Ansible Engine and has hired you to reimplement the existing deployment as a Proof of Concept (POC) and potential opening for company-wide adoption. In addition, the expectation is that the project allows a seamless move to third-party cloud providers.

In short, your assignment is to reimplement their "bad Ansible" into "good Ansible."

Goals

- Set up and customize your environment via `ansible.cfg`
- Explore and verify the lab infrastructure
  - Execute ad hoc commands (`ping` , `date` , `uptime`) on each Ansible node
- Create a well-formed static inventory
- Refactor the existing monolithic playbook:
  - Adopt a clear and consistent style
  - Remove redundant tasks (plays)
  - Provide meaningful and consistent feedback during playbook execution
  - Implement roles where appropriate
  - Use optimal module choices for each task
  - Leverage Ansible Galaxy when appropriate
  - Introduce the use of templates
  - Create and implement tagging guidelines

- Use handlers where appropriate
- Use Ansible Vault to protect sensitive information

1. Provision Lab Environment

1. Navigate to the OPENTLC lab portal (https://labs.opentlc.com/) and log in using your OPENTLC credentials.
   - If you have forgotten your credentials, visit the OPENTLC Account Management (https://www.opentlc.com/account/) page.
2. Select **Services → Catalogs → All Services → OPENTLC Automation**.
3. Select **Three Tier Application** and click **Order** to provision your environment.
4. Click the **Lab Parameters** tab and select your **Region**.
5. Accept terms and conditions if required, then click **Submit**.
6. Check your email for a message from Red Hat describing how to connect to the environment, including your GUID (unique identifier), and environment details.
   - Further details are provided in the lab instructions.

Deployment can take between 15 and 45 minutes depending on the software deployed.

1.1. Review Provisioned Environment Hosts

The lab environment consists of four internal servers behind a bastion host or *jumpbox*. To eliminate the configuration of the jumpbox for this lab, all of the internal hosts to the Internet are exposed. This allows direct access from your laptop. A front-end server, `frontend1`, is exposed for HTTP/HTTPS traffic and is used as a load balancer for the application servers.

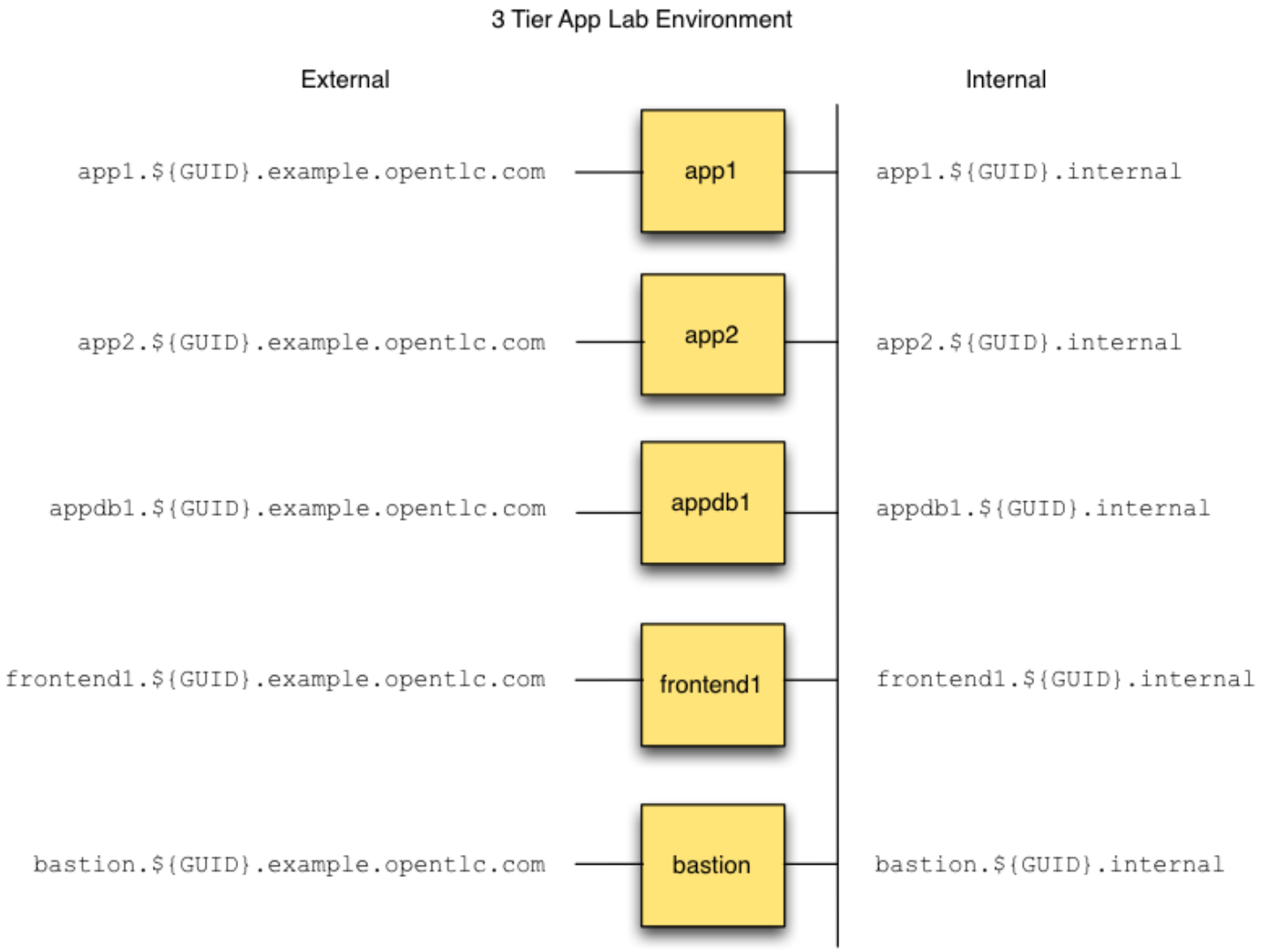
At this point, the machines do not have their respective payloads (HAProxy, Tomcat, PostgreSQL) installed or configured.

- Bastion server: `bastion.${GUID}.internal`, `bastion.${GUID}.example.opentlc.com`
- HAProxy server: `frontend1.${GUID}.internal`, `frontend1.${GUID}.example.opentlc.com`
- Tomcat servers: `app{1,2}.${GUID}.internal`, `app{1,2}.${GUID}.example.opentlc.com`
- Database server: `appdb1.${GUID}.internal`, `appdb1.${GUID}.example.opentlc.com`

You can either work and run your playbooks directly from `bastion`, or install `ansible` locally on your laptop and work from there.

Certain tasks below **must** be executed on the bastion host. Code and command examples will indicate which host in the prompt:

- On your laptop: `[laptop ]$`
- On the bastion host: `[user-company.com@bastion ~]$`
- On either, depending on where you plan to run Ansible Engine: `[laptop or bastion ~]$`



2. Explore and Verify Environment

In this section, you verify the infrastructure and hosts needed for Ansible Engine. Instances are created for you, and you can verify all of the hosts from the bastion host.

2.1. Connect to Environment

1. (Optional but recommended) Set some environment variables using your lab credentials:

```
[laptop ]$ export GUID=<"GUID from email">
[laptop ]$ export MYKEY=<~/.ssh/your_key.pem>
[laptop ]$ export MYUSER=<username-company.com>

# For example
[laptop ]$ export GUID=e4gh
[laptop ]$ export MYKEY=~/.ssh/sborenstkey
[laptop ]$ export MYUSER=shacharb-redhat.com
```

TEXT

2. Connect to the bastion host with your OPENTLC ID and private key:

```
[laptop ]$ ssh -i ${MYKEY} ${MYUSER}@bastion.${GUID}.example.opentlc.com
```

TEXT

2.2. Explore Environment

1. As `root` , use the `ansible --lists-hosts` command to list the available hosts:

```
[user-company.com@bastion ~]$ sudo -i
[root@bastion ~]# ansible all --list-hosts
```

TEXT

Sample Output

```
  hosts (5):
    frontend1.GUID.internal
    app1.GUID.internal
    app2.GUID.internal
    appdb1.GUID.internal
    support1.GUID.internal
```

TEXT

`support1.GUID.internal` is not used in this assignment and can be ignored.

2. Use the Ansible `ping` command to verify that all of your hosts are running:

```
[root@bastion ~]# ansible all -m ping -v
Using /etc/ansible/ansible.cfg as config file
```

TEXT

Sample Output

TEXT

```
frontend1.GUID.internal | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
appdb1.GUID.internal | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
app2.GUID.internal | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
app1.GUID.internal | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
support1.GUID.internal | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

The -v flag shows the local configuration, including which `ansible.cfg` is being used. Alternatively, use `ansible --version`.

3. On `bastion`, confirm that you can use `ssh` to access one or more internal servers, then exit the `ssh` session back to `bastion`:

TEXT

```
[user-company.com@bastion ~]$ sudo -i
[root@bastion ~]# export GUID=`hostname | awk -F"." '{print $2}'`
[root@bastion ~]# ssh app1.${GUID}.internal
Last login: Wed Sep 27 09:11:08 2017 from ip-192-199-0-140.ec2.internal
[ec2-user@app1 ~]$ exit
```

3. Set Up Client

In this section of the lab, you are required to set up your environment for Ansible development. You can use your own laptop or run these steps on the bastion host.

Pay careful attention to the prompts below—some tasks need to be performed on `bastion` before your laptop can communicate successfully with the target hosts.

On `bastion`, you enable your own non-`root` account for use with Ansible Engine and enable SSH access to the internal servers. As your user ID, **not** as `root`, you set up your environment.

1. Get the environment key from `/root/.ssh/${GUID}key.pem` on the bastion host, place it in your `~/.ssh/` directory, and set the correct permissions and file ownership:

```
TEXT
[user-company.com@bastion ~]$ export GUID=`hostname | awk -F"." '{print $2}'`
[user-company.com@bastion ~]$ mkdir ~/.ssh
[user-company.com@bastion ~]$ sudo cp /root/.ssh/${GUID}key.pem ~/.ssh
[user-company.com@bastion ~]$ sudo chown `whoami` ~/.ssh/${GUID}key.pem
[user-company.com@bastion ~]$ sudo chmod 400 ~/.ssh/${GUID}key.pem
```

2. Use `scp` or the Ansible `fetch` module to retrieve the above key from the bastion host to enable your laptop for SSH. Alternatively, use the following:

```
TEXT
[user-company.com@bastion ~]$ sudo cat /root/.ssh/${GUID}key.pem
# copy and paste the output into ~/.ssh/${GUID}key.pem on your laptop
[laptop ~]$ export GUID=<"GUID from email">
[laptop ~]$ sudo chmod 400 ~/.ssh/${GUID}key.pem
```

3. Test that you can access the `app1` instance, logging in as the `ec2-user`:

```
TEXT
[laptop or bastion ~]$ ssh -i ~/.ssh/${GUID}key.pem  ec2-
user@app1.${GUID}.example.opentlc.com
[ec2-user@ip-192-199-0-75 ~]$ exit
```

4. Make a copy that you can modify of the inventory file located under `/etc/ansible/hosts` on `bastion`:

```
TEXT
[user-company.com@bastion ~]$ cp /etc/ansible/hosts ~/myinventory.file
```

Copy it to your local machine if you are not using the bastion host.

5. If you are using your laptop, install `ansible` if it is not already installed.

If you want to use an inventory file other than the one provided in `/etc/ansible/hosts`, you need to specify it using the `-i myinventory.file` flag or set it in `ansible.cfg`.

6. Check that you can run ad hoc commands on your hosts:

```
TEXT
[laptop or bastion ~]$ ansible -i myinventory.file all -m ping -v
```

Sample Output

```
appdb1.GUID.internal | SUCCESS | rc=0 >>
 16:46:57 up 45 min,  1 user,  load average: 0.00, 0.01, 0.05

app2.GUID.internal | SUCCESS | rc=0 >>
 16:46:57 up 45 min,  1 user,  load average: 0.00, 0.01, 0.05

app1.GUID.internal | SUCCESS | rc=0 >>
 16:46:57 up 44 min,  1 user,  load average: 0.00, 0.01, 0.05

frontend1.GUID.internal | SUCCESS | rc=0 >>
 16:46:57 up 44 min,  1 user,  load average: 0.00, 0.01, 0.05

support1.GUID.internal | SUCCESS | rc=0 >>
 16:46:57 up 44 min,  1 user,  load average: 0.00, 0.01, 0.05
```

TEXT

4. Deploy Three-Tier Application Incorrectly

In this section, you complete the following steps to deploy the three-tier application the *wrong* way:

- Examine and prepare to run the playbook from the `bad-ansible` repository
- Run the playbook
- Clean up and reset the environment

The procedures below assume you are working on the bastion host. If you choose to work from your laptop, your output may differ.

4.1. Examine and Prepare Playbook

Currently, FinanceTech uses a single monolithic playbook to configure hosts and install HAProxy, Tomcat, and PostgreSQL on different systems.

1. Clone the `bad-ansible` repository from GitHub:

```
[laptop or bastion ~]$ git clone https://github.com/tonykay/bad-ansible.git
```

TEXT

The repository file on the bastion host is a good example to use when configuring the repositories on the other internal hosts.

2. On `bastion`, get the `open_three-tier-app.repo` file from the `/etc/yum.repos.d` directory:

```
[user-company.com@bastion ~]$ cp /etc/yum.repos.d/open_three-tier-app.repo bad-ansible/3tier-bad/
```
TEXT

Laptop users need to copy this file locally.

3. Review the playbooks and fix any issues before you run them.

Read the playbooks and plays very carefully and make sure you understand what they do. If you skip this step, it may hamper your progress in future labs.

4.2. Run `bad-playbook.yml` Playbook

1. Using the existing static inventory on `bastion`, run the `bad-playbook.yml` playbook and provide your GUID as an extra variable:

```
[laptop or bastion ~]$ ansible-playbook -i myinventory.file bad-ansible/3tier-bad/bad-playbook.yml -e "GUID=${GUID}"
```
TEXT

Make sure you set `GUID` or the application will fail.

2. If a task fails, investigate the playbook.

3. Validate that HAProxy is working by browsing to `http://frontend1.${GUID}.example.opentlc.com`:

```
[laptop or bastion ~]$ curl http://frontend1.${GUID}.example.opentlc.com
```
TEXT

Replace `${GUID}` with your GUID.

4.3. Clean Up and Reset Environment

1. Clean up the environment so you can test your refactored playbooks in the next sections:

```
[laptop or bastion ~]$ ansible-playbook -i myinventory.file  bad-ansible/3tier-bad/cleanup.yml
```
TEXT

- This step does the following:
  - Removes core packages including: `haproxy` , `tomcat` , and `postgresql`
  - Removes the `repo` file from `/etc/yum.repos.d`
  - Removes Tomcat's default `index.html` page

5. Refactor Existing Monolithic Playbook

Your job is to do the following:

- Break up the playbook into roles for better structure, reuse, and maintenance.
- Make sure the correct modules are in use and eliminate or reduce the use of modules such as `command` .
- Remove any manual steps.

Keep in mind that this on-premise deployment will soon be reused with one or more cloud providers.

5.1. Create Repository for Project

1. Create a `git` repository to hold your Ansible project.
2. Use the correct tools to build a best-practice structure for your project.

5.2. Refactor `bad-ansible`

The resulting playbooks need to run with informative messages, no errors, and be idempotent. The resulting deployment needs to also install, configure, and start the three-tier application environment.

1. Refactor the `bad-ansible` repository into your `good-ansible` repository using the following guidelines:
   - Adopt a consistent style and syntax throughout
     - Clean up the messages and feedback to make them informative and correct
   - Break up the playbook by introducing the use of roles

        - An interim step can involve breaking it into three playbooks using `include`
    - Eliminate any unnecessary tasks or plays
        - Use the correct modules for tasks
        - Eliminate unnecessary use of modules such as `command`
    - If appropriate, consider using Ansible Galaxy roles
    - Leverage templates for configuration files—for example:
        - `haproxy.cfg`
        - Tomcat's `index.html` - add server identity to output
    - Use variables to enhance portability and maintainability
    - Consider the use of handlers for post-configuration restarts, etc.
    - Use Ansible Vault to protect sensitive information
        - `open_three-tier-app.repo` - sensitive URL
        - Consider how to unlock vaults

During playbook and role development, you can roll back the three-tier application environment to provide a clean target for playbook runs:

```
[laptop or bastion ~]$ ansible-playbook -i myinventory.file  bad-ansible/3tier-bad/cleanup.yml
```
TEXT

2. Validate that the resulting playbooks and supporting roles run.
3. Validate that the three-tier application restarts after rebooting the lab environment.
    - Consider a simple playbook or ad hoc command to achieve this.
4. Create an Internet-accessible `git` repository such as GitHub, Bitbucket, or Gitlab.
    - It does not have to be public. Bitbucket and others provide free private repositories.

You use this `git` repository throughout the rest of the course, although one possible solution to this exercise is provided.

    - If working from your laptop, clone your repository onto `bastion` and confirm it runs there without changes.

The `good-ansible` GitHub repository contains one possible solution/refactoring of `bad-ansible`. It includes the use of roles, templates, Ansible Galaxy roles, `ansible-vault`, and more.

Strategies and approaches to refactoring are discussed in the Module 2 lab.

TEXT

```
[laptop or bastion ~]$ git clone https://github.com/tonykay/good-ansible.git
```