Configuration Management with Ansible

Raúl Estrada

Octubre 2020

Deployment with Ansible

We have covered the most fundamental features of Ansible. Let's now forget, just for a little while, about Docker and configure a complete deployment step using Ansible. We will run the calculator service on one server and the Redis service on the second server.

Deploying a web service

We prepare the calculator web service in three steps:

1. Configure the project to be executable.
2. Change the Redis host address.
3. Add calculator deployment to the playbook.

Adding calculator deployment to the playbook

Finally, we can add the deployment configuration as a new play in the `playbook.yml` file.

```
- hosts: web2
  become: yes
  become_method: sudo
  tasks:
  - name: ensure Java Runtime Environment is installed
    apt:
      name: default-jre
      state: present
  - name: create directory for Calculator
    file:
      path: /var/calculator
      state: directory
  - name: configure Calculator as a service
    file:
      path: /etc/init.d/calculator
      state: link
      force: yes
      src: /var/calculator/calculator.jar
  - name: copy Calculator
    copy:
      src: build/libs/calculator-0.0.1-SNAPSHOT.jar
      dest: /var/calculator/calculator.jar
      mode: a+x
    notify:
    - restart Calculator
  handlers:
  - name: restart Calculator
    service:
      name: calculator
      enabled: yes
      state: restarted
```

Let's walk through the steps we defined:

- **Prepare the environment**: This task ensures that the Java Runtime Environment is installed. Basically, it prepares the server environment, so that the calculator application would have all the necessary dependencies. With more complex applications, the list of dependent tools and libraries can be way longer.
- **Configure application as a service**: We would like to have the calculator application running as a Unix service, so that it will be manageable in the standard way. In this case, it's enough to create a link to our application in the `/etc/init.d/` directory.
- **Copy the new version**: The new version of the application is copied into the server. Note that if the source file didn't change, then the file won't be copied and therefore the service won't be restarted.
- **Restart the service**: As a handler, every time the new version of the application is copied, the service is restarted.

Running deployment

As always, we can execute the playbook using the `ansible-playbook` command. Before that, we need to build the calculator project with Gradle.

```
$ ./gradlew build
$ ansible-playbook playbook.yml
```

After the successful deployment, the service should be available and we can check it's working at `http://192.168.0.242:8080/sum?a=1&b=2`. As expected, it should return `3` as the output.

Ansible with Docker

As you may have noticed, Ansible and Docker address similar software deployment issues:

- **Environment configuration**: Both Ansible and Docker provide a way to configure the environment; however, they use different means. While Ansible uses scripts (encapsulated inside the Ansible modules), Docker encapsulates the whole environment inside a container.
- **Dependencies**: Ansible provides a way to deploy different services on the same or different hosts and let them be deployed together. Docker Compose has a similar functionality, which allows running multiple containers at the same time.
- **Scalability**: Ansible helps to scale services providing the inventory and host groups. Docker Compose has a similar functionality to automatically increase or decrease the number of running containers.
- **Automation with configuration files**: Both Docker and Ansible store the whole environment configuration and service dependencies in files (stored in the source control repository). For Ansible, this file is called `playbook.yml`. In the case of Docker, we have Dockerfile for the environment and docker-compose.yml for the dependencies and scaling.
- **Simplicity**: Both tools are very simple to use and provide a way to set up the whole running environment with a configuration file and just one command execution.

Benefits of Ansible

Ansible may seem redundant; however, it brings additional benefits to the delivery process:

- **Docker environment**: The Docker host itself has to be configured and managed. Every container is ultimately running on Linux machines, which needs kernel patching, Docker engine updates, network configuration, and so on. What's more, there may be different server machines with different Linux distributions and the responsibility of Ansible is to make sure the Docker engine is up and running.

- **Non-Dockerized applications**: Not everything is run inside a container. If part of the infrastructure is containerized and part is deployed in the standard way or in the cloud, then Ansible can manage it all with the playbook configuration file. There may be different reasons for not running an application as a container, for example performance, security, specific hardware requirements, Windows-based software, or working with the legacy software.
- **Inventory**: Ansible offers a very friendly way to manage the physical infrastructure using inventories, which store the information about all servers. It can also split the physical infrastructure into different environments: production, testing, development.
- **GUI**: Ansible offers a (commercial) GUI manager called Ansible Tower, which aims to improve the infrastructure management for the enterprises.
- **Improve testing process**: Ansible can help with the integration and acceptance testing and can encapsulate the testing scripts in a similar way that Docker Compose does.

Installing Docker

We can install the Docker engine using the following task in the Ansible playbook.

```
tasks:
- name: add docker apt keys
  apt_key:
    keyserver: hkp://p80.pool.sks-keyservers.net:80
    id: 9DC858229FC7DD38854AE2D88D81803C0EBFCD88
- name: update apt
  apt_repository:
    repo: deb [arch=amd64] https://download.docker.com/linux/ubuntu xenial
main stable
    state: present
- name: install Docker
  apt:
    name: docker-ce
    update_cache: yes
    state: present
- name: add admin to docker group
  user:
    name: admin
    groups: docker
    append: yes
```

```
- name: install python-pip
  apt:
    name: python-pip
    state: present
- name: install docker-py
  pip:
    name: docker-py
- name: install Docker Compose
  pip:
    name: docker-compose
    version: 1.9.0
```

The playbook looks slightly different for each operating system. The one presented here is for Ubuntu 16.04.

This configuration installs the Docker engine, enables the `admin` user to work with Docker, and installs Docker Compose with dependent tools.

Alternatively, you may also use the `docker_ubuntu` role as described here: `https://www.ansible.com/2014/02/12/installing-and-building-docker-with-ansible`.

When Docker is installed, we can add a task, which will run a Docker container.

Running Docker containers

Running Docker containers is done with the use of the `docker_container` module and it looks very similar to what we presented for the Docker Compose configuration. Let's add it to the `playbook.yml` file.

```
- name: run Redis container
  docker_container:
    name: redis
    image: redis
    state: started
    exposed_ports:
    - 6379
```

You can read more about all of the options of the `docker_container` module on the official Ansible page at: `https://docs.ansible.com/ansible/docker_container_module.html`.

Using Docker Compose

The Ansible playbook is very similar to the Docker Compose configuration. They even both share the same YAML file format. What's more, it is possible to use `docker-compose.yml` directly from Ansible. We will show how to do it, but first, let's define the `docker-compose.yml` file.

```
version: "2"
services:
  calculator:
    image: leszko/calculator:latest
    ports:
    - 8080
  redis:
    image: redis:latest
```

It is almost the same as what we defined in the previous chapter. This time we get the calculator image directly from the Docker Hub registry, and do not build it in `docker-compose.yml`, since we want to build the image once, push it to the registry, and then reuse it in every deployment step (on every environment), to make sure the same image is deployed on each Docker host machine. When we have `docker-compose.yml`, we are ready to add new tasks to `playbook.yml`.

```
- name: copy docker-compose.yml
  copy:
    src: ./docker-compose.yml
    dest: ./docker-compose.yml
- name: run docker-compose
  docker_service:
    project_src: .
    state: present
```

We first copy the docker-compose.yml file into the server and then execute `docker-compose`. As a result, Ansible creates two containers: calculator and redis.