Integrating Ansible with Jenkins in a CI/CD process

April 6, 2018 | Ricardo Zanini

The aim of this post is to demonstrate how to use Ansible for environment provisioning and application deployment in a Continuous Integration/Continuous Delivery (CI/CD) process using a Jenkins Pipeline.

Ansible is a powerful tool for IT automation and can be used in a CI/CD process to provision the target environment and to then deploy the application on it.

Jenkins is a well-known tool for implementing CI/CD. Shell scripts are commonly used for provisioning environments or to deploy apps during the pipeline flow. Although this could work, it is cumbersome to maintain and reuse scripts in the long run.

The purpose of using Ansible in the pipeline flow is to reuse roles and Playbooks for provisioning, leaving Jenkins only as a process orchestrator instead of a shell script executor.

The tools used to create the examples for this post are:

- Vagrant and libvirt to create the infrastructure for this lab
- SonarSource to bring up quality analysis to the CI/CD process
- Maven to set and deploy the Java project
- GIT for source code management and control
- Nexus is the repository for artifact binaries
- Jenkins to orchestrate the CI/CD pipeline flow
- And finally, Ansible to create all infrastructure for this lab and the to provision the application

The infrastructure architecture

Figure 1 illustrates the overall architecture for this lab. It has some elements of ALM (Application Lifecycle Management) to emulate a real-world scenario and apply our CI/CD demo pipeline flow.

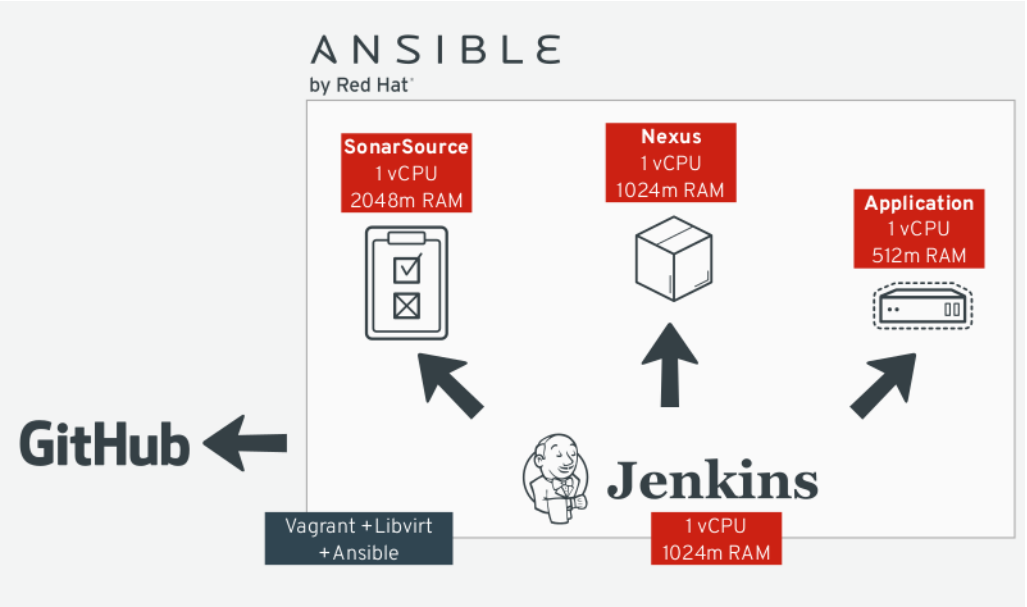

Figure 1 - Infrastructure architecture components overview

Figure 1 illustrates the following architectural elements:

- **Github** is where our project is hosted and where Jenkins will poll for changes to start the pipeline flow.
- **SonarSource** is our source code analysis server. If anything goes wrong during the analysis (e.g. not enough unit tests), the flow is interrupted. This step is important to guarantee the source code quality index.
- **Nexus** is the artifact repository. After a successful compilation, unit tests and quality analyses, the binaries are uploaded into it. Later those binaries will be downloaded by **Ansible** during the application deployment.
- The **Ansible Playbook**, which is a YAML file integrated in the application source code, deploys the **Spring Boot App** on to a CentOS machine.
- **Jenkins** is our CI/CD process orchestrator. It is responsible to put all the pieces together, resulting in the application successfully deployed in the target machine.

To put this infrastructure together, we built an **Ansible** Playbook using roles from the Ansible Galaxy community. More about this Playbook is discussed further in this article. If you are new to Ansible, check this article about how to get started. Spoiler alert: Ansible Galaxy will be your primary source to learn Ansible.

The environment for the virtual machines in this lab was managed by **Vagrant** with **libvirt**. Details about how this was done could be seen in the project Vagrant ALM at Github.

Example Pipeline Flow

Now that the tools and the architecture of this lab are well known, let's explore the CI/CD pipeline flow built for this lab. Note that the purpose of this pipeline is for demonstration only and may lack steps that a real-world CI/CD process might have.

Observe the figure below:

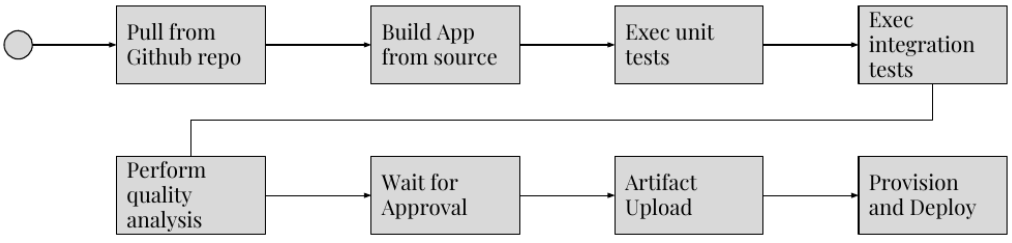

Figure 2 - The pipeline flow

The process starts by pulling the application source code from Github. The next thing to do is to update the project version according to the build number (e.g. my-fantastic-app-1.0.0-123). This way, we have a fingerprint of this build during deployment. Later, can be used as a metric and process control. After updating the project's version, **Jenkins** starts building the source code using Maven.

After a successful compilation, the pipeline will perform unit tests. If nothing goes wrong during the unit tests, the pipeline flow initiates the integration tests. In this case, the test framework creates all the infrastructure needed for the tests: an in-memory database with test data and a small web server to deploy the application. The integration tests are considered a success when all requests were validated against the application deployed in the test environment.

The output from unit and integration tests is a coverage report, which will be one of the artifacts used by Sonar server to generate quality metrics. The other one is the application source code. If the quality gate defined in the Sonar server is not met, the pipeline flow will fail.

If everything went fine, the flow stops its execution and waits for approval. The Jenkins server provides an interface for someone with the right permissions to manually promote the build. Figure 3 illustrates this process.

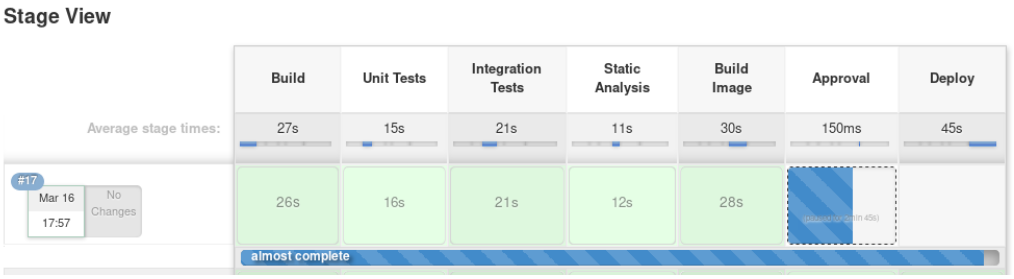

Figure 3 - Approval step example

After the approval, the pipeline continues to execute the flow and goes on to the next step, which is to upload the compiled artifact to the Nexus repository. Then we have a new application snapshot ready to be deployed.

Now it's time for Ansible to shine. The pipeline flow just sets the required parameters like the compiled artifact URL and the target host to execute the Ansible Playbook afterward. The Playbook is used to automate all target host configuration. In this lab, we'll use it to install Java and to prepare the environment to receive the Spring Boot Application as a service. The Playbook is going to install the application as a service and will poll its HTTP port until it receives a valid response to make sure the deployment was successful.

After the Ansible Playbook execution, the last step could be to send a notification to every stakeholder regarding the results of the new deployment via email or slack. This isn't implemented in our example pipeline yet, but I'm willing for pull requests!

Ansible Playbooks

Let's dig through the Playbooks used in this lab to get more details about what has been done until now. Ansible played a fundamental role twice in this lab. First, it automated all the infrastructure for this lab, then we used it as a tool to deploy our application through Jenkins pipeline.

Infrastructure provisioning

There's no secret in here. With a little knowledge about how Ansible works, you could create a lab (or even a production environment) using community Ansible roles like the ones used in this lab to create an ALM excerpt. The mindset behind this process was discussed in my other article: "A developer's shortcut to getting started with Ansible", please check it out for a comprehensive review of this process.

After designing the architecture, we searched for Ansible roles that could automate the actors of our design, like a Sonar and Jenkins server, for example. In the end, we've come up with the following Playbook:

```
---
- name: Deploy Jenkins CI
hosts: jenkins_server
remote_user: vagrant
become: yes

roles:
  - geerlingguy.repo-epel
  - geerlingguy.jenkins
  - geerlingguy.git
  - tecris.maven
  - geerlingguy.ansible

- name: Deploy Nexus Server
hosts: nexus_server
remote_user: vagrant
become: yes

roles:
  - geerlingguy.java
  - savoirfairelinux.nexus3-oss

- name: Deploy Sonar Server
hosts: sonar_server
remote_user: vagrant
become: yes

roles:
  - wtanaka.unzip
  - zanini.sonar

- name: On Premises CentOS
hosts: app_server
remote_user: vagrant
become: yes

roles:
  - jenkins-keys-config
```

Basically, we group the roles that make sense together in order to provide the desired infrastructure. Let's take a closer look at the Jenkins' provision:

```
- name: Deploy Jenkins CI
hosts: jenkins_server
remote_user: vagrant
become: yes

roles:
  - geerlingguy.repo-epel
  - geerlingguy.jenkins
  - geerlingguy.git
  - tecris.maven
  - geerlingguy.ansible
```

This task defines the host group where Jenkins will be deployed and describes the roles that we used to deploy the Jenkins server. We are going to need the EPEL repository for the CentOS to be enabled, the Jenkins installation itself and GIT, Maven and Ansible that are required for our pipeline. That's it. With barely 11 lines of code, we have a Jenkins server up and running prepared to start our CI/CD process.

To get this infrastructure ready, we took the advantage of the integration between Vagrant and Ansible. The vagrant file used in this lab could be seen in the Github vagrant-alm repository. Then with a simple "vagrant up" on the Vagrant file directory, we have the lab environment ready. More about how to prepare your machine to run Vagrant with libvirt could be seen in the docs.

Application deployment

The pipeline has been designed to prepare the application binaries, now called “artifact”, and to upload them in Nexus. The artifact can be reached in Nexus by an URL usually called Artifact URL. Ansible is also part of the pipeline and will receive the Artifact URL as the input for deployment. Thus, Ansible will be responsible not only for the deployment but also for the machine provisioning.

Let’s take a closer look at the Ansible Playbook that deploys our application on the target host:

```
---
- name: Install Java
hosts: app_server
become: yes
become_user: root
roles:
  - geerlingguy.java

- name: Download Application from Repo
hosts: app_server
tasks:
  - get_url:
      force: yes
      url: "{{ lookup('env','ARTIFACT_URL') }}"
      dest: "/tmp/{{ lookup('env','APP_NAME') }}.jar"
  - stat:
      path: "/tmp/{{ lookup('env','APP_NAME') }}.jar"

- name: Setup Spring Boot
hosts: app_server
become: yes
become_user: root
roles:
  - { role: pellepelster.springboot-role,
      spring_boot_file: "{{ lookup('env','APP_NAME') }}.jar",
      spring_boot_file_source: "/tmp/{{ lookup('env','APP_NAME') }}.jar",
      spring_boot_application_id: "{{ lookup('env','APP_NAME') }}"
  }
```

Details about this implementation can be seen in the demo application GitHub repository.

This Playbook will only perform three tasks:

1. Install Java based on a pre-defined role from Ansible Galaxy
2. Download the binaries from the Nexus repository based on the input from Jenkins
3. Set up the application as a Spring boot service again using a role from the community

Resources from the community were used to make things fast and to not reinvent the wheel. This Playbook is in the application repository, which means that the application knows how to deploy itself.

By using the Ansible Jenkins plugin, it was possible to call this Playbook from the pipeline by setting the variables required to execute it. Here’s an excerpt from the pipeline demonstrating how this is done:

```
def artifactUrl = "http://${NEXUS_URL}/repository/ansible-meetup/${repoPath}/${version}/${pom.artifactId}-${versior

withEnv(["ARTIFACT_URL=${artifactUrl}", "APP_NAME=${pom.artifactId}"]) {
  echo "The URL is ${env.ARTIFACT_URL} and the app name is ${env.APP_NAME}"

  // install galaxy roles
  sh "ansible-galaxy install -vvv -r provision/requirements.yml -p provision/roles/"

  ansiblePlaybook colorized: true,
  credentialsId: 'ssh-jenkins',
  limit: "${HOST_PROVISION}",
  installation: 'ansible',
  inventory: 'provision/inventory.ini',
  playbook: 'provision/playbook.yml',
  sudo: true,
  sudoUser: 'jenkins'
}
```

To make this happen, the target host must have the Jenkins user and its keys properly configured. This was done while provisioning the lab. All target hosts are CentOS machines with the Jenkins user and its key already set up. And guess what? There's a Ansible role to configure the user and its keys. This has been done because it's a prereq to have Ansible access the host somehow.

Tips and tricks

There are some tips that I would like to share to shorten your way towards using Ansible with Jenkins:

1. **Prepare the target host with the Jenkins user and its SSH keys**. The target host could be a pod on Red Hat OpenShift, a Virtual Machine, bare metal, etc. Doesn't matter. Ansible needs a way to connect to the host to perform its magic.
2. **Set the Jenkins user's private key on the credentials repository**. That way you can easily retrieve it on the pipeline code and send it as a parameter to the Ansible plugin.
3. Before running the deploy Playbook, **consider installing all required roles on the Jenkins server**. This could be done by performing a good old shell script run on the requirements file during the pipeline execution: "sh "ansible-galaxy install -vvv -r provision/requirements.yml -p provision/roles/"".
4. You may face some situations where **you need to deploy the application only on a specific host** (for a blue-green deployment, for example). You could do this by using the "-limit" parameter on the Ansible Playbook execution.

We've seen how Ansible could play an important role in a CI/CD pipeline flow. One could say that it was born to do this. That way the CI/CD pipeline won't worry about how to deploy an application or if the machine is properly provisioned; it just delegates to Ansible the deployment of the application.

One of Ansible's benefits is the ability to share and reuse the created roles among other projects and applications.

In our lab, we reused a bunch of roles from the community. With that, we could leverage the community power to keep the codebase getting better over the time. In your company, you could configure a repo for Ansible roles that the development and operations teams could contribute to creating a better code base, increasing the quality of your internal processes. That's what DevOps is all about!

Ricardo Zanini is a TAM in the Latin America region. He has expertise in integration, middleware and software engineering. Ricardo has been studying software quality since his first professional years and today helps strategic customers achieve success with Red Hat Middleware products and quality processes. Find more posts by Ricardo Zanini at https://www.redhat.com/en/blog/authors/ricardo-zanini

A Red Hat Technical Account Manager (TAM) is a specialized product expert who works collaboratively with IT organizations to strategically plan for successful deployments and help realize optimal performance and growth. The TAM is part of Red Hat's world-class Customer Experience and Engagement organization and provides proactive advice and guidance to help you identify and