# Alpaquita Linux
## Using Ansible to deploy Java applications

Alpaquita Linux
Revision 1.0
January 2024

bellsoft

BellSoft software contains open source software. Additional information about third party code is available at https://bell-sw.com/third_party_licenses. You can also get more information on how to get a copy of source code by contacting info@bell-sw.com.



The use of any software product referenced in this document is governed by the applicable license agreement, which is not modified in any way by the terms of this notice.

# Contents

# 1. Ansible overview

Ansible is a popular automation tool for managing IT systems.

This document demonstrates several Ansible features for deploying a Spring Boot Java application on an Alpaquita Linux instance.

- A *control node* is a system on which Ansible is installed.
- *Managed nodes* are systems that Ansible controls.

Ansible initiates SSH connections from the control node to each managed node and executes necessary configuration operations on the managed node. These operations may include file modifications, starting or stopping services, adding users, and much more.

Configuration operations are written to YAML files called playbooks. Each *playbook* contains an ordered list of plays. Each *play* contains an ordered list of tasks. Each *task* lists one or more Ansible *modules* that define what operations should be performed.

Modules are grouped into *collections*, and Ansible ships with a large number of collections and modules to cover many scenarios.

In addition to regular tasks, Ansible also provides *handlers*, that is tasks, which are executed only when notified.

In comparison to scripting, Ansible is essentially a state engine, and all its tasks are idempotent, so playbooks are written with a declarative approach.

For example, you can see a playbook with two plays below. Each contains only one task. The first one uses the `ansible.builtin.ping` module to ping systems in the `java_servers` group. The second one uses the `ansible.builting.user` module to declare that all systems in the same group should have a user with the name `user` and the comment `A user`.

```
- name: First play
  hosts: java_servers
  tasks:
    - name: Ping
      ansible.builtin.ping:
- name: Second play
  hosts: java_servers
  tasks:
    - name: Add a user
      ansible.builtin.user:
        name: user
```

```
        comment: A user
```

Ansible changes the configuration of the managed node only if the user does not exist or its parameters are different from those set in the playbook, that is when the actual state of the node differs from the declared state in the playbook.

# 2. Prerequisites

Perform the following steps before you proceed to the next sections.

Set up two hosts: one is a control node and the other is a managed node.

Install Ansible on a control node that can be any UNIX-like system with Python 3.9 or newer installed.

On Fedora, you can install Ansible as follows:

```
sudo dnf install ansible
```

On Ubuntu, use the following command:

```
sudo apt-get install ansible
```

For other types of systems, refer to their documentation and the official Ansible installation guide.

For the other host, the managed node, we need an Alpaquita Linux instance. It can be any type of instances, such as bare-metal, virtual machine or an instance in a cloud as long as the following requirements are met:

- A user with admin privileges is created;
- SSH access is set up from the control node to this instance using admin user credentials;
- A Python interpreter is installed.

Visit the Alpaquita download page and choose the appropriate version of Alpaquita.

In our example, the Alpaquita instance is a virtual machine deployed from an iso. During installation, we assigned the `192.168.71.100` ip to it. The user with admin privileges is `admin`. A Python interpreter was installed by connecting to the instance via SSH as `admin` and executing the following command:

```
sudo apk add python3
```

You will be creating some files during the course of this document. We recommend keeping the files in one place. Create a separate directory on the control node, for example: `alpaquita-ansible`. All the next steps are performed on the control node inside this directory.

# 3. Preparing an inventory

The number of managed nodes can be quite large, and specifying them all in the command line is cumbersome. Ansible provides an *inventory* file that contains logically organized sets of nodes.

Let's create an inventory file named `inventory` with the following content:

```
[alpaquita_nodes]
alpaquita-node ansible_host=192.168.71.100 ansible_user=admin

[alpaquita_nodes:vars]
ansible_python_interpreter=/usr/bin/python3
```

In this file we declare a group of nodes named `alpaquita_nodes` that contains a single node with the `alpaquita-node` id with information regarding its network address and the name of the user.

Section `[alpaquita_nodes:vars]` specifies variables for all members of the corresponding group.

Substitute the address and username with your values.

Now, after we have created an inventory, we can verify that Ansible connects to our Alpaquita instance by *pinging* all nodes in the `alpaquita_nodes` group.

```
ansible -i inventory -k -b -K -m ping alpaquita_nodes
```

- `-k` instructs Ansible to ask for an SSH password;
- `-b` option tells Ansible that the user we are connecting as is not root, and in order to gain admin privileges it should use `sudo` on the managed node, and the `-K` option makes Ansible ask for the sudo password.

# 4. Building a Java application

In this document we utilize the Spring PetClinic Sample Application Java application.

Build it on any machine with JDK Java 17 or later as follows:

```
git clone https://github.com/spring-projects/spring-petclinic.git
cd spring-petclinic
./mvnw package
```

After the application is ready, save the generated `target/*.jar` file as `spring-petclinic.jar` in the `alpaquita-ansible` directory on the control node.

# 5. Installing Java

To make Java applications work on our Alpaquita instance, install a JRE.

Alpaquita repositories provide Liberica JRE, to install it create a playbook file named `playbook.yaml` with a single task.

```
- name: Alpaquita setup
  hosts: alpaquita_nodes
  tasks:
    - name: Install Liberica JRE
      community.general.apk:
        name: liberica17-lite-jre-no-deps
```

In Alpaquita, packages are managed by APK, so you can use the `community.general.apk` Ansible module for package installation.

Execute the playbook `playbook.yaml` with the following command:

```
ansible-playbook -b -k -K -i inventory playbook.yaml
```

This command is the *common* command that we will use to execute the playbook later in this document.

# 6. Creating an application user

PetClinic is a web application. Running web applications under a user with admin privileges can lead to security issues, therefore we will create a special user and substitute our existing `admin` user with the newly created one.

Add the following tasks to `playbook.yaml` to create a new user:

```
- name: Install shadow package
  community.general.apk:
    name: shadow
- name: Add petclinic group
  ansible.builtin.group:
    name: petclinic
- name: Add petclinic user
  ansible.builtin.user:
    name: petclinic
    group: petclinic
    comment: PetClinic
    home: /home/petclinic
    create_home: true
    shell: /sbin/nologin
    password: !
```

We install the `shadow` package, because it provides `groupadd` and `useradd` utilities that are not present in the default Alpaquita installation.

Setting the shell to `/sbin/nologin` and password to `!` prohibits interactive login as the `petclinic` user.

Apply the configuration by executing the playbook. The `petclinic` user and group should be created in the system.

# 7. Deploying a Spring Boot application

Add the following task to the playbook to copy the application's jar to the Alpaquita system:

```
- name: Copy jar
  ansible.builtin.copy:
    src: spring-petclinic.jar
    dest: /home/petclinic
    owner: petclinic
    group: petclinic
    mode: 0444
```

After that, execute the playbook.

Now we can manually start the application. Login to the Alpaquita system as the `admin` user and execute the following command:

```
sudo -u petclinic java -jar /home/petclinic/spring-petclinic.jar
```

Now open the http://192.168.71.100:8080 URL in a web browser to see that the application is actually running.

Stop the `spring-petclinic.jar` application before proceeding to the next section.

# 8. Converting the application into a service

Starting applications as described in the previous section can be useful sometimes, but you can automate such routine operations. Convert the manual startup operations into a service and let a *service manager* be responsible for starting and stopping our applications when needed.

Alpaquita uses OpenRC to manage services. Services are defined in shell script files.

See the OpenRC documentation and Alpaquita Linux: Setting up OpenRC init system for additional information on OpenRC services.

In this case, we create a definition of a new service in `spring-petclinic-service.sh` file.

```
#!/usr/sbin/openrc-run

name="PetClinic service"
command="/usr/bin/java"
command_args="-jar /home/petclinic/spring-petclinic.jar"
command_background=true
pidfile="/run/$RC_SVCNAME.pid"
command_user="petclinic:petclinic"
```

Update the playbook and add tasks for copying the service file to the correct location and ensuring that the new service is running and automatically starts on system boot.

```
- name: Copy the service file
  ansible.builtin.copy:
    src: spring-petclinic-service.sh
    dest: /etc/init.d/petclinic
    owner: root
    group: root
    mode: 0755
- name: Enable petclinic service
  ansible.builtin.service:
    name: petclinic
    enabled: true
    state: started
```

After executing the playbook, open http://192.168.71.100:8080 in a browser to see that the application is running. If you reboot the Alpaquita instance, the application starts automatically.

# 9. Handling application updates

If you update `spring-petclinic.jar` with a new version and execute the playbook, the `Copy jar` task updates the configuration of the system by copying a new version of the file. However, the `petclinic` service is not restarted and runs the previous version of the application.

To update the `.jar` file, we introduce a new handler to restart the service and make the `Copy jar` task notify this handler after it has updated system configuration.

The updated playbook should look like the following (with irrelevant parts omitted):

```
- name: Alpaquita setup
  hosts: alpaquita_nodes
  tasks:
    <...>
    - name: Copy jar
      ansible.builtin.copy:
        src: spring-petclinic.jar
        dest: /home/petclinic
        <...>
      notify:
        - Restart petclinic service
    <...>
  handlers:
    - name: Restart petclinic service
      ansible.builtin.service:
        name: petclinic
        state: restarted
```

To verify the update procedure, go to the directory with PetClinic sources, update the image of pets displayed on the application welcome page with a new picture, and build a new jar file.

```
curl
https://upload.wikimedia.org/wikipedia/commons/4/47/PNG_transparency_demonstrat
ion_1.png \
  -o ./src/main/resources/static/resources/images/pets.png
git add src/main/resources/static/resources/images/pets.png
git commit -m 'Updated pets.png'
rm -rf target
./mvnw package
```

Copy the generated jar file to `spring-petclinic.jar` and execute the playbook. Ansible notifies that

it is copying the new file and restarting the service. Refreshing the page in the browser also shows the new image (depending on your browser, it may be necessary to perform the "force refresh" of the page).

# 10. Conclusion

This document only briefly describes a few Ansible features, but it can help you set up a system for some simple use cases.

If there is a requirement to have a system with an identical configuration, all you need to do is set up a new system and add a new line with access information to `inventory`.

You can go further and also perform *provisioning* of new systems with Ansible. For example, the Ansible ships AWS, Azure and, GCP modules to work with cloud resources, therefore you can make Ansible create your virtual machines in the cloud and configure them the way you need.

For more information, refer to the official Ansible documentation.

# Alpaquita Linux
## Using Ansible to deploy Java applications

bellsoft