For this recipe, you are going to need a Jenkins server to host the build and a tomcat server to act as the app server plus a repository to hold the example code. Ensure that your Jenkins server also has the Ansible installed;

The following steps will demonstrate how to use Jenkins, Ansible and Tomcat to deploy a very simple test application:

1. **Use the following command to create a new directory structure:**

**$ mkdir -p testapp/WEB-INF**

1. **Next, edit a new file under testapp/WEB-INF called web.xml and insert the following content:**

<web-app></web-app>

This is a very simple web app configuration file.

Save this file and create another file called index.jsp and insert the following code:

<html>

<head>

<meta http-equiv="refresh" content="1">

</head>

<body>

<H1>Test App</H1>

<%= date = new java.util.Date() %>

<p>Greetings, this is the first version of our test app. The time is currently: <%= date %></p>

</body>

</html>

This is a remarkably simple application, whose only role in life is to print out a pithy greeting, the date and time, use the HTML META tag to force the browser to refresh the page; however, it's perfectly suited to demonstrate the power of parallel deployment.

1. **Next, we are going to create a new Ansible playbook to perform our deployment. Create a new directory to hold our playbook using the following commands**:

$ mkdir -p appdeploy/inventory

$ mkdir -p appdeploy/group\_vars/appdeploy

This creates a basic structure to house our Ansible code, including a place to hold our inventory and variables.

1. **Next, we're going to create our Ansible inventory and create a new file called appdeploy/inventory/appinventory file and insert the following code:**

[appdeploy]

<< tomcatserver >>

Where << tomcatserver >> is the name/ip address of your tomcat server.

1. **Now that we have our inventory, we are ready to create our playbook. Create a new file called appdeploy.yml under the appdeploy folder and insert the following content:**

- hosts: appdeploy

gather\_facts: false

sudo: true

tasks:

- name: Deploy App

copy: src=/var/lib/jenkins/jobs/workspace/tomcat\_test /testapp.war dest="/usr/local/apache-tomcat-8.0.23/webapps/testapp##{{ buildnum }}.war" owner=tomcat group=tomcat

This is a simple playbook. First, we declare that this will only run against the servers that are tagged as appdeploy hosts. We are also declaring that this playbook does not need to gather facts. Not gathering facts can be a good way to speed up a playbook and is an excellent practice when you know that your Ansible code does not make use of them, as it skips a relatively expensive fact of gathering task. As usual, this is a trivial time saver against one host but when you scale out to hundreds of hosts, it can make a difference.

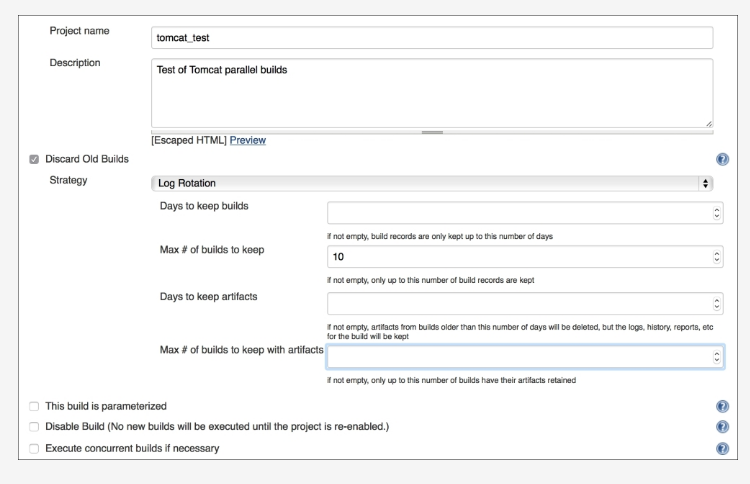
Next, we declare the tasks that we are going to carry out. In our case, we are only performing a single task, copying a war file to the remote host. Notice the {{ buildnum }} variable; this is going to be our link between Ansible and Jenkins and it will be explained later in the recipe.

1. **In the appdeploy/group\_vars/appdeploy directory, create a new file called main.yml and insert the following code:**

This is a simple kludge to ensure that the directory is added if you are using a Git repository. We are going to fill the content of this file by dynamically using Jenkins but Git has a nasty habit of not adding empty directory structures. This ensures that it will be included when we check it in and push it to our remote.

Once you have finished editing your files, create, check in, and push your code to your repository

1. Now, we can create our new Jenkins job. Log on to your Jenkins server and create a new item called tomcat\_test and ensure that it's set as a freestyle project. Once created, we can set the basic job options. We want to ensure that our job doesn't fill the disk of our Jenkins server, which is a real and immediate issue with builds that can generate large artifacts; ensure that your build matches the options listed in the following screenshot:



As you can see, I'm limiting the job to only keeping 10 builds at a time; you should edit this to suit your particular needs.

1. Now Configure the source code repository
2. it's time to get on and construct our build steps. First, we're going to use the jar command to create our Java archive. Add a new execute shell build step and ensure that it has the following command:

mv chapter7/testapp/\* . && jar cf testapp.war WEB-INF/index.jsp

This is a simple command that takes the checked out code and moves it into the working directory and uses the jar command to create the archive.

1. Next, we need to create the variable that will inform Ansible, which builds the application we are pushing. Create a new execute shell build step and add the following content:

echo "buildnum: $BUILD\_NUMBER" > chapter7/appdeploy/group\_vars/appdeploy/main.yml

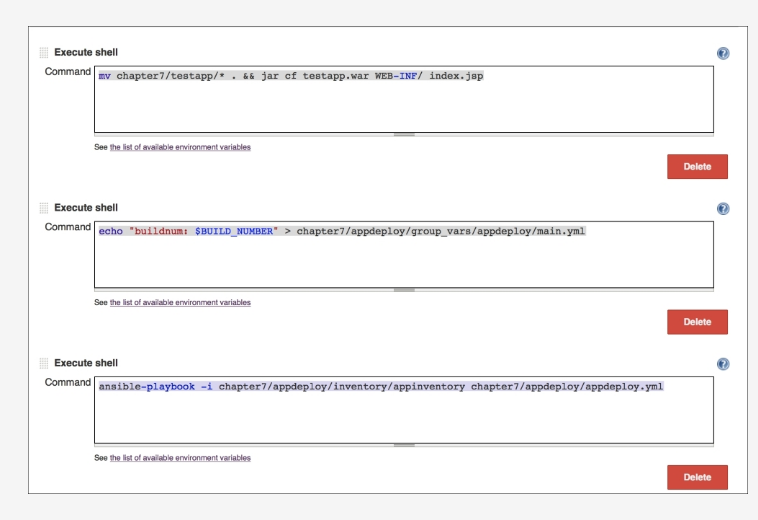
1. This command makes use of one of the environment variables that Jenkins sets as part of its build. These environment variables are astonishingly powerful, as they allow you to act on the output of your build.

You can find the list of available variables at your Jenkins server at the /env-vars.html/ URL; you can also act on other environment variables that you set yourself.

1. Finally, we can trigger our Ansible build. Add another execute shell build step and add the following command:

ansible-playbook -i chapter7/appdeploy/inventory/appinventory chapter7/appdeploy/appdeploy.yml

This invokes Ansible to use your inventory and playbook.



Optional Step

1. Before we can run our project, we need to make some changes to our servers. As our Ansible code is being run as the Jenkins user we need to ensure that it has access to the target server and it has a sudo access on each server. If you are using Ansible to control the tomcat servers, then you can add a Jenkins user and key via Ansible; see [Chapter 5](https://www.packtpub.com/mapt), **Automation with Ansible**, for an example. You will also need to add sudo access; this can be done by adding the following task into the Role that manages your user:

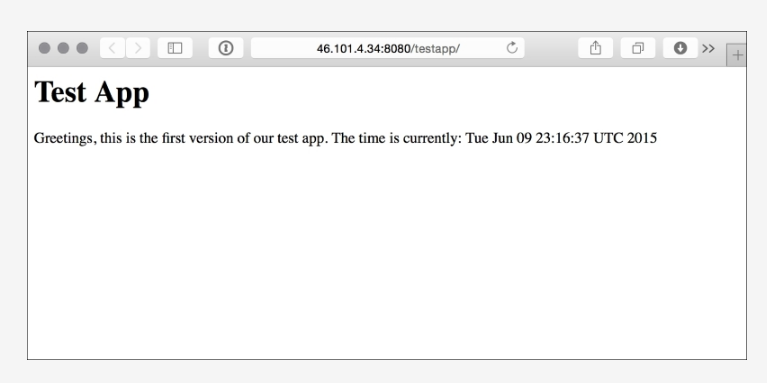
- name: Add Jenkins Sudo access

lineinfile: dest=/etc/sudoers state=present line='jenkins ALL=(ALL) NOPASSWD:ALL' validate='visudo -cf %s'

This code uses the Ansible lineinfile module to insert a new rule into the sudoers file; it will also use the visudo command to double check that the new rule is valid.Regardless of how you achieve it, ensure that your Jenkins user has both access to your target servers and a sudo access.

1. We're now ready to run our Jenkins build.

Open your browser, and point it at your test application. You should see something similar to the following screenshot:



If you see this, then excellent; your automated build has succeeded in taking a code from a developer and deploying it to a target machine without any manual interactions.

1. Now that we have deployed the first version of our code, it's time to make an amendment and deploy the second version. Keep your browser window open in the background and open up the code in testapp/index.jsp and edit it to resemble the following code example:

<html>

<head>

<meta http-equiv="refresh" content="1">

</head>

<body>

<H2>Test App</H2>

<%= date = new java.util.Date() %>

<strong>Greetings, this is the second version of our test app.</strong>

<p>The time is currently: <%= date %></p>

</body>

</html>

As you can see, this a slight tweak to our original code. Adding the maximum amount of style that a DevOps engineer generally decorates their apps with. Once you have made the tweak, save the code and check-in to your repository and push to your remote.