Automating your Tomcat with Ansible

Coty Sutherland
What will be covered

- Who am i?
- Brief Overview of Ansible
- Problem
- Solution
- Demo
- Potential improvements
- Thoughts and/or Questions?
- Thanks!
Who am i?

- Coty Sutherland
- Software Engineering Manager @ Red Hat
- Java Developer
- Tomcat committer since 2016
- Fedora tomcat package maintainer since 2015
- Husband and father of three
- East Coast, USA
Brief Overview of Ansible

“Ansible is a suite of software tools that enables infrastructure as code. It is open-source and the suite includes software provisioning, configuration management, and application deployment functionality.” - Wikipedia

It uses no agents and no additional custom security infrastructure, so it's easy to deploy - and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.
Why Ansible?

We’re not pushing this because it’s our product, we acquired Ansible because we think it’s the best automation solution on the market.

We like drinking our own champagne...and better product integration.

Ansible is outside of the JVM (unlike ant or maven) and provides a more standardized solution that can be applied across multiple applications.
Throughout this presentation, I’ll mention JWS, which is the product containing the Red Hat build of Apache Tomcat.

Product name could be better, but here we are :) 

- Apache Tomcat is the main component
- Vault Extension for Tomcat
- Mod_cluster
- Other small components extending and supporting Apache Tomcat
Problem

- Managing lots of instances of Tomcat is difficult and costly.
  - Defining and verifying deployment strategies
  - Technical debt incurred by rolling your own solution
- A specific issue I’ll focus on here is about how testing new Apache Tomcat releases is difficult and costly too.
  - Manual
  - Scripted/some automation solution
Solution

YOU GET AUTOMATION

AND YOU GET AUTOMATION
Solution, cont’d.

Automation with Ansible makes things easier!

Define and verify deployment strategies with Ansible’s declarative syntax (playbooks).

Repeatable state/status synchronization.

Testing your application on new Apache Tomcat releases is faster, easier, and repeatable using Ansible.
Introducing the JWS Ansible Collection...
Main JWS collection features

Easily install and configure JWS instances

Automatically configure product features, like mod_cluster

Uniform deployment strategies

Collection owner/maintainer is responsible for automation management rather than doing it yourself

By default, running the playbook will setup a basic Tomcat installation running on port 8080 with no applications deployed.
Overview

The template for the server.xml.j2 (and other conf files) covers the most basic use case of the server.

You can use your own template files if what we provided doesn’t work for you.

Full table table of options is documented within the role’s readme: https://github.com/ansible-middleware/jws/tree/main/roles/jws

### Service configuration

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>jws_apps_to_remove</td>
<td>Comma separated list of apps to undeploy</td>
<td>docs.ROOT.examples</td>
</tr>
<tr>
<td>jws_catalina_base</td>
<td>Tomcat catalina base env variable</td>
<td>{{ lookup('env', 'CATALINA_BASE') }}</td>
</tr>
<tr>
<td>jws_conf_properties</td>
<td>Path for tomcat configuration</td>
<td>./conf/catalina.properties</td>
</tr>
<tr>
<td>jws_conf_policy</td>
<td>Path for tomcat policy configuration</td>
<td>./conf/catalina.policy</td>
</tr>
<tr>
<td>jws_conf_logging</td>
<td>Path for logging configuration</td>
<td>./conf/logging.properties</td>
</tr>
<tr>
<td>jws_conf_context</td>
<td>Relative path to context.xml</td>
<td>./conf/context.xml</td>
</tr>
<tr>
<td>jws_conf_server</td>
<td>Relative path to server.xml</td>
<td>./conf/server.xml</td>
</tr>
<tr>
<td>jws_conf_web</td>
<td>Relative path to web.xml</td>
<td>./conf/web.xml</td>
</tr>
<tr>
<td>jws_conf_templates_context</td>
<td>Template to use for context.xml</td>
<td>templates/context.xml.j2</td>
</tr>
<tr>
<td>jws_conf_templates_server</td>
<td>Template to use for server.xml</td>
<td>templates/server.xml.j2</td>
</tr>
<tr>
<td>jws_conf_templates_web</td>
<td>Template to use for web.xml</td>
<td>templates/web.xml.j2</td>
</tr>
</tbody>
</table>
server.xml template syntax

The templates are Jinja2 syntax.

Here is an example of the configuration template for the server.xml portion that sets up the HTTPS connector for Tomcat.
Example use case: testing tomcat releases

Four main parts to this playbook:

- Get sources and run unit tests
- Download binary distro
- Configure and start Tomcat
- Deploy and test application
- name: "Download Apache Tomcat sources and run the unit test suite"
  block:
    - name: "Install required dependencies"
      include_tasks: fastpackage.yml
    vars:
      package_name: "{{ item }}"
    loop:
      - zip
      - unzip
      - ant
    - name: "Download source zip from: {{ tomcat_src_download_url }}"
      ansible.builtin.get_url:
        url: "{{ tomcat_src_download_url }}"
        dest: "{{ jws_install_dir }}"
        validate_certs: no
    - name: "Unzip source zip in {{ jws_install_dir }}"
      ansible.builtin.unarchive:
        remote_src: yes
        src: "{{ jws_install_dir }}/{{ tomcat_zip_filename }}"
        dest: "{{ jws_install_dir }}"
        creates: "{{ tomcat_source_home }}"
    - name: "Run unit tests"
      ansible.builtin.shell:
        # exclude tests that depend on openssl version a lot and only check nio
        cmd: "ant -DExecute.test.apr=false -DExecute.test.nio2=false -Dtest.exclude="**/TestCipher.java, **/TestOpenSsLCipherConfigurationParser.java" test > build.log 2>&1"
        chdir: "{{ tomcat_source_home }}"
        ignore_errors: True
        register: test_output
    - name: "Output failed tests"
      block:
        - name: "grep FAILED from logs"
          ansible.builtin.shell:
            cmd: "grep -A1 FAILED {{ tomcat_source_home }}/output/build/logs | grep -oe TEST-.*\:\"\.\"\""
            register: grep_output
        # We can change this to a fail if you want the playbook to stop execution when any tests fail
        - name: "Print grep output"
          ansible.builtin.debug:
            msg: "Check the following logs for failures: {{ grep_output.stdout_lines | join('', ',') }}"
            when: test_output.rc != 0
            when: "un unittests"
            when: test_output.rc != 0
        # Since the binary doesn't exist on the CON yet, we must manually download from the staging area
        - name: "Download binary zip from: {{ tomcat_bin_download_url }}"
          ansible.builtin.get_url:
            url: "{{ tomcat_bin_download_url }}"
            dest: "{{ jws_install_dir }}"
            validate_certs: no
tasks:
  - name: "Include role to unzip and configure Apache Tomcat, then smoke test (validate)"
    ansible.builtin.include_role:
      name: jws

  - name: "Deploy test application and make some requests"
    block:
      - name: "Deploy test webapp"
        ansible.builtin.copy:
          src: "testapp"
          dest: "{% jws_home %}/webapps/"
          mode: 0644

      # This will cause systemd to start Tomcat
      - name: "Force all notified handlers to run at this point, not waiting for normal sync points"
        ansible.builtin.meta: flush_handlers

  - name: "Wait for Tomcat HTTP port to be available"
    ansible.builtin.wait_for:
      port: 8080

  - name: "Verify webapp is responding as expected"
    ansible.builtin.uri:
      url: "http://localhost:8080/testapp"
      return_content: yes
      register: this
      failed_when: "'Hello!' not in this.content"
      until: this.status == 200
      retries: 3
      delay: 5
Demo!

```bash
[csuher@live tomat-release-test-playbook]$ # check to ensure /opt is empty
[csuher@live tomat-release-test-playbook]$ ll /opt/apache-tomcat-

[17900] v 7 root root 12 Sep 26 10:22 /opt/apache-tomcat-9.0.65-src
[csuher@live tomat-release-test-playbook]$ ll /opt/apache-tomcat-9.0.65-src/output/build
Total 4

[csuher@live tomat-release-test-playbook]$ ps aux | grep tomcat
[csuher@live tomat-release-test-playbook]$ ps aux | grep tomcat

[csuher@live tomat-release-test-playbook]$ # execute the playbook
[csuher@live tomat-release-test-playbook]$ ansible-playbook playbook.yml

[WARNING]: provided host list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

PLAY [Test Apache Tomcat Release In Progress] *********************************************************

TASK [Gathering Facts] ************************************************************************************************************
ok: [localhost]

TASK [Install required dependencies] ***********************************************************************************************
ok: [localhost]

TASK [Check arguments] **********************************************************
ok: [localhost]

TASK [Test if package zip is already installed] **********************************************************
ok: [localhost]

TASK [Check arguments] **********************************************************
ok: [localhost]

TASK [Test if package unzip is already installed] *********************************************************
ok: [localhost]

TASK [Check arguments] **********************************************************
ok: [localhost]

TASK [Test if package out is already installed] ***********************************************************
ok: [localhost]


[csuher@live tomat-release-test-playbook]$ ll /opt/apache-tomcat-

[17900] v 7 root root 12 Sep 26 10:22 /opt/apache-tomcat-9.0.65-src
```
Potential Improvements

- Provide a major version and automatically pull and test available releases for that major version
- Add logic to role in collection rather than a unique playbook
- Add property to check for development releases (after we include the logic into the role)
- Startup with startup.sh rather than requiring systemd
- Do something with the test output!
- Check file hashes and keys after downloading
- Fail if a test in the test suite fails
- Set variables to skip tests, etc
- Add a small cluster to test on in our validation logic
What do you think? Interested in using or contributing?

For using: https://galaxy.ansible.com/middleware_automation/jws

For contributing: https://github.com/ansible-middleware/jws

Issues tracked on GitHub!
Questions?

Email: csutherl@apache.org

Twitter: @CotySutherland

LinkedIn: https://www.linkedin.com/in/cotysutherland/
Thanks!