1. Check connectivity to the targets

First, I wrote a playbook to check connectivity to my targets:

---

- name: Check Connectivity and Report

hosts: nodes

gather\_facts: false

tasks:

- name: 01 - Test Connectivity

ansible.builtin.ping:

register: connectivity

ignore\_unreachable: true

- name: 02 - Save summary of connectivity check

ansible.builtin.set\_fact:

summary: "{{ (summary | default([])) + [ item + ';' + \_result ] }}"

vars:

\_result: "{{ (hostvars[item]['connectivity']['msg'] | default('OK')).splitlines() | join() }}"

loop: "{{ ansible\_play\_hosts }}"

delegate\_to: localhost

run\_once: true

- name: 03 - Show result

ansible.builtin.debug:

msg: "{{ summary }}"

delegate\_to: localhost

run\_once: true

- name: 04 - Save result to csv file

ansible.builtin.copy:

content: "{{ (summary | sort | join('\n')) + '\n' }}"

dest: /tmp/connectivity\_test.csv

delegate\_to: aapwork

run\_once: true

...

The playbook runs against all my nodes, and I explicitly set gather\_facts to false because I want to accomplish the connectivity test in a task with a special flag (ignore\_unreachable).

Skip to the bottom of list

Image

IT Automation ebook

Some comments about the tasks:

01 - Test Connectivity: ignore\_unreachable is set to true. Without this, the playbook would not execute the remaining tasks for this node. Notice that the next tasks run on localhost, but that is all I need to use the connectivity test's results for my summary.

02 - Save summary of connectivity check: This executes after all nodes are tested. I run a loop based on ansible\_play\_hosts (an Ansible magic variable containing a list of all hosts processed in this playbook). For each host, I add an element into the array/list named summary. I used some Jinja2 filters to handle cases where a line feed appears in the output. This summarization task includes:

delegate\_to = localhost runs on the localhost (AAP controller or Ansible controller).

The run\_once = true loop processes the list of hosts, but I only invoke the task once (instead of running the loop multiple times).

03 - Show result: This is a simple display of the array/list accumulated in the previous task. Also, it's executed only once and on the localhost. (And yes, these two tasks could be coded as a block.)

04 - Save result to a CSV file: The last task dumps the array/list containing the summary to a file, which is an external server in my example. Here are some important aspects:

I executed this in my AAP, so the localhost is my Execution Environment. This is why I want to write the file to a server I can connect to later to retrieve the output file. Saving a file and trying to retrieve it from an EE would require additional steps, which are not necessary for this use case.

If you run this playbook from the command line, it is OK to use localhost as the delegated host in this task because it is easy to get the file manually.

The Jinja2 templates sort the output and convert each list item to a line in the

**Troubleshooting the controller**

**27.1. Error logs**

The controller server errors are logged in /var/log/tower. Supervisors logs can be found in /var/log/supervisor/. Nginx web server errors are logged in the httpd error log. Configure other controller logging needs in /etc/tower/conf.d/.

Explore client-side issues using the JavaScript console built into most browsers and report any errors to Ansible via the Red Hat Customer portal at <https://access.redhat.com/>.

**27.2. sosreport**

The sosreport is a utility that collects diagnostic information for Support to be able to use to analyze and investigate the issues you report. To properly provide Technical Support this information, refer to the [Knowledgebase article for sosreport](https://access.redhat.com/solutions/3592) from the Red Hat Customer portal to perform the following procedures:

1. [Install the sosreport](https://access.redhat.com/solutions/3592#install) utility.
2. [Generate an sosreport](https://access.redhat.com/solutions/3592#command).
3. Provide the sosreport to Red Hat Support.

**27.3. Problems connecting to your host**

If you are unable to run the helloworld.yml example playbook from the Quick Start Guide or other playbooks due to host connection errors, try the following:

* Can you ssh to your host? Ansible depends on SSH access to the servers you are managing.
* Are your hostnames and IPs correctly added in your inventory file? (Check for typos.)

**27.4. Unable to login to the controller via HTTP**

Access to the controller is intentionally restricted through a secure protocol (HTTPS). In cases where your configuration is set up to run a controller node behind a load balancer or proxy as “HTTP only”, and you only want to access it without SSL (for troubleshooting, for example), you must add the following settings in the custom.py file located at /etc/tower/conf.d of your controller instance:

SESSION\_COOKIE\_SECURE **=** **False**

CSRF\_COOKIE\_SECURE **=** **False**

Changing these settings to False will allow the controller to manage cookies and login sessions when using the HTTP protocol. This must be done on every node of a cluster installation to properly take effect.

To apply the changes, run:

automation**-**controller**-**service restart

**27.5. WebSockets port for live events not working**

automation controller uses port 80/443 on the controller server to stream live updates of playbook activity and other events to the client browser. These ports are configured for 80/443 by default, but if they are blocked by firewalls, close any firewall rules that opened up or added for the previous websocket ports, this will ensure your firewall allows traffic through this port.

**27.6. Problems running a playbook**

If you are unable to run the helloworld.yml example playbook from the Quick Start Guide or other playbooks due to playbook errors, try the following:

* Are you authenticating with the user currently running the commands? If not, check how the username has been setup or pass the --user=username or -u username commands to specify a user.
* Is your YAML file correctly indented? You may need to line up your whitespace correctly. Indentation level is significant in YAML. You can use yamlint to check your playbook. For more information, refer to the YAML primer at: <http://docs.ansible.com/YAMLSyntax.html>
* Items beginning with a - are considered list items or plays. Items with the format of key: value operate as hashes or dictionaries. Ensure you don’t have extra or missing - plays.

**27.7. Problems when running a job**

If you are having trouble running a job from a playbook, you should review the playbook YAML file. When importing a playbook, either manually or via a source control mechanism, keep in mind that the host definition is controlled by the controller and should be set to hosts: all.

**27.8. Playbooks aren’t showing up in the “Job Template” drop-down**

If your playbooks are not showing up in the Job Template drop-down list, here are a few things you can check:

* Make sure that the playbook is valid YML and can be parsed by Ansible.
* Make sure the permissions and ownership of the project path (/var/lib/awx/projects) is set up so that the “awx” system user can view the files. You can run this command to change the ownership:

chown awx **-**R **/**var**/**lib**/**awx**/**projects**/**

**27.9. Playbook stays in pending**

If you are attempting to run a playbook Job and it stays in the “Pending” state indefinitely, try the following:

* Ensure all supervisor services are running via supervisorctl status.
* Check to ensure that the /var/ partition has more than 1 GB of space available. Jobs will not complete with insufficient space on the /var/ partition.
* Run automation-controller-service restart on the controller server.

If you continue to have problems, run sosreport as root on the controller server, then file a [support request](http://support.ansible.com/) with the result.

**27.10. Cancel a controller job**

When issuing a cancel request on a currently running controller job, the controller issues a SIGINT to the ansible-playbook process. While this causes Ansible to stop dispatching new tasks and exit, in many cases, module tasks that were already dispatched to remote hosts will run to completion. This behavior is similar to pressing Ctrl-C during a command-line Ansible run.

With respect to software dependencies, if a running job is canceled, the job is essentially removed but the dependencies will remain.

**27.11. Reusing an external database causes installations to fail**

Instances have been reported where reusing the external DB during subsequent installation of nodes causes installation failures.

For example, say that you performed a clustered installation. Next, say that you needed to do this again and performed a second clustered installation reusing the same external database, only this subsequent installation failed.

When setting up an external database which has been used in a prior installation, the database used for the clustered node must be manually cleared before any additional installations can succeed.

Automation controller uses container technology to isolate jobs from each other. By default, only the current project is exposed to the container running a job template.

You may find that you need to customize your playbook runs to expose additional directories. To fine tune your usage of job isolation, there are certain variables that can be set.

By default, automation controller will use the system’s tmp directory (/tmp by default) as its staging area. This can be changed in the **Job Execution Path** field of the Jobs settings screen, or in the REST API at /api/v2/settings/jobs:

AWX\_ISOLATION\_BASE\_PATH = "/opt/tmp"

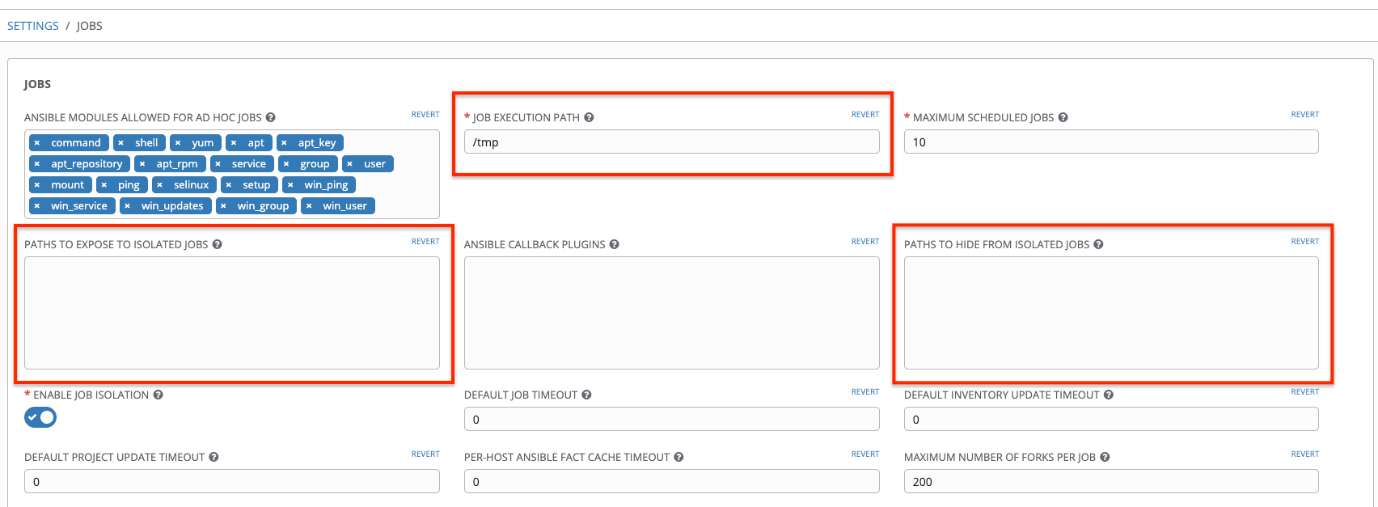
If there are any additional directories that should specifically be exposed from the host to the container that playbooks run in, you can specify those in the **Paths to Expose to Isolated Jobs** field of the Jobs setting scren, or in the REST API at /api/v2/settings/jobs:

AWX\_ISOLATION\_SHOW\_PATHS = ['/list/of/', '/paths']

**Note**

The primary file you may want to add to AWX\_ISOLATION\_SHOW\_PATHS is /var/lib/awx/.ssh, if your playbooks need to use keys or settings defined there.

The above fields can be found in the Jobs Settings window:



**27.12. Private EC2 VPC Instances in the controller Inventory**

By default, the controller only shows instances in a VPC that have an Elastic IP (EIP) associated with them. To see all of your VPC instances, perform the following steps:

1. In the controller interface, select your inventory.
2. Click on the group that has the Source set to AWS, and click on the Source tab.
3. In the Source Variables box, enter:

vpc\_destination\_variable: private\_ip\_address

Next, save and then trigger an update of the group. Once this is done, you should be able to see all of your VPC instances.

**Note**

The controller must be running inside the VPC with access to those instances if you want to configure them.

**27.13. Troubleshooting “Error: provided hosts list is empty”**

If you receive the message “Skipping: No Hosts Matched” when you are trying to run a playbook through the controller, here are a few things to check:

* Make sure that your hosts declaration line in your playbook matches the name of your group/host in inventory exactly (these are case sensitive).
* If it does match and you are using Ansible Core 2.0 or later, check your group names for spaces and modify them to use underscores or no spaces to ensure that the groups can be recognized.
* Make sure that if you have specified a Limit in the Job Template that it is a valid limit value and still matches something in your inventory. The Limit field takes a pattern argument, described here: <http://docs.ansible.com/intro_patterns.html>

Please file a support ticket if you still run into issues after checking these options.