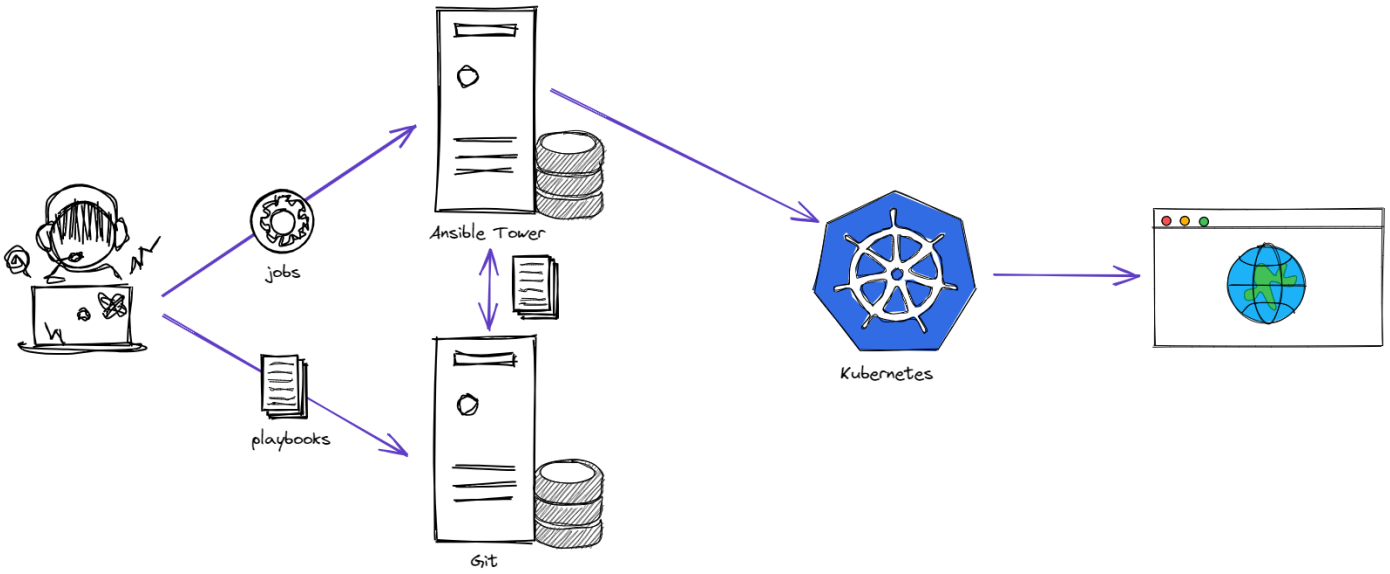


Simao’s guide to Ansible Tower *Kubernetes module*

Unofficial Notes

Simao Ferreira

Version 1.0, 2021-07-05

Table of Contents

This definitely has nothing to do with the Government of Canada at all. These are somewhat personal and opinionated notes from Simao Ferreira.

[DON'T PANIC]

Working with Ansible can be hard.

Working with Kubernetes can also be hard.

Working with Ansible Tower on Kubernetes can seem near impossible!

The goal of this guide is to give clear detailed steps in order to easily add packages to Ansible Tower. I will cover all the steps to do this whether you are using Tower in a container or Tower in a virtual machine. Usually you can probably find procedures for these sorts of things on DigitalOcean or Stack Overflow. But not this. Not that I could find anyhow. Vendor documentation can be sometimes cryptic but nonetheless here is the link that these procedures are based on.

https://docs.ansible.com/ansible-tower/latest/html/upgrade-migration-guide/virtualenv.html

Also, some space cadet decided to unplug my OpenShift lab so I had to do this on a single node k3 Rancher "cluster". This is why there are no OpenShift screen shots. **Sorry**.

With all that in mind - Let’s begin!

1. Procedure to add python modules inside Tower on `OpenShift`

The general idea is to create a persistent volume that will be mounted on the web containers and Task of the Tower pod in which we configure a virtual environment.

Procedure further **at your own risk**. Well, this part does assume you can at least spell *O+p+e+n+S+h+i+f+t* at least.

If you are here by mistake, quicky hit *Ctrl+Alt+w* or select **File › Quit** and bail out of here!

1.1. Create a new Python virtual environment (Kubernetes)

1. If you are where you are supposed to be then navigate to your `tower` project.

```
> oc project tower
```

2. Create a persistent volume that is mode ReadWriteMany RWX.

```
> oc create -f postgres-pvc.yml
```

Listing 1. postgres-pvc.yml

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my-environment1
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 2Gi
```

You may need to specify the storage class in order to get an RWX persistent volume claim.

3. Modify the `ansible-tower` deployment.

   Mount the persistent volume in the 2 containers “`ansible-tower-web`” and “`ansible-tower-task`”. Here are the different locations where this configuration must be added.

Listing 2. Redacted deployment example

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: ansible-tower

spec:
[...]
  template:
[...]
    spec:
[...]
      containers:
        - name: ansible-tower-web
[...]
            volumeMounts:
              - name: my-environment1
                mountPath: /var/lib/awx/venv/my-environment1
[...]
        - name: ansible-tower-task
[...]
            volumeMounts:
              - name: my-environment1
                mountPath: /var/lib/awx/venv/my-environment1
[...]
      volumes:
        - name: my-environment1
          persistentVolumeClaim:
            claimName: my-environment1
[...]
```

Save the YAML. This *should* force the Tower Pod to restart.

4. Verify that the mount is now present from a shell session inside the `ansible-tower-web` and `ansible-tower-task` containers.

```
> ls /var/lib/awx/venv
```

1.2. Create a new Python virtual environment

These steps moving forward will work on both container and traditional deployments. Hopefully you didn't try section 1 on a virtual machine or something. It's also a good time for a right now.

1. From a shell, execute the following to initialize a new python environment. You can actually do this for any path so long as you configure it in the custom virtual environment paths setting in Tower. But for this document I will use the default path.
    a. For python 2

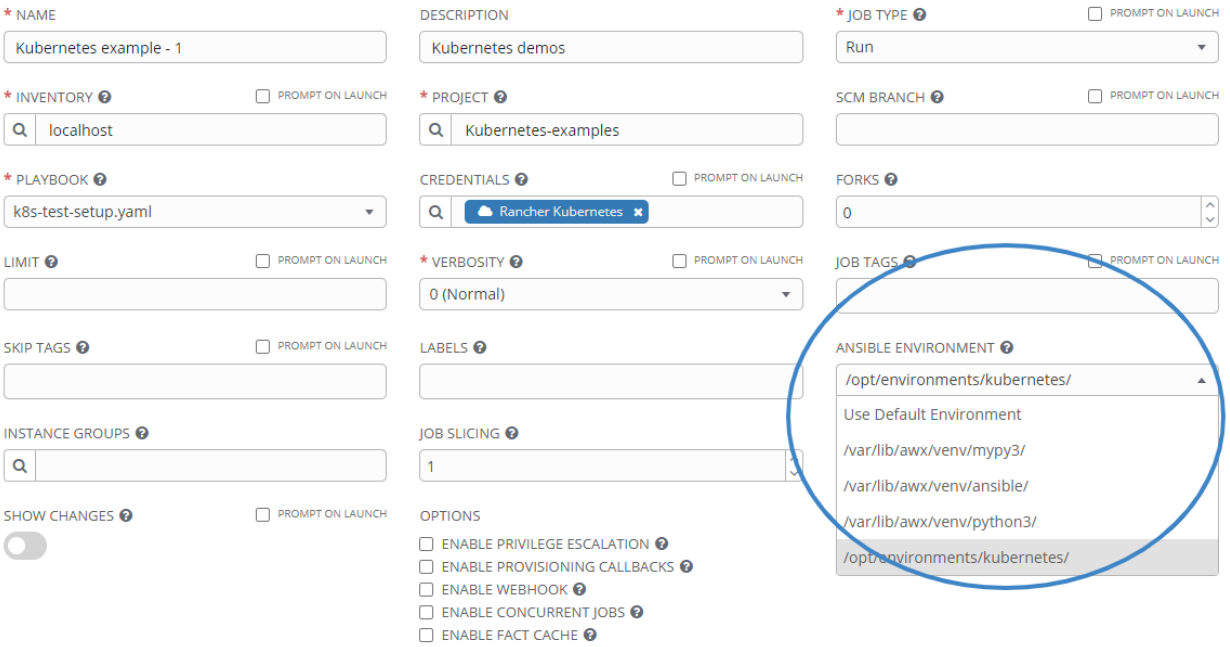
```
> virtualenv /var/lib/awx/venv/my-environment1/
```

b. For python 3 (*you should use this one*)

```
> python3 -m venv /var/lib/awx/venv/my-environment1/
```

Validate in Ansible Tower that the new environment is present. You can apply this at any hierarchical level of your choice, whether it is Organization, Project and even by Job Template.

You most likely need to log out of Ansible Tower and reconnect in order to see the changes. Also, my screenshot doesn't reflect the same environment paths as this document… deal with it.



1.3. Add Python modules to virtual environment

If your environment is offline then pip will probably not work. The offline method is quite different and I don't want to bother writing it out here.

> '*sigh…*'

Try searching G here

> '*oh look, it's a stack overflow page*'

1. First install the `Ansible` module (obviously). You can choose the version that meets your needs. Here is the list you can choose from: https://pypi.org/project/ansible/#history (Perhaps stick with 2.9.x if you want to be supported by **Red Hat**)

Don’t use the system pip… use the one located inside the bin folder from your environment folder.

```
> ./bin/pip3 install "ansible == 2.9.23"
```

2. Then install the `psutil` module needed by Ansible Tower.

```
> ./bin/pip3 install psutil
```

3. You can now install the modules of your choice.

Such as OpenShift… which also installs Kubernetes. ✔

```
> ./bin/pip3 install openshift
```

You can create as many virtual environments as you need. If this not enough than repeat this procedure using different mount names.

> Are we done?
>
> What is there to do?
>
> Oh, right. How does this work on Tower running in a container?

1.4. Create a management and installation pod to build modules

We must create a place where it will be possible to add python modules. As a reminder, some modules require compilation with the python-dev and gcc tools. Since these are not present in the base images of Ansible Tower, we will need to create a container with these tools.

I did not create this repository that contains the necessary tools in the directory module-builder from the following Git repo: https://github.com/sferr/openshift-tower-tuto1.git

This repository contains an example of a Kubernetes manifest to instantiate a pod with an image and mounts the virtual environment directory. I deployed this container image on Quay.io, here: https://quay.io/repository/sferr/ansible-module-builder?tab=info **So you don’t have to**.

1. Clone the repository

```
> git clone https://github.com/sferr/openshift-tower-tuto1.git
```

2. Instantiate the pods with this command

```
> cd openshift-tower-tuto1/
> oc apply -f module-builder/pod.yaml
```

You can also copy the contents of the Pod.yaml file available in the “module-builder” directory into the OpenShift console’s Pods button.

…

The Pods button is under Workloads…

Once the new Pod is operational, all you have to do is open a terminal and go to the directory containing the virtual environment. The new pod will contain a directory `/var/lib/awx/venv/my-environment1` which will be mounted on the same persistent volume as Ansible Tower. Modules added to this environment will therefore be automatically available in Ansible Tower.

> **Step 1.3** should now work the same… but you probably need to use `./bin/pip` instead of `./bin/pip3`.

2. Creating Kubernetes credentials in Tower

Tower does not currently have a native Kubernetes credential type, but it does provide the ability to create custom Credential Types. Kubernetes credentials are typically stored in a Kubeconfig file which is stored locally on the user workstation in the file ~/.kube/config by default. First we need to create a Kubernetes Credential Type, and then create a Credential using the new credential type which stores the contents of the Kubeconfig file.

> This part of the prodecure is based on this URL: https://docs.ansible.com/ansible-tower/latest/html/userguide/credential_types.html

2.1. Credential Type

1. Navigate to **Administration > Credential Types** and add a new credential type.
2. Enter the following YAML in the Input Configuration:

   Listing 3. input configuration

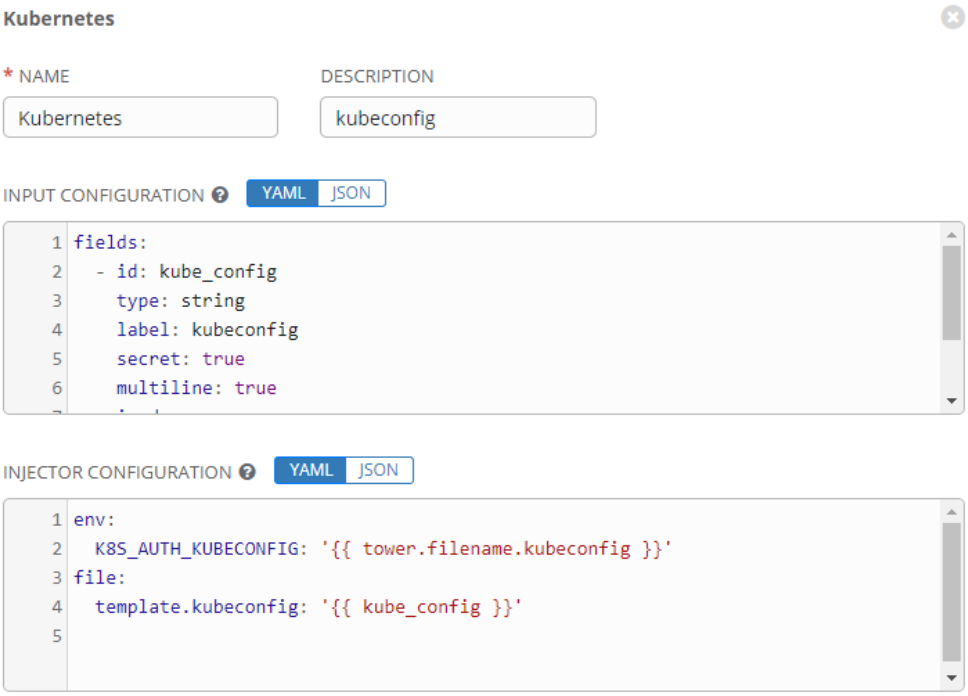
   ```
   ---
   fields:
     - id: kube_config
       type: string
       label: kubeconfig
       secret: true
       multiline: true
   required:
     - kube_config
   ```

3. Enter the following YAML in the Injector Configuration:

   Listing 4. injector configuration

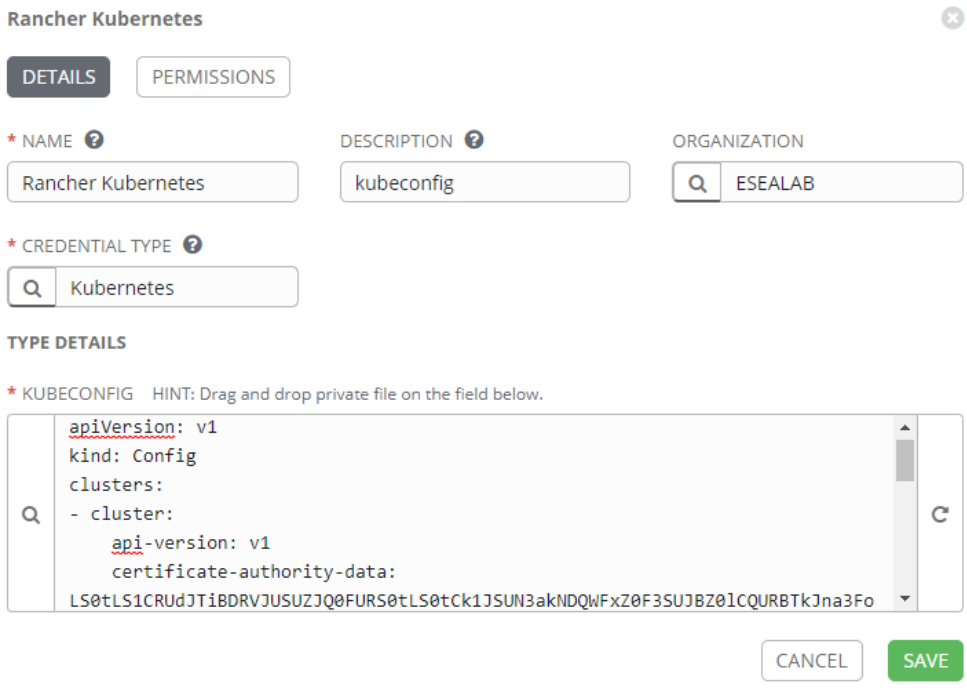
   ```
   ---
   env:
     K8S_AUTH_KUBECONFIG: "{{ tower.filename.kubeconfig }}"
   file:
     template.kubeconfig: "{{ kube_config }}"
   ```

   When complete the configuration should look similar to the image below:



2.2. Credential

1. Navigate to Credentials and add a new Credential
2. In the Credential Type, select the Kubernetes custom Credential Type that was just created. In the Kubeconfig field, copy/paste the contents of your Kubeconfig file. Your configuration should look similar to the following:



3. Ansible Playbook Examples

> These examples (and this documentation) can be found here:
> https://github.com/sferr/openshift-tower-tuto2

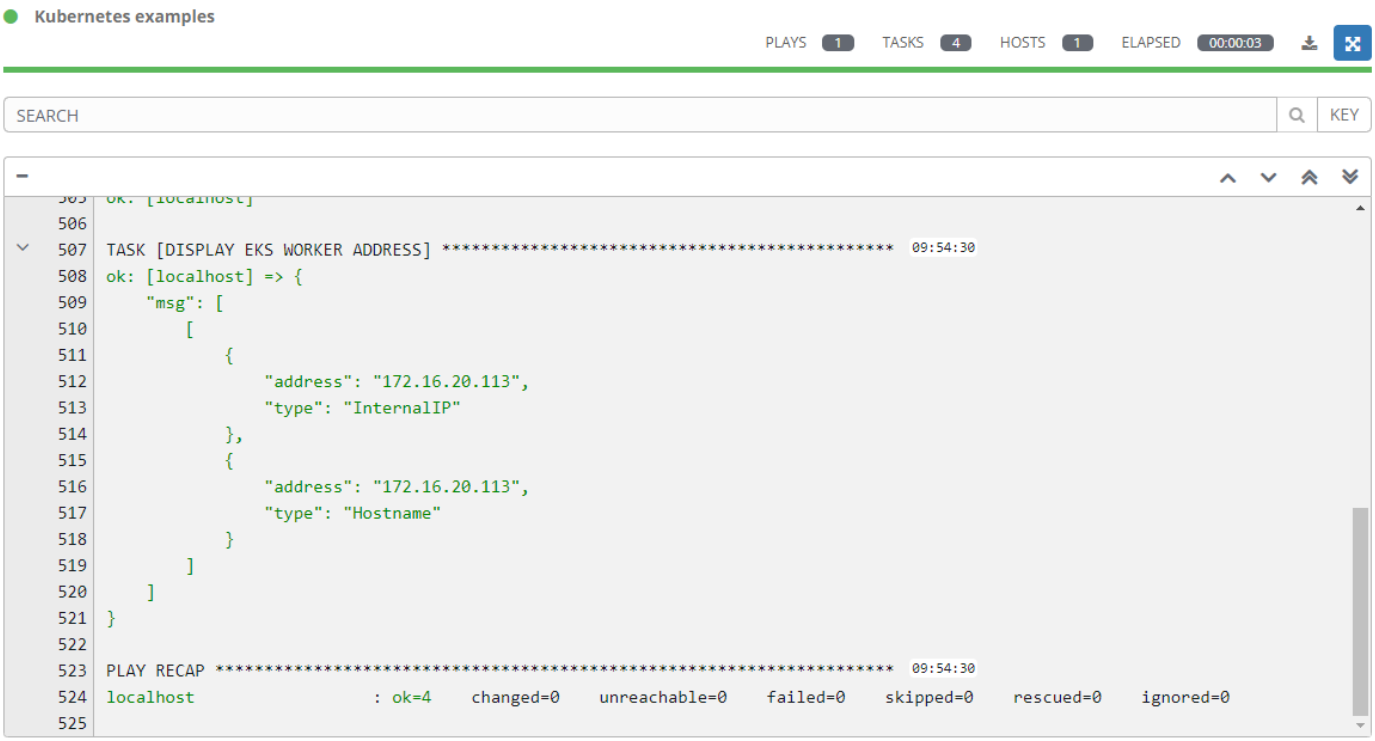
3.1. Playbook example 1

In this first example we will determine if the credentials work and if so the next few tasks will gather data from the nodes and output their hostname and IP. The playbook I used is below:

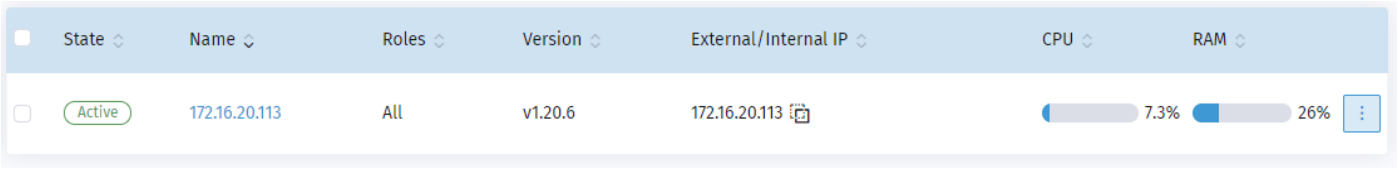
Listing 5. k8s-test-setup.yaml

```
---
- name: RETRIEVE WORKER NODE DETAILS FROM KUBERNETES NODE
  hosts: localhost
  gather_facts: no

  collections:
    - kubernetes

  tasks:
    - name: GET WORKER NODE DETAILS
      k8s_info:
        kind: Node
        namespace: default
      register: node_result

    - name: DISPLAY OUTPUT
      debug:
        msg: "{{ node_result }}"

    - name: SET VARIABLE STORING WORKER NODE ADDRESS
      set_fact:
        worker_address: "{{ node_result | json_query(query) }}"
      vars:
        query: "resources[].status.addresses"

    - name: DISPLAY WORKER ADDRESS
      debug:
        msg: "{{ worker_address }}"
```

And here is the output I got within Tower.



Pretty cool! Let's check Rancher and see if it's accurate. Looks good! And yes, the hostname is also the IP address in my case.



3.2. Playbook example 2

In this second example we're just going to deploy a simple application from YAML. Behold, the playbook:

Listing 6. deploy-busybox.yaml

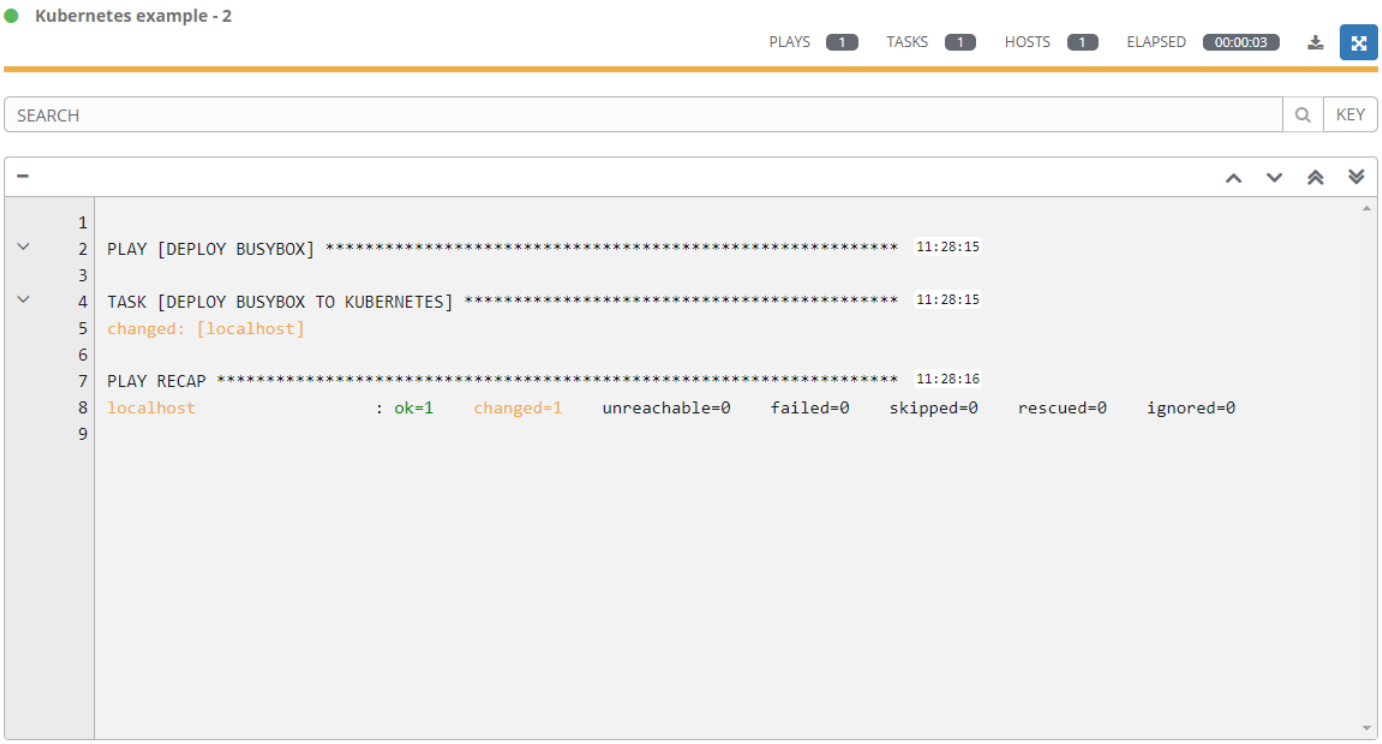
```
---
- name: DEPLOY BUSYBOX
  hosts: localhost
  gather_facts: no

  tasks:

  - name: DEPLOY BUSYBOX TO KUBERNETES
    k8s:
        definition: "{{ lookup('template', 'busybox.yaml') | from_yaml }}"
        state: present
```

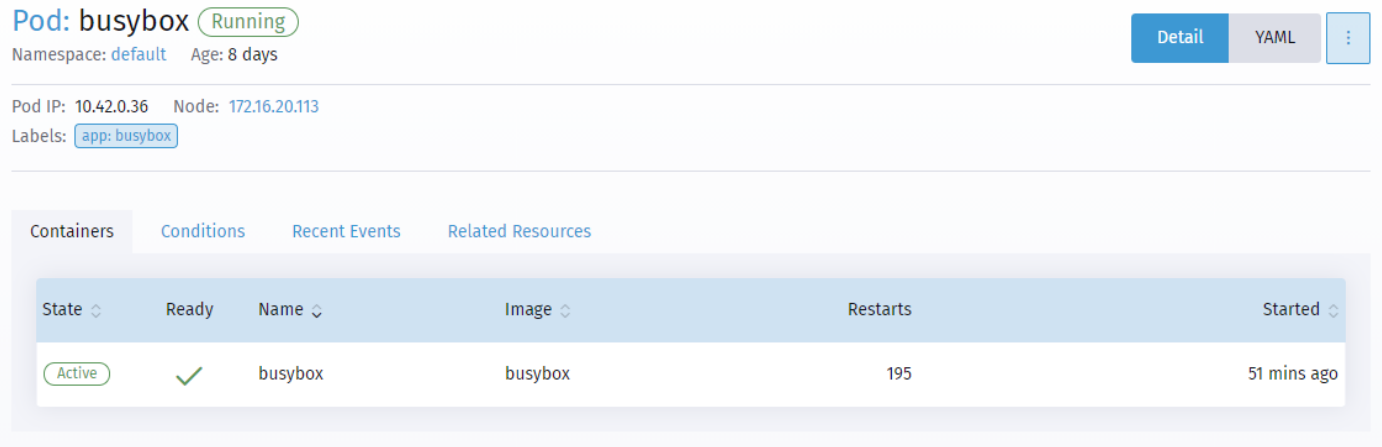
Okay that doesn't look like much… behold the Kubernetes YAML:

Listing 7. busybox.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: busybox
  labels:
    app: busybox
  namespace: default
spec:
  containers:
  - name: busybox
    image: busybox
    command: ['sleep','3600']
    imagePullPolicy: IfNotPresent
  restartPolicy: Always
```

Okay, okay it's still not that impressive. But it does the job and below you will see that the containerized application is **RUNNING**.





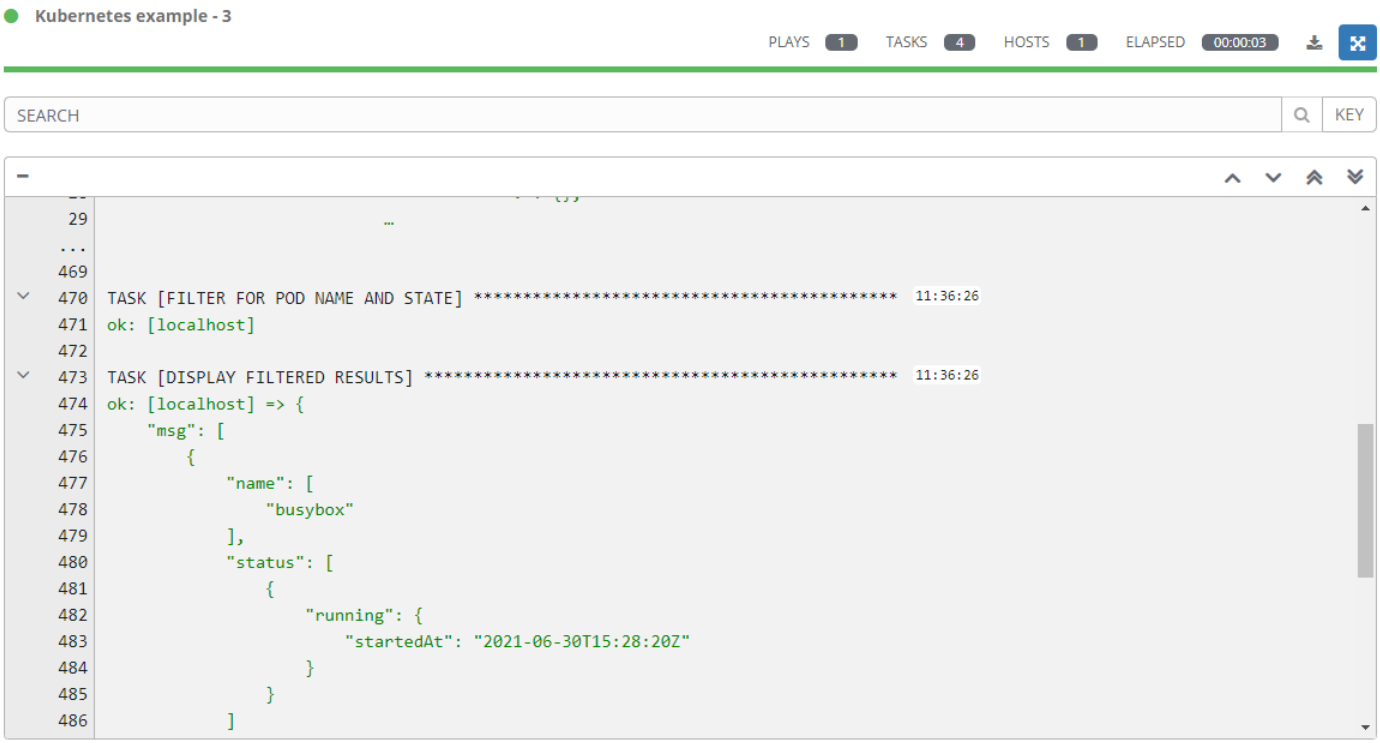
3.3. Playbook example 3

In the final example we're just going to collect all the details from the pod and show the output. The playbook:

Listing 8. display_pod_info.yaml

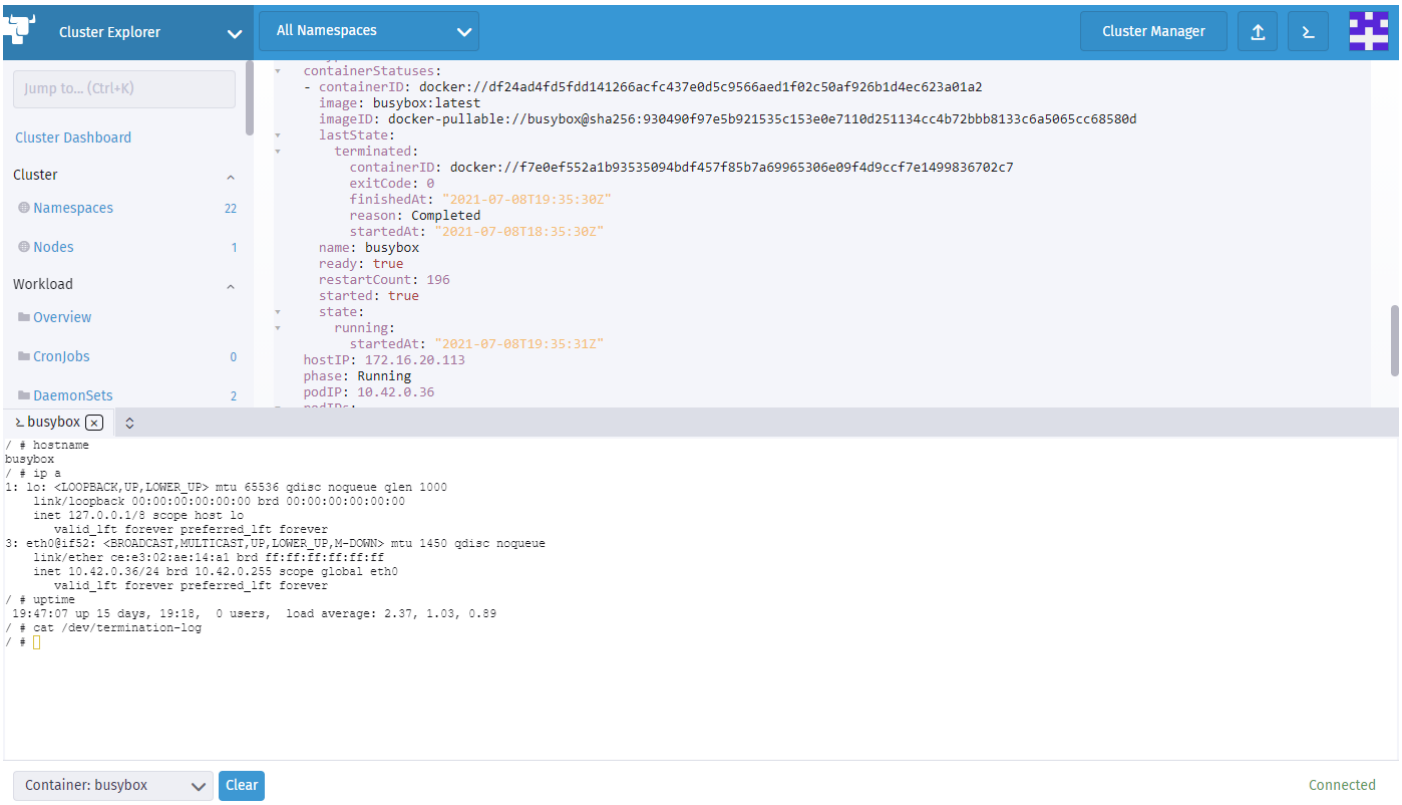
```
---
- name: RETRIEVE POD DETAILS FROM KUBERNETES
  hosts: localhost
  gather_facts: no

  collections:
    - kubernetes

  tasks:
  - name: GET POD DETAILS
    k8s_info:
      kind: Pod
      namespace: default
    register: pod_result

  - name: DISPLAY OUTPUT
    debug:
      msg: "{{ pod_result }}"

  - name: FILTER FOR POD NAME AND STATE
    set_fact:
      filtered_result: "{{ pod_result | json_query(query) }}"
    vars:
      query: "resources[].{name: status.containerStatuses[].name, status:
status.containerStatuses[].state}"

  - name: DISPLAY FILTERED RESULTS
    debug:
      msg: "{{ filtered_result }}"
```

The output in Tower:



And then the confirmation that the data matches!

> Okay it doesn't match at all and I could have showed more data in the Tower output. But A. I didn't want to add extra pages of just data output to this doc and B. I ran the two on different days and the pod restarted a bunch of times in the time between.



> I hope this guide provides some useful information on how to get up and running quickly automating Kubernetes with Ansible Tower!

Brought to you with ❤ *by me.*

Name	Title	Alias
Simao Ferreira	**Techy of the Galaxy**	@simao_f

Powered by Open Source