Introducing Ansible AWX and Red Hat Ansible Automation Platform

This chapter will examine two graphical interfaces for Ansible: the commercial **Red Hat Ansible Automation Platform** and the open source **Ansible AWX** – or to give it its full name, **Ansible Web eXecutable**.

This chapter will focus on the open source Ansible AWX because it is freely available and, outside of the resources required to run the tool, requires no upfront costs or contracts.

We will discuss how to install Ansible AWX and why you would want to use it. After all, we are 16 chapters into our journey with Ansible and haven’t needed to use a graphical interface yet – so why now?

By the end of this chapter, we will have done the following:

* Discussed Red Hat Ansible Automation Platform versus Ansible AWX
* Installed and configured Ansible AWX
* Deployed our Microsoft Azure cloud application using Ansible AWX

Technical requirements

While we will only deploy Ansible AWX in this chapter, its requirements are complex. Because of this, rather than running it locally, I will provide instructions for deploying a Kubernetes cluster in Microsoft Azure using the AKS service.

If you are following along, you will need access to a Microsoft Azure account and have the Azure CLI installed. For more information, see [*Chapter 9*](https://subscription.packtpub.com/book/cloud-and-networking/9781835088913/9), *Moving to the Cloud*.

Red Hat Ansible Automation Platform versus AWX

Red Hat Ansible Automation Platform and Ansible AWX are two powerful tools Red Hat provides for managing and streamlining your Ansible deployments. Both tools offer web-based interfaces that simplify the execution and management of Ansible playbooks, making it easier for users to leverage Ansible’s automation capabilities without requiring extensive command-line knowledge.

Red Hat Ansible Automation Platform, formerly known as **Ansible Tower**, is a comprehensive enterprise-grade solution that goes beyond the capabilities of Ansible Tower. It integrates various components to create a cohesive and expansive automation environment. Some key features of Red Hat Ansible Automation Platform are as follows:

* **Centralized control**: Red Hat Ansible Automation Platform provides a unified web-based dashboard for defining, scheduling, and monitoring automation jobs from a central location.
* **Role-based access control** (**RBAC**): With granular access, you can ensure that your users have appropriate access to automation resources, enhancing security and control.
* **Workflow management**: Create complex workflows that combine multiple playbooks, job templates, and inventory sources, as well as supporting dependencies, conditionals, and approvals.
* **Scalability and flexibility**: Automation can be scaled to meet the needs of large enterprises, supporting diverse infrastructures, including cloud platforms, containers, and network devices.
* **Content collections**: Access pre-packaged modules and plugins that have been expertly curated to expedite the implementation of automation projects.
* **Automation Hub**: This centralized repository hosts certified, partner-supported, and community-driven content. It fosters collaboration and accessibility to high-quality resources.
* **Automation analytics**: Utilize sophisticated analytics tools to scrutinize performance, utilization, and various KPIs across different clusters and instances.
* **Integration with Red Hat ecosystem**: Seamless integration with other Red Hat products such as Red Hat Insights and Red Hat Satellite, fostering a cohesive environment.

On the other hand, Ansible AWX is the open source upstream project for Red Hat Ansible Automation Platform. It provides many of the platform’s core features but follows a community-driven development model with more frequent releases. While Ansible AWX offers a solid foundation for automation, Red Hat Ansible Automation Platform may need some enterprise-specific features and integrations.

The choice between Red Hat Ansible Automation Platform and Ansible AWX depends on your organization’s needs and requirements. Red Hat Ansible Automation Platform is ideal for enterprises seeking a robust, feature-rich solution with commercial support and seamless integration with the Red Hat ecosystem. It offers advanced features and is designed to handle complex automation needs across diverse environments.

On the other hand, Ansible AWX is a suitable choice for organizations that prefer an open source solution and are comfortable with community-driven support. It provides a solid foundation for automation and benefits from more frequent updates and community contributions.

Both Red Hat Ansible Automation Platform and Ansible AWX allow organizations to automate at scale, reduce manual efforts, and improve the consistency and reliability of their IT operations. They provide user-friendly interfaces and enable effective team collaboration, increasing efficiency and improved compliance.

Ansible AWX

To say that installing Ansible AWX is complicated is an understatement. Since Red Hat first open sourced the project, deploying it has always been difficult.

Luckily, the first release was containerized, and it has slowly transitioned from running in a small number of containers to being able to run in a Kubernetes cluster and managed by the AWX Operator.

Information

A Kubernetes Operator uses custom resources to automate application and component management in Kubernetes clusters. It extends the cluster’s behavior without modifying the Kubernetes code itself. Operators can handle various tasks, such as deployment, backups, upgrades, and service discovery, reducing manual intervention and increasing the system’s reliability.

Let’s start by launching our own Kubernetes in Microsoft Azure and configuring our local machine so that we can deploy and configure the AWX Operator.

Deploying and configuring the Ansible AWX Operator

The first thing we need to do is deploy the Kubernetes cluster. To do this, we will use the Azure CLI to launch an AKS cluster. To start with, we need to set some variables on the command line to define the resource names, which Azure region we would like the cluster to deploy into, and how many compute nodes we require:

$ AKSLOCATION=uksouth

$ AKSRG=rg-awx-cluster

$ AKSCLUSTER=aks-awx-cluster

$ AKSNUMNODES=2

CopyExplain

Next up, let’s create the Azure Resource Group we will be deploying our cluster into; this will make it easy to remove once we have finished as we need to delete the group and its contents:

$ az group create --name $AKSRG --location $AKSLOCATION

CopyExplain

With the resource group in place, we can now launch the AKS cluster:

$ az aks create \

     --resource-group $AKSRG \

     --name $AKSCLUSTER \

     --node-count $AKSNUMNODES \

     --generate-ssh-keys

CopyExplain

This will take around 5 minutes to deploy. If you don’t have the kubectl command installed on your local machine, then you can run the following command to have the Azure CLI install it for you:

$ az aks install-cli

CopyExplain

Finally, with kubectl installed, you can configure the credentials and contexts by running the following command:

$ az aks get-credentials --resource-group $AKSRG --name $AKSCLUSTER

CopyExplain

With our cluster now launched and available, we must install and configure the AWX Operator using Helm.

Information

**Helm** is a package manager that simplifies Kubernetes deployment by packaging applications as charts and defining necessary resources and configurations. For more details and installation instructions, see https://helm.sh/.

First, we need to enable the AWX repository and pull it down to our local machine:

$ helm repo add awx-operator https://ansible.github.io/awx-operator/

$ helm repo update

CopyExplain

Now, we need to deploy the AWX Operator to our cluster:

$ helm install -n awx --create-namespace awx awx-operator/awx-operator --version 2.12.1

CopyExplain

It will take a minute or two to deploy.

Please note

You might have noticed that the preceding command specifies an explicit version number because there are some known bugs with the current release, which is a major update from the version we are using.

You can run the following command to check the status of the deployment:

$ kubectl get pods -n awx

CopyExplain

Once everything is ready, you should see something like the following screen:

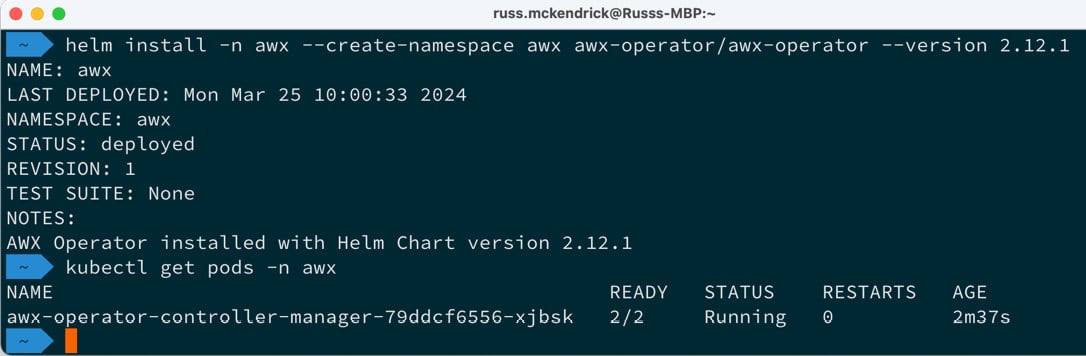


Figure 16.1 – Deploying the AWX Operator

With the AWX Operator deployed with our cluster, we can request that the operator now deploy AWX itself. To do this, run the following command:

$ kubectl apply -f https://raw.githubusercontent.com/PacktPublishing/Learn-Ansible-Second-Edition/main/Chapter16/awx/ansible-awx.yaml

CopyExplain

This command simply passes the following YAML configuration to the operator to instruct it how to deploy our AWX installation:

---

apiVersion: awx.ansible.com/v1beta1

kind: AWX

metadata:

  name: ansible-awx

  namespace: awx

spec:

  service\_type: loadbalancer

CopyExplain

As you can see, there’s not much to it, so please don’t consider this a production-ready AWX instance. All we are instructing the AWX Operator to do is deploy AWX and expose the service via a load balancer so that we can connect to it.

Now, we wait; our AWX installation will take 15 to 20 minutes to deploy the application and bootstrap itself.

You can check the status of the containers and the load balancer service by running the following code:

$ kubectl get pods -n awx

$ kubectl get svc ansible-awx-service -n awx

CopyExplain

Once the basics have been deployed, you should see something like the following. These are the containers that service the AWX application. As you can see, there are ones for the database, task runner, and the web interface:

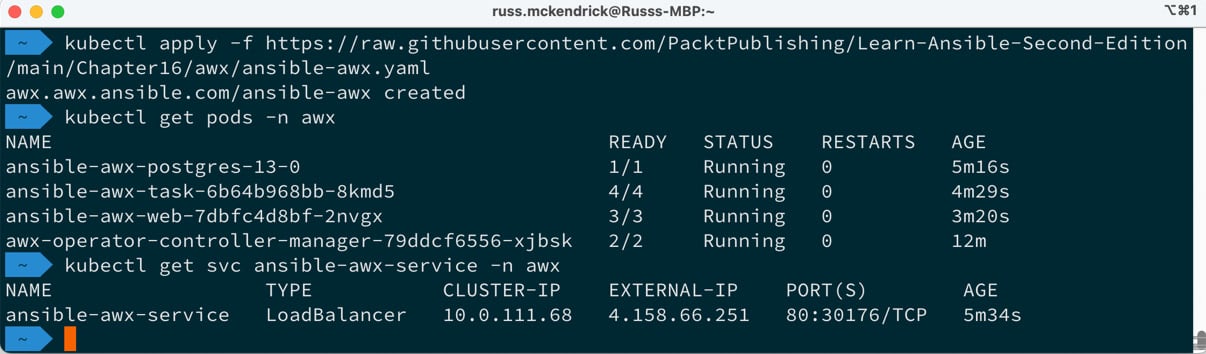


Figure 16.2 – Checking the status of our AWX deployment

Once your deployment looks like the preceding output, the final step is to grab the admin password. To do this, run the following command – the secret will always be named ansible-awx-admin-password:

$ kubectl get secret -n awx ansible-awx-admin-password -o jsonpath="{.data.password}" | base64 –decode

CopyExplain

This will grab the base64 encoded secret from the Kubernetes secret store and decode it for you – it should look like this:

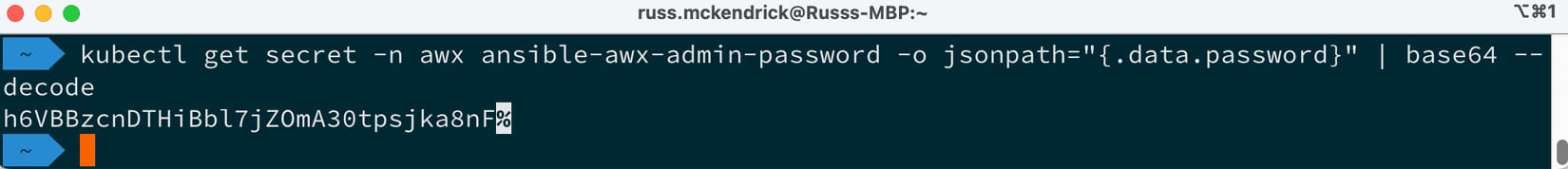


Figure 16.3 – Grabbing the admin password

As you may have noticed in the preceding output, there is a % icon at the end – this is not part of the password, and you need everything before that.

Please make a note of the password and the EXTERNAL-IP value from the previous commands as this tells you where to go to log in and what credentials to use. In the preceding deployment (which has long since been terminated), these details are as follows:

* **URL**: http://4.158.66.251/
* **Username**: admin
* **Password**: h6VBBzcnDTHiBbl7jZOmA30tpsjka8nF

When you go to the URL, you should be greeted with a login page that looks like this:

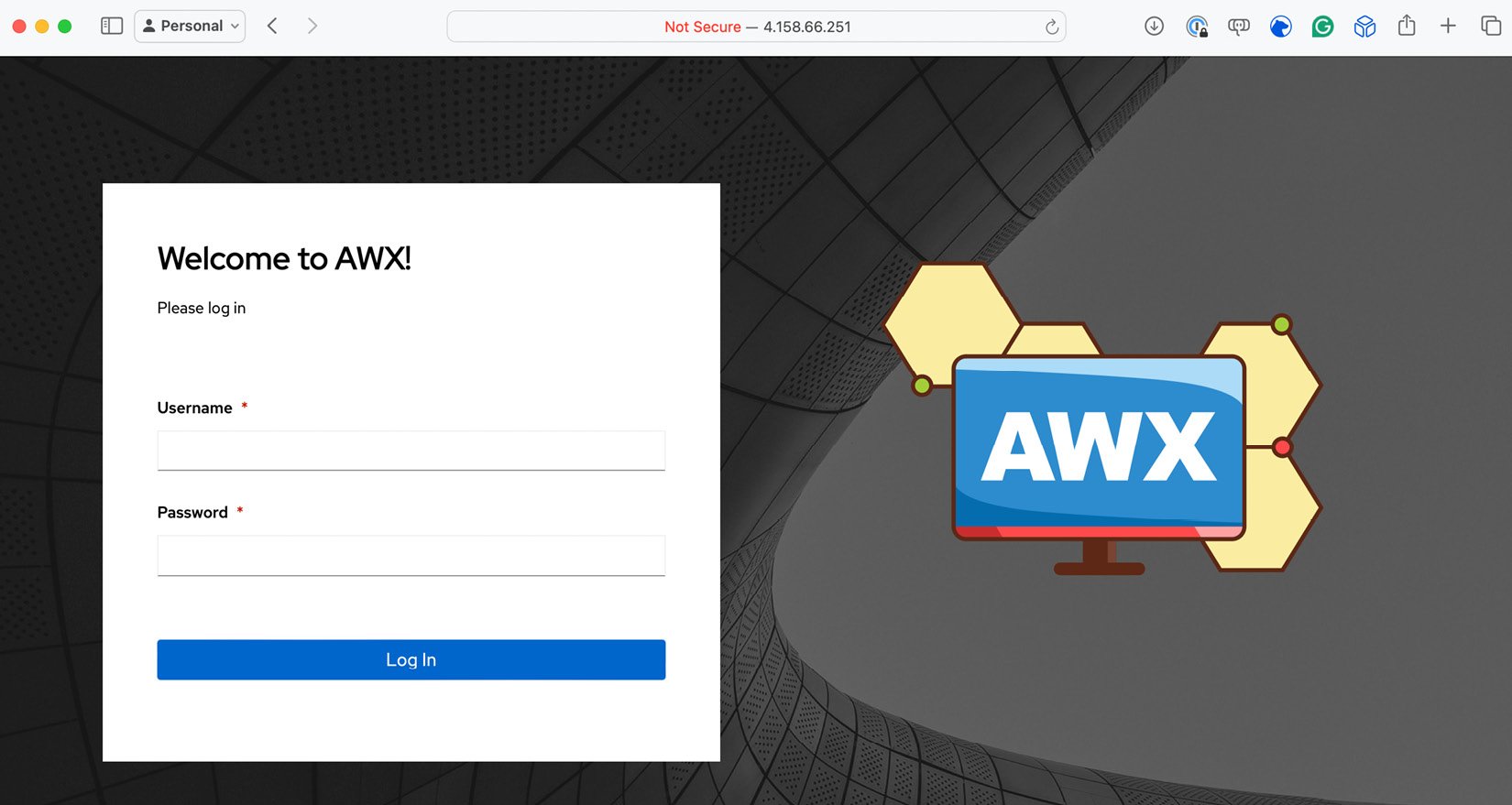


Figure 16.4 – Grabbing the admin password

Once you log in, you will be taken to your empty AWX instance:

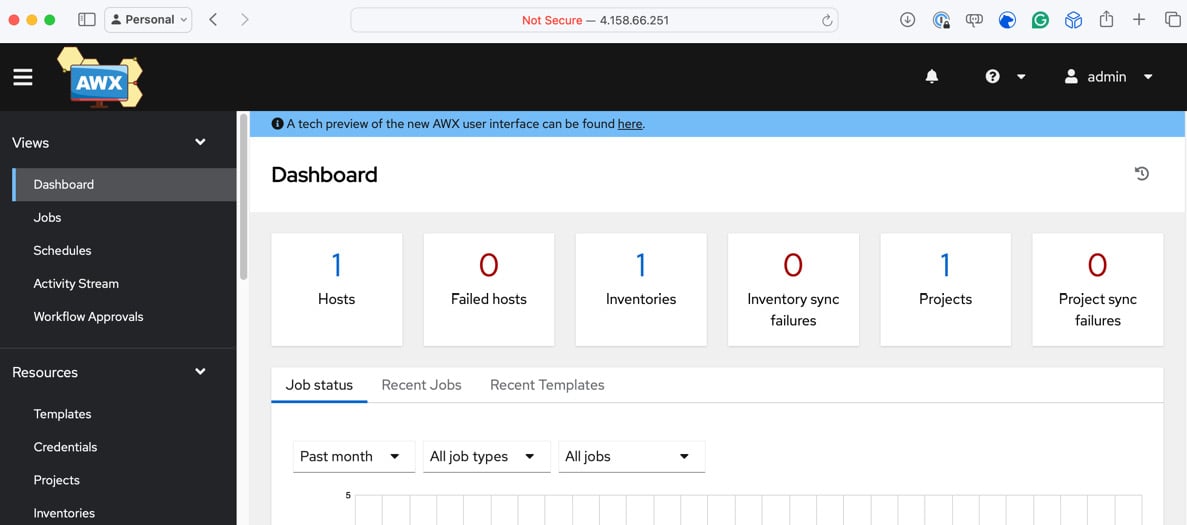


Figure 16.5 – Grabbing the admin password

Now, let’s set up our playbook.

Setting up our playbook

Before running our playbook, we must import it into Ansible AWX and configure the supporting credentials, such as our Azure Service Principle. We’ll start with a project.

Adding a new project

First, we need to add a new project, where we tell Ansible AWX about the repository hosting our playbook. As mentioned previously, we will use a GitHub repository that houses the code. To add a new project, click on **Projects** under **Resources** in the left menu and then click on the **Add** button.

Here, you will be asked for several bits of information; enter the following:

* **Name**: Azure WordPress
* **Description**: Deploy WordPress in Azure
* **Organization**: Default
* **Execution Environment**: Select **AWX EE (latest)**
* **Source Control Type**: GIT

When you select the **Source Control Type** value a second section will appear that asks for details about where your source is hosted:

* **Source Control URL**: https://github.com/PacktPublishing/Learn-Ansible-Second-Edition.git
* **Source Control Branch/Tag/Commit**: Leave blank
* **Source Control Refspec**: Leave blank
* **Source Control Credential**: Leave blank
* **Options**: Just select Clean

Once you have entered these details, click **Save**. Now, if you return to the **Projects** page, you should see that Ansible has already downloaded the source for the playbook:

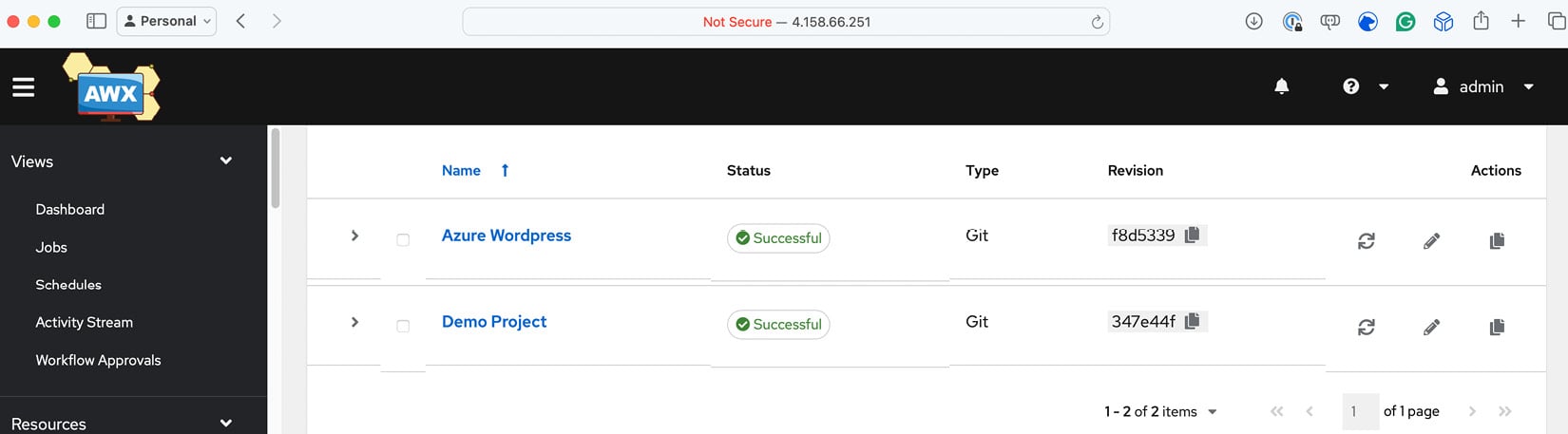


Figure 16.6 – Adding the project and downloading the code from GitHub

Adding credentials

Next, we must tell Ansible AWX the credentials to use when accessing our Azure environment; to add these, click **Credentials**. This can also be found under the **Resources** section of the left-hand menu. Click **Add** and enter the following:

* **Name**: Azure
* **Description**: Credentials for Azure
* **Organization**: Default
* **Credential Type**: Select **Microsoft Azure Resource Manager**

As before, this will open a separate section; here, you will need to enter details of the service principle we created in [*Chapter 15*](https://subscription.packtpub.com/book/cloud-and-networking/9781835088913/15), *Using Ansible with GitHub Actions and Azure DevOps*:

* **Subscription ID**: Enter the subscription ID; in the example from the previous chapter, this was e80d5ad9-e2c5-4ade-a866-bcfbae2b8aea
* **Username**: Leave blank
* **Password**: Leave blank
* **Client ID**: Enter the appId value that was returned when you created the service principle; in the previous chapter’s example, this was 2616e3df-826d-4d9b-9152-3de141465a69
* **Client Secret**: Enter the password value that was returned when you created the service principle; in the previous chapter’s example, this was Y4j8Q~gVO\*NoTaREalPa55w0rdpP-pdaw
* **Tenant ID**: Enter the tenant ID; in the example from the previous chapter, this was c5df827f-a940-4d7c-b313-426cb3c6b1fe

Once the form has been filled in, click **Save**. Once saved, you will notice that the **Client Secret** value is marked as **Encrypted**:

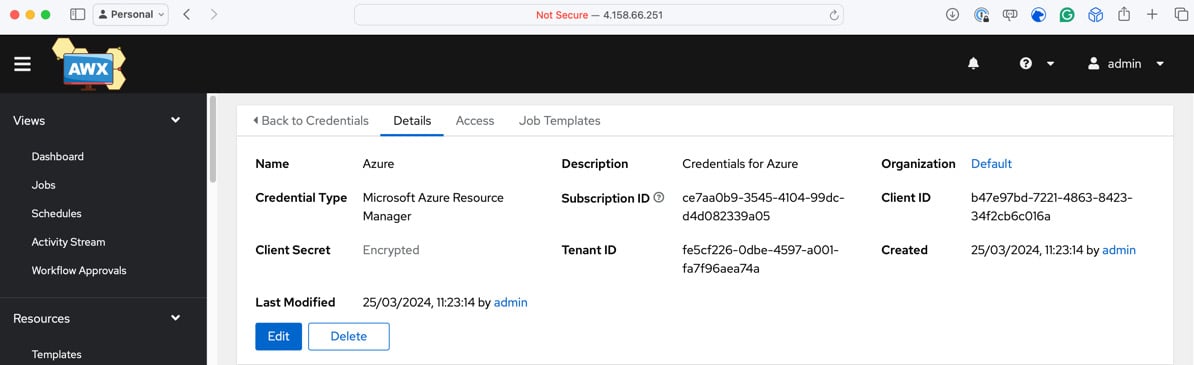


Figure 16.7 – Adding our Service Principle to Ansible AWX

When you save sensitive information in Ansible AWX, it is encrypted, and you only have the option to **Replace** or **Revert** it. At no point can you view this information again.

Next, we need to create a credential that contains the private portion of the SSH key we generated in [*Chapter 15*](https://subscription.packtpub.com/book/cloud-and-networking/9781835088913/15)*, Using Ansible with GitHub Actions and Azure DevOps*. To do this click on **Add** again, but this time, enter the following:

* **Name**: AzureVM
* **Description**: Private SSH Key for Azure VMs
* **Organization**: Default
* **Credential Type**: Select **Machine**

In the additional information boxes, enter the following information:

* **Username**: azureadmin
* **Password**: Leave blank
* **SSH Private Key**: Copy and paste the contents of the private key or upload the private key file
* **Remaining options**: Leave blank

Once filled in, click **Save**. Once back on the **Credentials** screen, click **Add** once more and enter the following:

* **Name**: Ansible Galaxy
* **Description**: Ansible Galaxy creds for Default org
* **Organization**: Default
* **Credential Type**: Select **Ansible Galaxy/Automation Hub API Token**

Then, enter this information:

* **Galaxy Server URL**: https://galaxy.ansible.com
* **Remaining options**: Leave blank

Again, click **Save**. Now, it’s time to add our final set of credentials:

* **Name**: WordPress Vault
* **Description**: Vault Password for WordPress secrets
* **Organization**: Default
* **Credential Type**: Select **Vault**

In the **Type Details** section, enter the following:

* **Vault Password**: I have added the passwords (which I will tell you later in this chapter) as pre-encrypted Ansible Vault variables in group\_vars/common.yml in the Chapter16 playbook. Because of that, you must enter a password of wibble here – if you don’t enter this, the example playbook will fail.
* **Vault Identifier**: Leave blank.

That was our final credentials. So, let’s move on to the next configuration step.

Adding an inventory

Now that we have all our credentials in place, we need to recreate the content of the production inventory file within Ansible AWX. As a reminder, the inventory file we have been using looks like this (minus the comments):

[local]

localhost ansible\_connection=local

[vmgroup]

[azure\_vms:children]

vmgroup

[azure\_vms:vars]

ansible\_ssh\_user=adminuser

ansible\_ssh\_private\_key\_file=~/.ssh/id\_rsa

host\_key\_checking=False

CopyExplain

To add the inventory, click on **Inventories**, which is again in the left-hand menu. The **Add** button now brings up a drop-down list; we want to select **Add inventory** from that list.

In the form that opens, enter the following:

* **Name**: Azure Inventory
* **Description**: Azure Inventory
* **Organization**: Default
* **Instance Groups**: We will add these in a moment
* **Labels**: Leave blank
* **Variables**: Enter the values listed here:
* ansible\_ssh\_user: "adminuser"
* ansible\_ssh\_private\_key\_file: "~/.ssh/id\_rsa"

host\_key\_checking: false

CopyExplain

Once entered, click **Save**; this will create the inventory. Now, we can add the two groups we need. To do this, click **Groups**, which can be found in the row on the buttons above the details of the inventory:

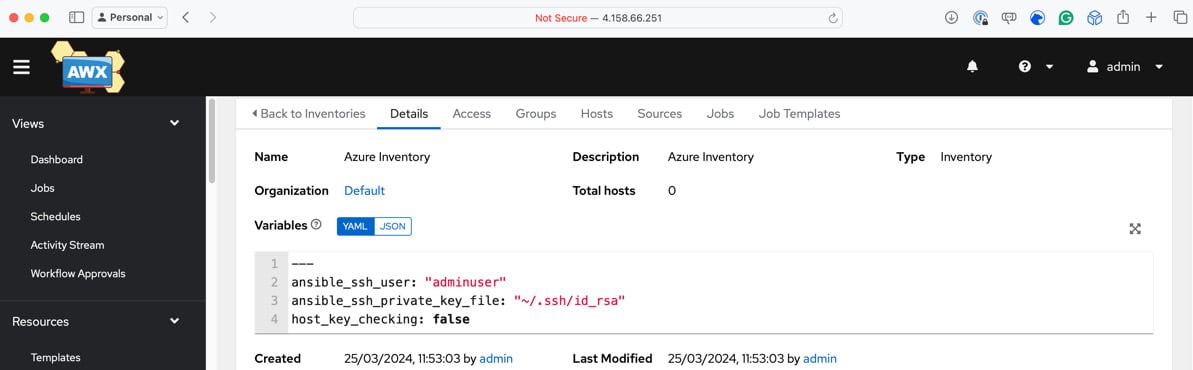


Figure 16.8 – Adding the inventory to Ansible AWX

Click **Add** and enter the following details:

* **Name**: vmgroup
* **Description**: vmgroup
* **Variables**: Leave blank

Then, click **Save**, repeat the process, and add a second group using the following details:

* **Name**: azure\_vms
* **Description**: azure\_vms
* **Variables**: Leave blank

Again, click **Save**; you should now have two groups listed.

Now that we have our project, inventory, and some credentials for accessing our Azure environment, we need to add the templates to launch and configure the cluster and terminate it.

Adding the templates

Let’s look at adding the templates.

Information

We will pass a runtime variable to our playbook, which will contain the public part of the SSH key – we added the private portion as a credential earlier in this chapter – and will be called ssh\_key\_public. Please ensure you have the public key when filling out these details.

Click **Templates** in the left-hand menu and, in the drop-down menu of the **Add** button, select **Job Template**. This is the most extensive form we have encountered; however, parts will be populated automatically when we fill in the details. Let’s make a start:

* **Name**: Launch WordPress
* **Description**: Launch WordPress in Azure
* **Job Type**: Select **Run**
* **Inventory**: Select **Azure Inventory**
* **Project**: Select **Azure WordPress**
* **Execution Environment**: Select **AWX EE (latest)**
* **Playbook**: Choose **Chapter16/site.yml** from the drop-down list
* **Credentials**: Select the following:
  + **Machine**: **AzureVM**
  + **Microsoft Azure Resource Manager**: **Azure**
  + **Vault**: **WordPress Vault**
* **Variables**: You should enter the ssh\_key\_public variable here; a truncated version of what to enter is shown here:
* ---

ssh\_key\_public: "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDCGosD5doqnJgOLpkztaDvIZFaCKoChm9yyU6FPaci9fZR60SCXbOu1zeMmyJouFH7xVBv7xw5HBk0FDNLXrssR5B7YHiti8= youremail@example.com"

CopyExplain

* **Remaining options**: Leave blank

Click **Save**; you will be taken to the overview of the template:

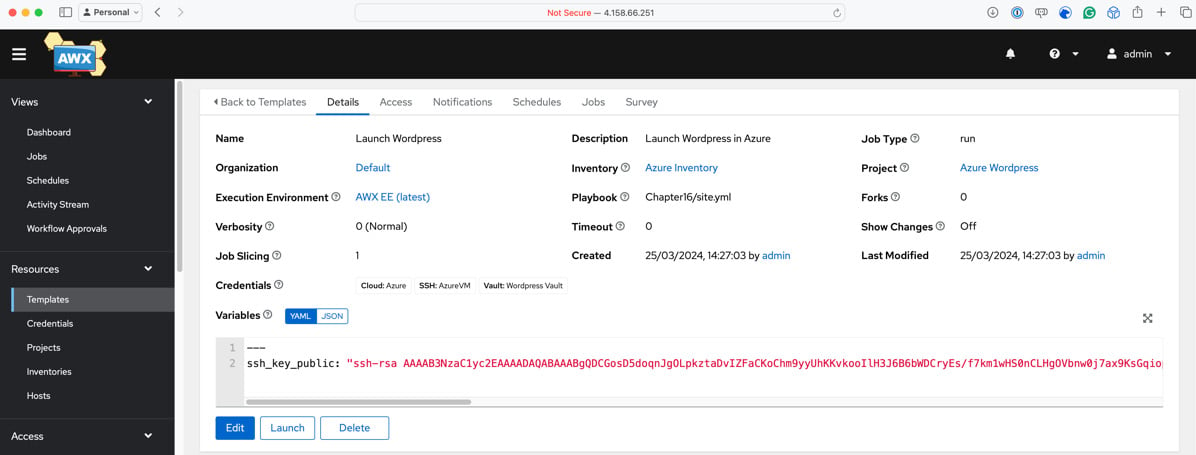


Figure 16.9 – The completed template

Once added, we need to repeat this process with the following details for the Playbook that terminates our deployment:

* **Name**: Terminate WordPress
* **Description**: Terminate WordPress in Azure
* **Job Type**: Select **Run**
* **Inventory**: Select **Azure Inventory**
* **Project**: Select **Azure WordPress**
* **Execution Environment**: Select **AWX EE (latest)**
* **Playbook**: Choose **Chapter16/destroy.yml** from the drop-down list
* **Credentials**: Select the following:
  + **Microsoft Azure Resource Manager**: **Azure**
* **Remaining options**: Leave blank

Once you’ve filled in these details, click **Save**.

We have everything we need to run our playbooks, so let’s do that.

Running our playbooks

Back on the **Templates** page, you should see the two templates we have configured listed:

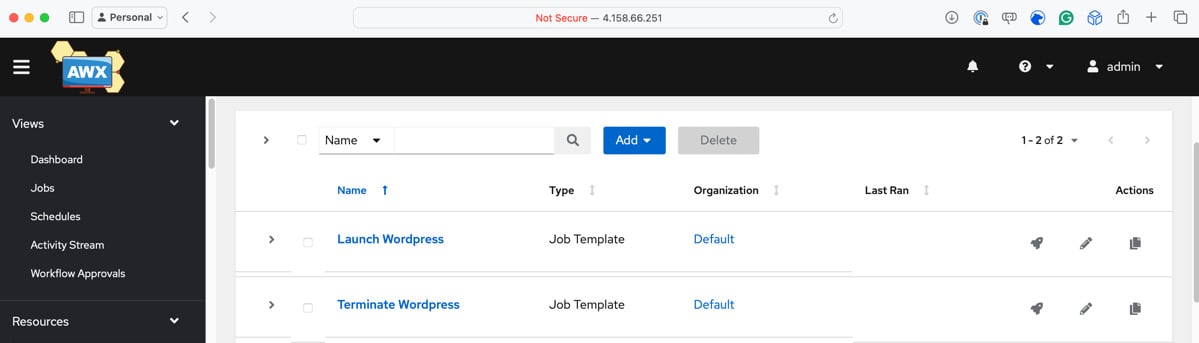


Figure 16.10 – Our two templates

To run the playbook from this page, click on the **Rocket** icon on the **Launch WordPress** template; this will initiate the playbook run and take you to a job page where you will be able to review the status of the playbook job:

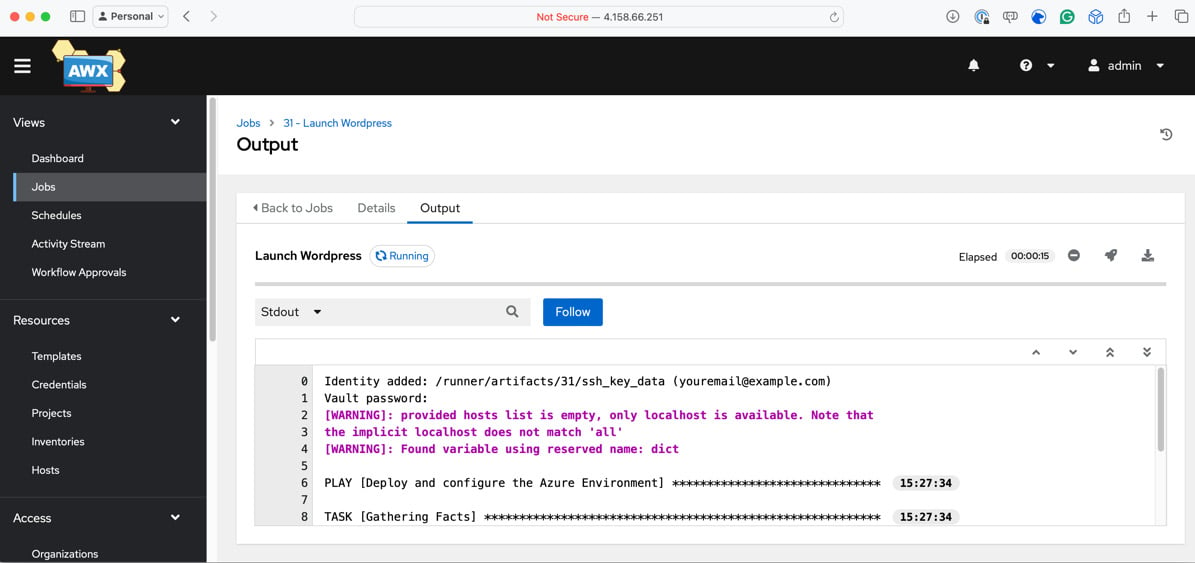


Figure 16.11 – Launching WordPress in Azure using Ansible AWX

If everything has worked as planned, after about 5 minutes, you should get confirmation that the playbook has been completed and that your resources have been launched:

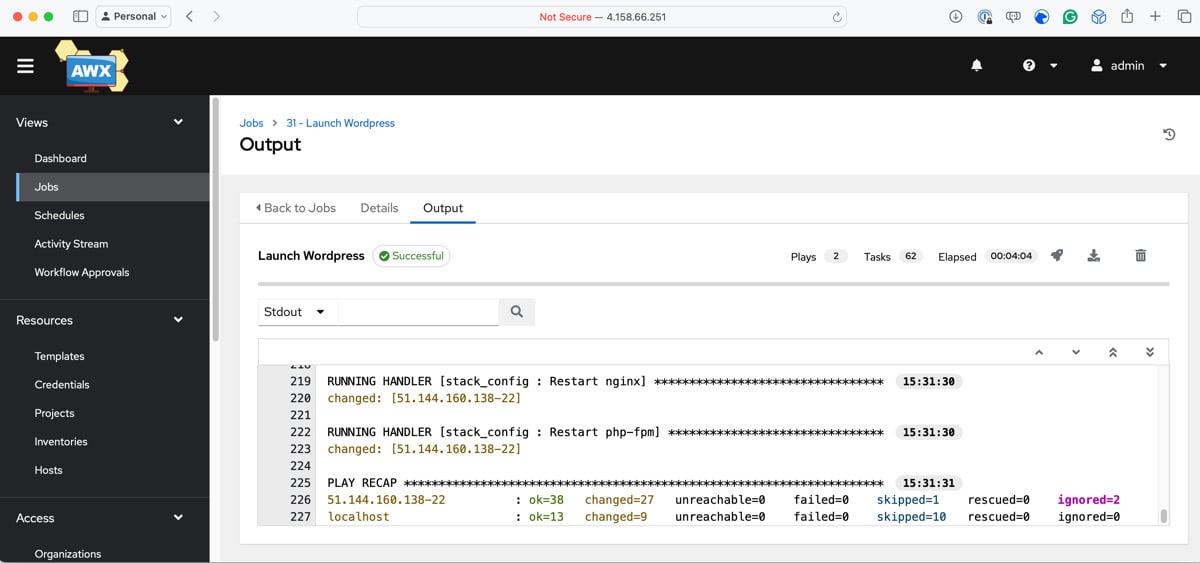


Figure 16.12 – Ansible AWX has finished running the playbook

From here, you can re-run the launch playbook again, and it should pick up the newly deployed resources as it did when we re-ran the playbook on our local machine.

Given the number of Azure resources we have launched, before we review the changes to the playbook code to run it in Ansible AWX, we should terminate the WordPress resources. Click on the **Rocket** icon next to the **Terminate WordPress** template to tear down the resources we just launched.

Terminating the Kubernetes cluster

Before terminating the Azure AKS resources, I recommend clicking around and exploring the Ansible AWX interface. Once you’ve finished, you can remove the Azure resources and tidy up your local configuration by running the following commands:

$ AKSRG=rg-awx-cluster  
$ AKSCLUSTER=aks-awx-cluster  
$ az aks delete --resource-group $AKSRG --name $AKSCLUSTER  
$ az group delete --name $AKSRG  
$ kubectl config delete-cluster $AKSCLUSTER  
$ kubectl config delete-context $AKSCLUSTER

The cluster will take about 5 minutes to remove. To be safe, please don’t close any windows until it has finished.

Information

As always, please double-check that your cloud resources have been terminated – you don’t want to incur any unexpected costs.

Now that we’ve terminated all the cost-incurring resources, let’s discuss some of the considerations we had to make in our Playbook.

Playbook considerations

While we touched very lightly on some of the changes that we had to make to our playbook so that it runs on Ansible AWX, let’s do a deeper dive now.

Changes to the existing playbook

As we were running the code locally, to keep the playbook simple, we created a file called secrets.yml and loaded the variables from there. Now that we are running Ansible in a shared environment, we should treat our Ansible execution environment as if it were ephemeral, meaning that we cannot rely on this approach.

I used Ansible Vault to encrypt the passwords and ship them within the code to get around this. To do this, I ran the following commands:

$ ansible-vault encrypt\_string 'SomeP4ssw0rd4MySQL' --name 'db\_password'

$ ansible-vault encrypt\_string 'aP455w0rd4W0rDPR355' --name 'wp\_password'

When prompted to enter the Vault password, I entered wibble as the password, which we then set the Vault password in Ansible AWX when adding credentials. You can see the results of the preceding command in the group\_vars/common.yml file.

Going back to the playbook code when we ran the playbook from our local machine in [*Chapter 9*](https://subscription.packtpub.com/book/cloud-and-networking/9781835088913/9), *Moving to the Cloud*, the variable that contains the data for the public SSH key looked like this:

vm\_config:

  key:

    path: "/home/adminuser/.ssh/authorized\_keys"

    data: "{{ lookup('file', '~/.ssh/id\_rsa.pub') }}"

As you can see, we populate the vm\_config.key.data variable by reading in the contents of the ~/.ssh/id\_rsa.pub file. However, when we moved our playbook to Ansible AWX, this file no longer exists.

Because of that, we added the ssh\_key\_public variable, which contains the public portion of the private key we uploaded when adding the machine credential when we configured the template. This launches the resources in Azure. This meant the code needed to be updated to the following:

vm\_config:

  key:

    path: "/home/adminuser/.ssh/authorized\_keys"

    data: "{{ ssh\_key\_public }}"

As far as changes go, there’s nothing too dramatic and hopefully not unexpected.

Ansible Galaxy collections

You may not have noticed, but we didn’t have to consider the modules that interact with Azure, one of the initial things we covered in [*Chapter 9*](https://subscription.packtpub.com/book/cloud-and-networking/9781835088913/9), *Moving to the Cloud*.

Ansible AWX does not support these and other collections of modules we need for our playbook to run out of the box, so how did our playbook work without giving an error?

When we first added the project, we configured it to use the GitHub repository that supports this book and contains all the code we have discussed so far. This repository can be found at https://github.com/packtPublishing/Learn-Ansible-Second-Edition/.

We only instructed Ansible AWX to use the site.yml and destory.yml files from the Chapter16 folder, but in the background, Ansible AWX also used the requirements.yml file, which can be found in the collections folder in the repository’s root.

This file contains the following code:

---

collections:

  - name: "azure.azcollection"

    source: "https://galaxy.ansible.com"

  - name: "community.general"

    source: "https://galaxy.ansible.com"

  - name: "community.mysql"

    source: "https://galaxy.ansible.com"

As you can see, this is letting Ansible AWX know that it needs to download the azure.azcollection, community.general and community.mysql collections from Ansible Galaxy and, in the background, install their prerequisites.

The only thing we needed to do to get this to work was to create the Ansible Galaxy credential and attach it to our default organization. This means that whenever Ansible AWX comes across a collections/requirements.yml file, it will authenticate against Ansible Galaxy using the credentials provided, which in our case were anonymous as we weren’t pulling a private collection.

We can also do things such as pin collections to a particular version or add a role:

collections:

  - name: "azure.azcollection"

    source: "https://galaxy.ansible.com"

    version: 2.0.0

roles:

  - name: "russmckendrick.learnansible\_example"

    source: "https://galaxy.ansible.com"

You can also provide different URLs if you are self-hosting an installation of Ansible Galaxy or even provide links to Git repos containing your roles and collections.

This means that Ansible AWX can be as flexible as running Ansible from your local machine.

Before we finish discussing Ansible AWX, let’s look at the pros and cons of running it.

Ansible AWX’s advantages and disadvantages

I am sure you will agree from our time with Ansible AWX that it looks like a great tool. However, there are some advantages and disadvantages to running it.

Open source

Ansible AWX is an open source project, which means it is freely available for anyone to use, modify, and contribute to. This can significantly reduce costs compared to proprietary solutions. However, it has limited Enterprise features.

Ansible AWX offers a good range of features. Still, some advanced enterprise-specific capabilities in Red Hat Ansible Automation Platform, such as advanced reporting, **service-level agreement** (**SLA**) management, and more comprehensive integrations, may be needed.

Community-driven development

Being open source, Ansible AWX has a strong community of developers and users actively contributing to its development, providing support, and sharing best practices.

However, as an open source project, Ansible AWX relies on community support rather than official commercial support. The community is generally active and helpful, but there are no guaranteed response times or even that someone will be able to help outside of the commercial Red Hat Ansible Automation Platform offering.

Frequent updates and improvements

Ansible AWX follows a more frequent release cycle than Red Hat Automation Platform. This means that you can gain access to new features, bug fixes, and improvements more quickly.

Ansible AWX’s frequent release cycle means you may need to update more often to access the latest features and bug fixes. Upgrading Ansible AWX can require more effort to ensure compatibility and stability, especially in production environments.

Updates and Ansible AWX have always been challenging; they have always been more of a migration than an in-place update.

Using our quick deployment of Ansible AWX as an example, we would need a way to upgrade it. We would have to deploy an external database server outside of our Kubernetes cluster for a more production-like environment – this would contain and persist all our data and configuration.

To *update* Ansible AWX, we would need to tear down all of the resources in the cluster (minus the database), update the AWX Operator, and then redeploy Ansible AWX running the latest version – this would then connect to our external database and run all of the necessary database migration scripts to update our schema and data to make it compatible with the new version.

Solid foundation

Ansible AWX provides robust features for managing and executing Ansible playbooks, making it a solid choice for organizations starting their automation journey or having more straightforward automation requirements.

Flexibility and customization

While Ansible AWX integrates with various tools and systems, it may have a different level of out-of-the-box integrations and certified content than Red Hat Automation Platform, which is designed to work seamlessly with other Red Hat products and has a broader ecosystem of supported integrations.

Ansible AWX may also have limitations when managing large-scale deployments or complex enterprise environments. Additional setup, configuration, and resources may be required to handle high-volume automation tasks effectively.

Summary

This chapter explored Ansible AWX and touched upon Red Hat Automation Platform, two powerful graphical interfaces for managing and streamlining Ansible deployments.

We learned about their differences, the benefits they offer, and how to install and configure Ansible AWX on a Kubernetes cluster in Microsoft Azure. We successfully ran our playbook to launch and terminate WordPress running in Azure using Ansible AWX by setting up a project, credentials, inventory, and templates.

Throughout the process, we discovered the necessary playbook considerations and modifications, such as using Ansible Vault for sensitive information, handling SSH keys, and leveraging Ansible Galaxy collections.

While Ansible AWX offers numerous advantages, including its open source nature, community-driven development, and solid foundation, it is essential to be aware of its potential limitations in enterprise environments and the challenges associated with updating the platform.

The only thing we didn’t discuss was the costs of running the commercially supported enterprise-grade Red Hat Automation Platform. Red Hat does not publicly publish them on its website. You must contact one of its partners or Red Hat directly for details.

In our next and final chapter, we will look at some of the ways you can integrate Ansible into your daily workflows, debug your playbooks as they run, and some real-world examples of how I have used Ansible.