**Installing AWX**

This document provides a guide for installing AWX.

**Table of contents**

* [Getting started](https://github.com/ansible/awx/blob/devel/INSTALL.md#getting-started)
  + [Clone the repo](https://github.com/ansible/awx/blob/devel/INSTALL.md#clone-the-repo)
  + [AWX branding](https://github.com/ansible/awx/blob/devel/INSTALL.md#awx-branding)
  + [Prerequisites](https://github.com/ansible/awx/blob/devel/INSTALL.md#prerequisites)
  + [System Requirements](https://github.com/ansible/awx/blob/devel/INSTALL.md#system-requirements)
  + [AWX Tunables](https://github.com/ansible/awx/blob/devel/INSTALL.md#awx-tunables)
  + [Choose a deployment platform](https://github.com/ansible/awx/blob/devel/INSTALL.md#choose-a-deployment-platform)
  + [Official vs Building Images](https://github.com/ansible/awx/blob/devel/INSTALL.md#official-vs-building-images)
* [OpenShift](https://github.com/ansible/awx/blob/devel/INSTALL.md#openshift)
  + [Prerequisites](https://github.com/ansible/awx/blob/devel/INSTALL.md#prerequisites-1)
    - [Deploying to Minishift](https://github.com/ansible/awx/blob/devel/INSTALL.md#deploying-to-minishift)
  + [Pre-build steps](https://github.com/ansible/awx/blob/devel/INSTALL.md#pre-build-steps)
  + [PostgreSQL](https://github.com/ansible/awx/blob/devel/INSTALL.md#postgresql)
  + [Start the build](https://github.com/ansible/awx/blob/devel/INSTALL.md#start-the-build)
  + [Post build](https://github.com/ansible/awx/blob/devel/INSTALL.md#post-build)
  + [Accessing AWX](https://github.com/ansible/awx/blob/devel/INSTALL.md#accessing-awx)
* [Kubernetes](https://github.com/ansible/awx/blob/devel/INSTALL.md#kubernetes)
  + [Prerequisites](https://github.com/ansible/awx/blob/devel/INSTALL.md#prerequisites-2)
  + [Pre-build steps](https://github.com/ansible/awx/blob/devel/INSTALL.md#pre-build-steps-1)
  + [Start the build](https://github.com/ansible/awx/blob/devel/INSTALL.md#start-the-build-1)
  + [Accessing AWX](https://github.com/ansible/awx/blob/devel/INSTALL.md#accessing-awx-1)
  + [SSL Termination](https://github.com/ansible/awx/blob/devel/INSTALL.md#ssl-termination)
* [Docker and Docker Compose](https://github.com/ansible/awx/blob/devel/INSTALL.md#docker-and-docker-compose)
  + [Prerequisites](https://github.com/ansible/awx/blob/devel/INSTALL.md#prerequisites-3)
  + [Pre-build steps](https://github.com/ansible/awx/blob/devel/INSTALL.md#pre-build-steps-2)
    - [Deploying to a remote host](https://github.com/ansible/awx/blob/devel/INSTALL.md#deploying-to-a-remote-host)
    - [Inventory variables](https://github.com/ansible/awx/blob/devel/INSTALL.md#inventory-variables)
      * [Docker registry](https://github.com/ansible/awx/blob/devel/INSTALL.md#docker-registry)
      * [PostgreSQL](https://github.com/ansible/awx/blob/devel/INSTALL.md#postgresql-1)
      * [Proxy settings](https://github.com/ansible/awx/blob/devel/INSTALL.md#proxy-settings)
  + [Start the build](https://github.com/ansible/awx/blob/devel/INSTALL.md#start-the-build-2)
  + [Post build](https://github.com/ansible/awx/blob/devel/INSTALL.md#post-build-1)
  + [Accessing AWX](https://github.com/ansible/awx/blob/devel/INSTALL.md#accessing-awx-2)

**Getting started**

**Clone the repo**

If you have not already done so, you will need to clone, or create a local copy, of the [AWX repo](https://github.com/ansible/awx). For more on how to clone the repo, view [git clone help](https://git-scm.com/docs/git-clone).

Once you have a local copy, run commands within the root of the project tree.

**AWX branding**

You can optionally install the AWX branding assets from the [awx-logos repo](https://github.com/ansible/awx-logos). Prior to installing, please review and agree to the [trademark guidelines](https://github.com/ansible/awx-logos/blob/master/TRADEMARKS.md).

To install the assets, clone the awx-logos repo so that it is next to your awx clone. As you progress through the installation steps, you'll be setting variables in the [inventory](https://github.com/ansible/awx/blob/devel/installer/inventory) file. To include the assets in the build, set awx\_official=true.

**Prerequisites**

Before you can run a deployment, you'll need the following installed in your local environment:

* [Ansible](http://docs.ansible.com/ansible/latest/intro_installation.html) Requires Version 2.4+
* [Docker](https://docs.docker.com/engine/installation/)
* [docker-py](https://github.com/docker/docker-py) Python module
* [GNU Make](https://www.gnu.org/software/make/)
* [Git](https://git-scm.com/) Requires Version 1.8.4+

**System Requirements**

The system that runs the AWX service will need to satisfy the following requirements

* At leasts 4GB of memory
* At least 2 cpu cores
* At least 20GB of space
* Running Docker, Openshift, or Kubernetes

**AWX Tunables**

**TODO** add tunable bits

**Choose a deployment platform**

We currently support running AWX as a containerized application using Docker images deployed to either an OpenShift cluster, docker-compose or a standalone Docker daemon. The remainder of this document will walk you through the process of building the images, and deploying them to either platform.

The [installer](https://github.com/ansible/awx/blob/devel/installer) directory contains an [inventory](https://github.com/ansible/awx/blob/devel/installer/inventory) file, and a playbook, [install.yml](https://github.com/ansible/awx/blob/devel/installer/install.yml). You'll begin by setting variables in the inventory file according to the platform you wish to use, and then you'll start the image build and deployment process by running the playbook.

In the sections below, you'll find deployment details and instructions for each platform:

* [Docker and Docker Compose](https://github.com/ansible/awx/blob/devel/INSTALL.md#docker-and-docker-compose)
* [OpenShift](https://github.com/ansible/awx/blob/devel/INSTALL.md#openshift)
* [Kubernetes](https://github.com/ansible/awx/blob/devel/INSTALL.md#kubernetes).

**Official vs Building Images**

When installing AWX you have the option of building your own images or using the images provided on DockerHub (see[awx\_web](https://hub.docker.com/r/ansible/awx_web/) and [awx\_task](https://hub.docker.com/r/ansible/awx_task/))

This is controlled by the following variables in the inventory file

dockerhub\_base=ansible

dockerhub\_version=latest

If these variables are present then all deployments will use these hosted images. If the variables are not present then the images will be built during the install.

*dockerhub\_base*

The base location on DockerHub where the images are hosted (by default this pulls container images namedansible/awx\_web:tag and ansible/awx\_task:tag)

*dockerhub\_version*

Multiple versions are provided. latest always pulls the most recent. You may also select version numbers at different granularities: 1, 1.0, 1.0.1, 1.0.0.123

**OpenShift**

**Prerequisites**

To complete a deployment to OpenShift, you will obviously need access to an OpenShift cluster. For demo and testing purposes, you can use [Minishift](https://github.com/minishift/minishift) to create a single node cluster running inside a virtual machine.

You will also need to have the oc command in your PATH. The install.yml playbook will call out to oc when logging into, and creating objects on the cluster.

**Deploying to Minishift**

Install Minishift by following the [installation guide](https://docs.openshift.org/latest/minishift/getting-started/installing.html).

The Minishift VM contains a Docker daemon, which you can use to build the AWX images. This is generally the approach you should take, and we recommend doing so. To use this instance, run the following command to setup your environment:

# Set DOCKER environment variable to point to the Minishift VM

$ eval $(minishift docker-env)

**Note**

If you choose to not use the Docker instance running inside the VM, and build the images externally, you will have to enable the OpenShift cluster to access the images. This involves pushing the images to an external Docker registry, and granting the cluster access to it, or exposing the internal registry, and pushing the images into it.

**Pre-build steps**

Before starting the build process, review the [inventory](https://github.com/ansible/awx/blob/devel/installer/inventory) file, and uncomment and provide values for the following variables found in the [all:vars] section:

*openshift\_host*

IP address or hostname of the OpenShift cluster. If you're using Minishift, this will be the value returned by minishift ip.

*awx\_openshift\_project*

Name of the OpenShift project that will be created, and used as the namespace for the AWX app. Defaults to *awx*.

*openshift\_user*

Username of the OpenShift user that will create the project, and deploy the application. Defaults to *developer*.

*docker\_registry*

IP address and port, or URL, for accessing a registry that the OpenShift cluster can access. Defaults to *172.30.1.1:5000*, the internal registry delivered with Minishift. This is not needed if you are using official hosted images.

*docker\_registry\_repository*

Namespace to use when pushing and pulling images to and from the registry. Generally this will match the project name. It defaults to *awx*. This is not needed if you are using official hosted images.

*docker\_registry\_username*

Username of the user that will push images to the registry. Will generally match the *openshift\_user* value. Defaults to*developer*. This is not needed if you are using official hosted images.

**PostgreSQL**

AWX requires access to a PostgreSQL database, and by default, one will be created and deployed in a pod. The database is configured for persistence and will create a persistent volume claim named postgresql. By default it will claim 5GB from the available persistent volume pool. This can be tuned by setting a variable in the inventory file or on the command line during the ansible-playbook run.

ansible-playbook ... -e pg\_volume\_capacity=n

If you wish to use an external database, in the inventory file, set the value of pg\_hostname, and update pg\_username,pg\_password, pg\_database, and pg\_port with the connection information. When setting pg\_hostname the installer will assume you have configured the database in that location and will not launch the postgresql pod.

**Start the build**

To start the build, you will pass two *extra* variables on the command line. The first is *openshift\_password*, which is the password for the *openshift\_user*, and the second is *docker\_registry\_password*, which is the password associated with*docker\_registry\_username*.

If you're using the OpenShift internal registry, then you'll pass an access token for the *docker\_registry\_password* value, rather than a password. The oc whoami -t command will generate the required token, as long as you're logged into the cluster viaoc cluster login.

To start the build and deployment, run the following (docker\_registry\_password is optional if using official images):

# Start the build and deployment

$ ansible-playbook -i inventory install.yml -e openshift\_password=developer -e docker\_registry\_password=$(oc whoami -t)

**Post build**

After the playbook run completes, check the status of the deployment by running oc get pods:

# View the running pods

$ oc get pods

NAME READY STATUS RESTARTS AGE

awx-3886581826-5mv0l 4/4 Running 0 8s

postgresql-1-l85fh 1/1 Running 0 20m

In the above example, the name of the AWX pod is awx-3886581826-5mv0l. Before accessing the AWX web interface, setup tasks and database migrations need to complete. These tasks are running in the awx\_task container inside the AWX pod. To monitor their status, tail the container's STDOUT by running the following command, replacing the AWX pod name with the pod name from your environment:

# Follow the awx\_task log output

$ oc logs -f awx-3886581826-5mv0l -c awx-celery

You will see the following indicating that database migrations are running:

Using /etc/ansible/ansible.cfg as config file

127.0.0.1 | SUCCESS => {

"changed": false,

"db": "awx"

}

Operations to perform:

Synchronize unmigrated apps: solo, api, staticfiles, messages, channels, django\_extensions, ui, rest\_framework, polymorphic

Apply all migrations: sso, taggit, sessions, djcelery, sites, kombu\_transport\_django, social\_auth, contenttypes, auth, conf, main

Synchronizing apps without migrations:

Creating tables...

Running deferred SQL...

Installing custom SQL...

Running migrations:

Rendering model states... DONE

Applying contenttypes.0001\_initial... OK

Applying contenttypes.0002\_remove\_content\_type\_name... OK

Applying auth.0001\_initial... OK

Applying auth.0002\_alter\_permission\_name\_max\_length... OK

Applying auth.0003\_alter\_user\_email\_max\_length... OK

Applying auth.0004\_alter\_user\_username\_opts... OK

Applying auth.0005\_alter\_user\_last\_login\_null... OK

Applying auth.0006\_require\_contenttypes\_0002... OK

Applying taggit.0001\_initial... OK

Applying taggit.0002\_auto\_20150616\_2121... OK

...

When you see output similar to the following, you'll know that database migrations have completed, and you can access the web interface:

Python 2.7.5 (default, Nov 6 2016, 00:28:07)

[GCC 4.8.5 20150623 (Red Hat 4.8.5-11)] on linux2

Type "help", "copyright", "credits" or "license" for more information.

(InteractiveConsole)

>>> <User: admin>

>>> Default organization added.

Demo Credential, Inventory, and Job Template added.

Successfully registered instance awx-3886581826-5mv0l

(changed: True)

Creating instance group tower

Added instance awx-3886581826-5mv0l to tower

Once database migrations complete, the web interface will be accessible.

**Accessing AWX**

The AWX web interface is running in the AWX pod, behind the awx-web-svc service. To view the service, and its port value, run the following command:

# View available services

$ oc get services

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE

awx-web-svc 172.30.111.74 <nodes> 8052:30083/TCP 37m

postgresql 172.30.102.9 <none> 5432/TCP 38m

The deployment process creates a route, awx-web-svc, to expose the service. How the ingres is actually created will vary depending on your environment, and how the cluster is configured. You can view the route, and the external IP address and hostname assigned to it, by running the following command:

# View available routes

$ oc get routes

NAME HOST/PORT PATH SERVICES PORT TERMINATION WILDCARD

awx-web-svc awx-web-svc-awx.192.168.64.2.nip.io awx-web-svc http edge/Allow None

The above example is taken from a Minishift instance. From a web browser, use https to access the HOST/PORT value from your environment. Using the above example, the URL to access the server would be [https://awx-web-svc-awx.192.168.64.2.nip.io](https://awx-web-svc-awx.192.168.64.2.nip.io/).

Once you access the AWX server, you will be prompted with a login dialog. The default administrator username is admin, and the password is password.

**Kubernetes**

**Prerequisites**

A Kubernetes deployment will require you to have access to a Kubernetes cluster as well as the following tools:

* [kubectl](https://kubernetes.io/docs/tasks/tools/install-kubectl/)
* [helm](https://docs.helm.sh/using_helm/#quickstart-guide)

The installation program will reference kubectl directly. helm is only necessary if you are letting the installer configure PostgreSQL for you.

**Pre-build steps**

Before starting the build process, review the [inventory](https://github.com/ansible/awx/blob/devel/installer/inventory) file, and uncomment and provide values for the following variables found in the [all:vars] section uncommenting when necessary. Make sure the openshift and standalone docker sections are commented out:

*kubernetes\_context*

Prior to running the installer, make sure you've configured the context for the cluster you'll be installing to. This is how the installer knows which cluster to connect to and what authentication to use

*awx\_kubernetes\_namespace*

Name of the Kubernetes namespace where the AWX resources will be installed. This will be created if it doesn't exist

*docker\_registry\_*

These settings should be used if building your own base images. You'll need access to an external registry and are responsible for making sure your kube cluster can talk to it and use it. If these are undefined and the dockerhub\_ configuration settings are uncommented then the images will be pulled from dockerhub instead

**Start the build**

After making changes to the inventory file use ansible-playbook to begin the install

$ ansible-playbook -i inventory install.yml

**Post build**

After the playbook run completes, check the status of the deployment by running kubectl get pods --namespace awx(replace awx with the namespace you used):

# View the running pods, it may take a few minutes for everything to be marked in the Running state

$ kubectl get pods --namespace awx

NAME READY STATUS RESTARTS AGE

awx-2558692395-2r8ss 4/4 Running 0 29s

awx-postgresql-355348841-kltkn 1/1 Running 0 1m

**Accessing AWX**

The AWX web interface is running in the AWX pod behind the awx-web-svc service:

# View available services

$ kubectl get svc --namespace awx

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

awx-postgresql ClusterIP 10.7.250.208 <none> 5432/TCP 2m

awx-web-svc NodePort 10.7.241.35 <none> 80:30177/TCP 1m

The deployment process creates an Ingress named awx-web-svc also. Some kubernetes cloud providers will automatically handle routing configuration when an Ingress is created others may require that you more explicitly configure it. You can see what kubernetes knows about things with:

kubectl get ing --namespace awx

NAME HOSTS ADDRESS PORTS AGE

awx-web-svc \* 35.227.x.y 80 3m

If your provider is able to allocate an IP Address from the Ingress controller then you can navigate to the address and access the AWX interface. For some providers it can take a few minutes to allocate and make this accessible. For other providers it may require you to manually intervene.

**SSL Termination**

Unlike Openshift's Route the Kubernetes Ingress doesn't yet handle SSL termination. As such the default configuration will only expose AWX through HTTP on port 80. You are responsible for configuring SSL support until support is added (either to Kubernetes or AWX itself).

**Docker or Docker-Compose**

**Prerequisites**

* [Docker](https://docs.docker.com/engine/installation/) on the host where AWX will be deployed. After installing Docker, the Docker service must be started (depending on your OS, you may have to add the local user that uses Docker to the docker group, refer to the documentation for details)
* [docker-py](https://github.com/docker/docker-py) Python module.

If you're installing using Docker Compose, you'll need [Docker Compose](https://docs.docker.com/compose/install/).

**Pre-build steps**

**Deploying to a remote host**

By default, the delivered [installer/inventory](https://github.com/ansible/awx/blob/devel/installer/inventory) file will deploy AWX to the local host. It is possible; however, to deploy to a remote host. The [installer/install.yml](https://github.com/ansible/awx/blob/devel/installer/install.yml) playbook can be used to build images on the local host, and ship the built images to, and run deployment tasks on, a remote host. To do this, modify the [installer/inventory](https://github.com/ansible/awx/blob/devel/installer/inventory) file, by commenting out localhost, and adding the remote host.

For example, suppose you wish to build images locally on your CI/CD host, and deploy them to a remote host named *awx-server*. To do this, add *awx-server* to the [installer/inventory](https://github.com/ansible/awx/blob/devel/installer/inventory) file, and comment out or remove localhost, as demonstrated by the following:

# localhost ansible\_connection=local

awx-server

[all:vars]

...

In the above example, image build tasks will be delegated to localhost, which is typically where the clone of the AWX project exists. Built images will be archived, copied to remote host, and imported into the remote Docker image cache. Tasks to start the AWX containers will then execute on the remote host.

If you choose to use the official images then the remote host will be the one to pull those images.

**Note**

You may also want to set additional variables to control how Ansible connects to the host. For more information about this, view [Behavioral Inventory Parameters](http://docs.ansible.com/ansible/latest/intro_inventory.html#id12).

As mentioned above, in [Prerequisites](https://github.com/ansible/awx/blob/devel/INSTALL.md#prerequisites-1), the prerequisites are required on the remote host.

When deploying to a remote host, the playook does not execute tasks with the become option. For this reason, make sure the user that connects to the remote host has privileges to run the docker command. This typically means that non-privileged users need to be part of the docker group.

**Inventory variables**

Before starting the build process, review the [inventory](https://github.com/ansible/awx/blob/devel/installer/inventory) file, and uncomment and provide values for the following variables found in the [all:vars] section:

*postgres\_data\_dir*

If you're using the default PostgreSQL container (see [PostgreSQL](https://github.com/ansible/awx/blob/devel/INSTALL.md#postgresql-1) below), provide a path that can be mounted to the container, and where the database can be persisted.

*host\_port*

Provide a port number that can be mapped from the Docker daemon host to the web server running inside the AWX container. Defaults to *80*.

*use\_docker\_compose*

Switch to true to use Docker Compose instead of the standalone Docker install.

*docker\_compose\_dir*

When using docker-compose, the docker-compose.yml file will be created there (default /var/lib/awx).

**Docker registry**

If you wish to tag and push built images to a Docker registry, set the following variables in the inventory file:

*docker\_registry*

IP address and port, or URL, for accessing a registry.

*docker\_registry\_repository*

Namespace to use when pushing and pulling images to and from the registry. Defaults to *awx*.

*docker\_registry\_username*

Username of the user that will push images to the registry. Defaults to *developer*.

*docker\_remove\_local\_images*

Due to the way that the docker\_image module behaves, images will not be pushed to a remote repository if they are present locally. Set this to delete local versions of the images that will be pushed to the remote. This will fail if containers are currently running from those images.

**Note**

These settings are ignored if using official images

**Proxy settings**

*http\_proxy*

IP address and port, or URL, for using an http\_proxy.

*https\_proxy*

IP address and port, or URL, for using an https\_proxy.

*no\_proxy*

Exclude IP address or URL from the proxy.

**PostgreSQL**

AWX requires access to a PostgreSQL database, and by default, one will be created and deployed in a container, and data will be persisted to a host volume. In this scenario, you must set the value of postgres\_data\_dir to a path that can be mounted to the container. When the container is stopped, the database files will still exist in the specified path.

If you wish to use an external database, in the inventory file, set the value of pg\_hostname, and update pg\_username,pg\_password, pg\_database, and pg\_port with the connection information.

**Start the build**

If you are not pushing images to a Docker registry, start the build by running the following:

# Set the working directory to installer

$ cd installer

# Run the Ansible playbook

$ ansible-playbook -i inventory install.yml

If you're pushing built images to a repository, then use the -e option to pass the registry password as follows, replacing*password* with the password of the username assigned to docker\_registry\_username (note that you will also need to removedockerhub\_base and dockerhub\_version from the inventory file):

# Set the working directory to installer

$ cd installer

# Run the Ansible playbook

$ ansible-playbook -i inventory -e docker\_registry\_password=password install.yml

**Post build**

After the playbook run completes, Docker will report up to 5 running containers. If you chose to use an existing PostgresSQL database, then it will report 4. You can view the running containers using the docker ps command, as follows:

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

e240ed8209cd awx\_task:1.0.0.8 "/tini -- /bin/sh ..." 2 minutes ago Up About a minute 8052/tcp awx\_task

1cfd02601690 awx\_web:1.0.0.8 "/tini -- /bin/sh ..." 2 minutes ago Up About a minute 0.0.0.0:80->8052/tcp awx\_web

55a552142bcd memcached:alpine "docker-entrypoint..." 2 minutes ago Up 2 minutes 11211/tcp memcached

84011c072aad rabbitmq:3 "docker-entrypoint..." 2 minutes ago Up 2 minutes 4369/tcp, 5671-5672/tcp, 25672/tcp rabbitmq

97e196120ab3 postgres:9.6 "docker-entrypoint..." 2 minutes ago Up 2 minutes 5432/tcp postgres

If you're deploying using Docker Compose, container names will be prefixed by the name of the folder where the docker-compose.yml file is created (by default, awx).

Immediately after the containers start, the *awx\_task* container will perform required setup tasks, including database migrations. These tasks need to complete before the web interface can be accessed. To monitor the progress, you can follow the container's STDOUT by running the following:

# Tail the the awx\_task log

$ docker logs -f awx\_task

You will see output similar to the following:

Using /etc/ansible/ansible.cfg as config file

127.0.0.1 | SUCCESS => {

"changed": false,

"db": "awx"

}

Operations to perform:

Synchronize unmigrated apps: solo, api, staticfiles, messages, channels, django\_extensions, ui, rest\_framework, polymorphic

Apply all migrations: sso, taggit, sessions, djcelery, sites, kombu\_transport\_django, social\_auth, contenttypes, auth, conf, main

Synchronizing apps without migrations:

Creating tables...

Running deferred SQL...

Installing custom SQL...

Running migrations:

Rendering model states... DONE

Applying contenttypes.0001\_initial... OK

Applying contenttypes.0002\_remove\_content\_type\_name... OK

Applying auth.0001\_initial... OK

Applying auth.0002\_alter\_permission\_name\_max\_length... OK

Applying auth.0003\_alter\_user\_email\_max\_length... OK

Applying auth.0004\_alter\_user\_username\_opts... OK

Applying auth.0005\_alter\_user\_last\_login\_null... OK

Applying auth.0006\_require\_contenttypes\_0002... OK

Applying taggit.0001\_initial... OK

Applying taggit.0002\_auto\_20150616\_2121... OK

Applying main.0001\_initial... OK

...

Once migrations complete, you will see the following log output, indicating that migrations have completed:

Python 2.7.5 (default, Nov 6 2016, 00:28:07)

[GCC 4.8.5 20150623 (Red Hat 4.8.5-11)] on linux2

Type "help", "copyright", "credits" or "license" for more information.

(InteractiveConsole)

>>> <User: admin>

>>> Default organization added.

Demo Credential, Inventory, and Job Template added.

Successfully registered instance awx

(changed: True)

Creating instance group tower

Added instance awx to tower

(changed: True)

...

**Accessing AWX**

The AWX web server is accessible on the deployment host, using the *host\_port* value set in the *inventory* file. The default URL is [http://localhost](http://localhost/).

You will prompted with a login dialog. The default administrator username is admin, and the password is password.

**Maintenance using docker-compose**

After the installation, maintenance operations with docker-compose can be done by using the docker-compose.yml file created at the location pointed by docker\_compose\_dir.

Among the possible operations, you may:

* Stop AWX : docker-compose stop
* Upgrade AWX : docker-compose pull && docker-compose up --force-recreate

See the [docker-compose documentation](https://docs.docker.com/compose/) for details.