

[Home](#) / [Ansible](#) / [Install Ansible AWX on Ubuntu 24.04 / Debian...](#)

ANSIBLE

Install Ansible AWX on Ubuntu 24.04 / Debian 13



Josphat Mutai

Updated May 12, 2026 · 6 min read

Ansible AWX is the open-source upstream project behind Red Hat Ansible Automation Platform (formerly Ansible Tower). It gives you a web-based UI, REST API, and role-based access control for managing Ansible playbooks at scale. The modern deployment method uses the AWX Operator on Kubernetes. This guide walks through installing AWX on Ubuntu 24.04 or Debian 13 using K3s as a lightweight Kubernetes distribution and the AWX Operator deployed via Kustomize.

Architecture Overview

The deployment stack looks like this:

- **K3s** – A lightweight, certified Kubernetes distribution that runs as a single binary. It



provides the cluster that AWX needs.

- **AWX Operator** – A Kubernetes operator that automates the deployment and lifecycle management of AWX instances. It handles the database, web server, task runner, and Redis components.
- **Kustomize** – A configuration management tool built into kubectl that lets you customize Kubernetes manifests without templating.

Prerequisites

- Ubuntu 24.04 or Debian 13 server (minimum 4 GB RAM, 2 CPUs, 20 GB disk)
- Root or sudo access
- A fully qualified domain name pointing to the server (optional but recommended for TLS)

AWX is resource-hungry. For production workloads, plan for at least 8 GB RAM and 4 CPUs.

Step 1 – Update the System

Start with a fully updated system:



```
sudo apt update && sudo apt upgrade -y
```

Install a few packages that will be needed later:

```
sudo apt install curl git make jq -y
```

Step 2 – Install K3s

K3s installs as a single binary and sets up a fully functional Kubernetes cluster in under a minute. Run the official installer:

```
curl -sfL https://get.k3s.io | sh -
```

Wait for K3s to start, then verify the node is ready:

```
sudo kubectl get nodes
```



Expected output:

NAME	STATUS	ROLES	AGE	VERSION
awx-server	Ready	control-plane,master	30s	v1.31.x+k3s1

Configure kubectl for your regular user so you do not need sudo for every command:

```
mkdir -p ~/.kube
sudo cp /etc/rancher/k3s/k3s.yaml ~/.kube/config
sudo chown $USER:$USER ~/.kube/config
export KUBECONFIG=~/.kube/config
echo 'export KUBECONFIG=~/.kube/config' >> ~/.bashrc
```

Verify kubectl works without sudo:

```
kubectl get nodes
```

Step 3 – Create the AWX Namespace

Keep AWX resources organized in their own namespace:



```
kubectl create namespace awx
```

Verify it exists:

```
kubectl get namespaces
```

Step 4 – Deploy the AWX Operator with Kustomize

Create a working directory for your Kustomize files:

```
mkdir -p ~/awx-deployment  
cd ~/awx-deployment
```

Check the latest AWX Operator release version on the [AWX Operator GitHub releases page](#). At the time of writing, the latest release is 2.19.1. Set it as a variable:

```
AWX_OPERATOR_VERSION=2.19.1
```



Create the `kustomization.yaml` file that tells Kustomize to pull the AWX Operator manifests:

```
cat <<EOF > kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - github.com/ansible/awx-operator/config/default?ref=${AWX_OPERATOR_VERSION}
images:
  - name: quay.io/ansible/awx-operator
    newTag: ${AWX_OPERATOR_VERSION}
  - name: gcr.io/kubebuilder/kube-rbac-proxy
    newName: quay.io/brancz/kube-rbac-proxy
    newTag: v0.18.0
namespace: awx
EOF
```

The second image override is required as of May 2026. AWX Operator 2.19.1 still references `gcr.io/kubebuilder/kube-rbac-proxy:v0.15.0`, but Google retired that registry path and the manifest now returns 404. Without the override, the operator pod sits in `ContainerCreating` with `ErrImagePull`. The [upstream AWX forum thread](#) tracks the issue. `quay.io/brancz/kube-rbac-proxy` is the original maintainer's registry and is a drop-in replacement.

^

Apply the kustomization to deploy the AWX Operator:

```
kubectl apply -k ~/awx-deployment
```

Wait for the operator pod to reach the Running state:

```
kubectl -n awx get pods -w
```

You should see the `awx-operator-controller-manager` pod transition to `Running` with `2/2` containers ready. Press Ctrl+C once it stabilizes.

Step 5 – Deploy the AWX Instance

Now create an AWX custom resource that tells the operator to spin up an actual AWX deployment. Create the manifest:

```
cat <<EOF > awx-instance.yaml
apiVersion: awx.ansible.com/v1beta1
kind: AWX
metadata:
  name: awx
  namespace: awx
```



```
spec:
  service_type: NodePort
  nodeport_port: 30080
EOF
```

Add the instance file to your kustomization. Update `kustomization.yaml` to include it:

```
cat <<EOF > kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - github.com/ansible/awx-operator/config/default?ref=${AWX_OPERATOR_VERSION}
  - awx-instance.yaml
images:
  - name: quay.io/ansible/awx-operator
    newTag: ${AWX_OPERATOR_VERSION}
  - name: gcr.io/kubebuilder/kube-rbac-proxy
    newName: quay.io/brancz/kube-rbac-proxy
    newTag: v0.18.0
namespace: awx
EOF
```

Apply the updated kustomization:



```
kubectl apply -k ~/awx-deployment
```

The operator now creates several resources: a PostgreSQL database pod, a Redis pod, the AWX web and task containers, and supporting services. Monitor the progress:

```
kubectl -n awx get pods -w
```

This process takes several minutes as container images are pulled. Wait until all pods show **Running** and all containers within each pod are ready.

Check all AWX resources:

```
kubectl -n awx get pods,svc,pvc
```

Step 6 – Retrieve the Admin Password

The AWX Operator generates a random admin password and stores it in a Kubernetes secret. Retrieve it:



```
kubectl -n awx get secret awx-admin-password -o jsonpath='{.data.password}' | base64 --decode; echo
```

Save this password. You need it to log in. The default admin username is `admin`.

Step 7 – Access the AWX Web Interface

Since we configured the service type as NodePort on port 30080, the AWX UI is accessible at:

```
http://10.0.1.50:30080
```

Open this URL in your browser. Log in with username `admin` and the password you retrieved in the previous step.

If you have a firewall active, allow the port:

```
sudo ufw allow 30080/tcp
```

Verify the service is listening:



```
kubectl -n awx get svc awx-service
```

Step 8 – Set Up Ingress with TLS (Optional)

For production, expose AWX through an Ingress resource with TLS instead of a raw NodePort. K3s ships with Traefik as the default ingress controller.

First, update your AWX instance to use ClusterIP instead of NodePort:

```
cat <<EOF > awx-instance.yaml
apiVersion: awx.ansible.com/v1beta1
kind: AWX
metadata:
  name: awx
  namespace: awx
spec:
  service_type: ClusterIP
  ingress_type: ingress
  hostname: awx.example.com
EOF
```



Re-apply and let the operator update the deployment:

```
kubectl apply -k ~/awx-deployment
```

Monitoring AWX Operator Logs

If something goes wrong during deployment, the operator logs are the first place to look:

```
kubectl -n awx logs -f deployment/awx-operator-controller-manager -c awx-manager
```

For AWX application logs:

```
kubectl -n awx logs -f deployment/awx-web -c awx-web  
kubectl -n awx logs -f deployment/awx-task -c awx-task
```

Upgrading AWX

To upgrade AWX, update the operator version in your kustomization.yaml, then re-apply:

```
AWX_OPERATOR_VERSION=2.20.0 # set to the new version
```



```
cat <<EOF > kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - github.com/ansible/awx-operator/config/default?ref=${AWX_OPERATOR_VERSION}
  - awx-instance.yaml
images:
  - name: quay.io/ansible/awx-operator
    newTag: ${AWX_OPERATOR_VERSION}
  - name: gcr.io/kubebuilder/kube-rbac-proxy
    newName: quay.io/brancz/kube-rbac-proxy
    newTag: v0.18.0
namespace: awx
EOF

kubectl apply -k ~/awx-deployment
```

The operator handles the rolling upgrade of AWX components automatically.

Troubleshooting

Pods stuck in Pending state – Check node resources with `kubectl describe node`. AWX needs at least 4 GB RAM available. If the node is resource-constrained, the scheduler cannot place the pods.



PostgreSQL pod crashes – Inspect the PVC. If the underlying storage is full or inaccessible, the database pod fails. Check with `kubectl -n awx describe pvc`.

Cannot reach the UI on port 30080 – Confirm the service is running with `kubectl -n awx get svc`. Check that no firewall rule blocks the port. On cloud VMs, also check the security group or network ACL.

Image pull errors – AWX images are pulled from quay.io. If your server has restricted outbound access, you may need to configure a proxy or mirror.

Wrapping Up

You now have Ansible AWX running on a K3s cluster on Ubuntu 24.04 or Debian 13. The AWX Operator handles the heavy lifting of deploying and managing the AWX application stack, while Kustomize keeps your deployment configuration clean and version-controlled. From here, you can create inventories, add credentials, build job templates, and start automating your infrastructure through the AWX web interface.



**Josphat Mutai**

Founder of Computingforgeeks. Senior Systems Engineer with over a decade of hands-on experience building and managing production infrastructure. Core expertise in Linux/UNIX administration, cloud platforms (AWS, GCP, Azure), and Kubernetes ecosystems (EKS, GKE, OpenShift). Deep experience with virtualization (KVM, Proxmox, OpenStack), infrastructure as code (Terraform, Terragrunt, Ansible, Crossplane), GitOps workflows (ArgoCD, FluxCD), Nix/NixOS, and container orchestration at scale. Currently focused on integrating AI and LLMs into DevOps and platform engineering workflows. Every guide on this site is tested on real systems before publishing.

KEEP READING