Installing {PlatformName}

{PlatformNameShort} is a modular platform. You can deploy {ControllerName} with other automation platform components, such as {HubName} and {EDAcontroller}. For more information about the components provided with {PlatformNameShort}, see {PlatformName} components in the {PlatformName} Planning Guide.

There are several supported installation scenarios for {PlatformName}. To install {PlatformName}, you must edit the inventory file parameters to specify your installation scenario. You can use one of the following as a basis for your own inventory file: * Single {ControllerName} with external (installer managed) database * Single {HubName} with external (installer managed) database * Single controller, single automation hub, and single event-driven ansible controller node with external (installer managed) database

Editing the {PlatformName} installer inventory file

You can use the {PlatformName} installer inventory file to specify your installation scenario.

Procedure

1. Navigate to the installer:
   a. [RPM installed package]

   ```
   $ cd /opt/ansible-automation-platform/installer/
   ```

   b. [bundled installer]

   ```
   $ cd ansible-automation-platform-setup-bundle-<latest-version>
   ```

   c. [online installer]

   ```
   $ cd ansible-automation-platform-setup-<latest-version>
   ```

2. Open the `inventory` file with a text editor.
3. Edit `inventory` file parameters to specify your installation scenario. You can use one of the supported Installation scenario examples as the basis for your `inventory` file.

Additional resources

- For a comprehensive list of pre-defined variables used in Ansible installation inventory files, see Inventory file variables.

Inventory file examples based on installation scenarios

Red Hat supports several installation scenarios for {PlatformNameShort}. Review the following examples and select the example suitable for your preferred installation scenario.

Inventory file recommendations based on installation scenarios

Before selecting your installation method for {PlatformNameShort}, review the following recommendations. Familiarity with these recommendations will streamline the installation process.

- For {PlatformName} or {HubName}: Add an {HubName} host in the `[automationhub]` group.
- Internal databases `[database]` are not supported. See the Containerized {PlatformName} Installation Guide for further information on using the containerized installer for environments requiring a monolithc deployment.
- Do not install {ControllerName} and {HubName} on the same node for versions of {PlatformNameShort} in a production or customer environment. This can cause contention issues and heavy resource use.
- Provide a reachable IP address or fully qualified domain name (FQDN) for the `[automationhub]` and `[automationcontroller]` hosts to ensure users can sync and install content from {HubName} from a different node. Do not use 'localhost'.
- `admin` is the default user ID for the initial log in to Ansible Automation Platform and cannot be changed in the inventory file.
- Use of special characters for `pg_password` is limited. The `!`, `#`, `0` and `@` characters are supported. Use of other special characters can cause the setup to fail.
- Enter your Red Hat Registry Service Account credentials in `registry_username` and `registry_password` to link to the Red Hat container registry.
- The inventory file variables `registry_username` and `registry_password` are only required if a non-bundle installer is used.

Single {ControllerName} with external (installer managed) database

Use this example to populate the inventory file to install {PlatformName}. This installation inventory file includes a single {ControllerName} node with an external database on a separate node.

```
[automationcontroller]
controller.example.com

[database]
data.example.com
```

```
[all:vars]
admin_password='<password>'
pg_host='data.example.com'
pg_port=5432
pg_database='awx'
pg_username='awx'
pg_password='<password>'
pg_sslmode='prefer'  # set to 'verify-full' for client-side enforced SSL

registry_url='registry.redhat.io'
registry_username='<registry username>'
registry_password='<registry password>'

# SSL-related variables
# If set, this will install a custom CA certificate to the system trust store.
# custom_ca_cert=/path/to/ca.crt
# Certificate and key to install in nginx for the web UI and API
# web_server_ssl_cert=/path/to/tower.cert
# web_server_ssl_key=/path/to/tower.key
# Server-side SSL settings for PostgreSQL (when we are installing it).
# postgres_use_ssl=False
# postgres_ssl_cert=/path/to/pgsql.crt
# postgres_ssl_key=/path/to/pgsql.key
```

Single {ControllerName and single {HubName} with external (installer managed) database

Use this example to populate the inventory file to deploy single instances of {ControllerName} and {HubName} with an external (installer managed) database.

```
[automationcontroller]
controller.example.com

[automationhub]
automationhub.example.com

[database]
data.example.com

[all:vars]
admin_password='<password>'
pg_host='data.example.com'
pg_port='5432'
pg_database='awx'
pg_username='awx'
pg_password='<password>'
pg_sslmode='prefer'  # set to 'verify-full' for client-side enforced SSL
```

```
registry_url='registry.redhat.io'
registry_username='<registry username>'
registry_password='<registry password>'

automationhub_admin_password= <PASSWORD>

automationhub_pg_host='data.example.com'
automationhub_pg_port=5432

automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password=<PASSWORD>
automationhub_pg_sslmode='prefer'

# The default install will deploy a TLS enabled Automation Hub.
# If for some reason this is not the behavior wanted one can
# disable TLS enabled deployment.
#
# automationhub_disable_https = False
# The default install will generate self-signed certificates for the Automation
# Hub service. If you are providing valid certificate via automationhub_ssl_cert
# and automationhub_ssl_key, one should toggle that value to True.
#
# automationhub_ssl_validate_certs = False
# SSL-related variables
# If set, this will install a custom CA certificate to the system trust store.
# custom_ca_cert=/path/to/ca.crt
# Certificate and key to install in Automation Hub node
# automationhub_ssl_cert=/path/to/automationhub.cert
# automationhub_ssl_key=/path/to/automationhub.key

# Certificate and key to install in nginx for the web UI and API
# web_server_ssl_cert=/path/to/tower.cert
# web_server_ssl_key=/path/to/tower.key
# Server-side SSL settings for PostgreSQL (when we are installing it).
# postgres_use_ssl=False
# postgres_ssl_cert=/path/to/pgsql.crt
# postgres_ssl_key=/path/to/pgsql.key
```

Connecting {HubName} to a {RHSSO} environment

You can configure the inventory file further to connect {HubName} to a {RHSSO} installation.

You must configure a different set of variables when connecting to a {RHSSO} installation managed by {PlatformNameShort} than when connecting to an external {RHSSO} installation.

For more information about these inventory variables, refer to the {HubNameMain} variables.

High availability {HubName}

Use the following examples to populate the inventory file to install a highly available {HubName}. This inventory file includes a highly available {HubName} with a clustered setup.

You can configure your HA deployment further to implement {RHSSO} and enable a high availability deployment of {HubName} on SELinux.

Specify database host IP

- Specify the IP address for your database host, using the `automation_pg_host` and `automation_pg_port` inventory variables. For example:

```
automationhub_pg_host='192.0.2.10'
automationhub_pg_port=5432
```

- Also specify the IP address for your database host in the [database] section, using the value in the `automationhub_pg_host` inventory variable:

```
[database]
192.0.2.10
```

List all instances in a clustered setup

- If installing a clustered setup, replace `localhost ansible_connection=local` in the [automationhub] section with the hostname or IP of all instances. For example:

```
[automationhub]
automationhub1.testing.ansible.com ansible_user=cloud-user ansible_host=192.0.2.18
automationhub2.testing.ansible.com ansible_user=cloud-user ansible_host=192.0.2.20
automationhub3.testing.ansible.com ansible_user=cloud-user ansible_host=192.0.2.22
```

Next steps

Check that the following directives are present in `/etc/pulp/settings.py` in each of the {PrivateHubName} servers:

```
USE_X_FORWARDED_PORT = True
USE_X_FORWARDED_HOST = True
```

NOTE	If `automationhub_main_url` is not specified, the first node in the [automationhub] group will be used as default.

Enabling a high availability (HA) deployment of {HubName} on SELinux

You can configure the inventory file to enable high availability deployment of {HubName} on SELinux. You must create two mount points for `/var/lib/pulp` and `/var/lib/pulp/pulpcore_static`, and then assign the appropriate SELinux contexts to each.

NOTE	You must add the context for `/var/lib/pulp` pulpcore_static and run the {PlatformNameShort} installer before adding the context for `/var/lib/pulp`.

Prerequisites

- You have already configured a NFS export on your server.

Procedure

1. Create a mount point at `/var/lib/pulp`:

```
$ mkdir /var/lib/pulp/
```

2. Open `/etc/fstab` using a text editor, then add the following values:

```
srv_rhel8:/data /var/lib/pulp nfs defaults,_netdev,nosharecache 0 0
srv_rhel8:/data/pulpcore_static /var/lib/pulp/pulpcore_static nfs
defaults,_netdev,nosharecache,context="system_u:object_r:httpd_sys_content_rw_t:s0"
0 0
```

3. Run the reload systemd manager configuration command:

```
$ systemctl daemon-reload
```

4. Run the mount command for `/var/lib/pulp`:

```
$ mount /var/lib/pulp
```

5. Create a mount point at `/var/lib/pulp/pulpcore_static`:

```
$ mkdir /var/lib/pulp/pulpcore_static
```

6. Run the mount command:

```
$ mount -a
```

7. With the mount points set up, run the {PlatformNameShort} installer:

```
$ setup.sh -- -b --become-user root
```

8. After the installation is complete, unmount the `/var/lib/pulp/` mount point.

Next steps

1. Apply the appropriate SELinux context.
2. Configure the pulpcore.serivce.

Additional Resources

- See the SELinux Requirements on the Pulp Project documentation for a list of SELinux contexts.
- See the Filesystem Layout for a full description of Pulp folders.

Configuring pulpcore.service

After you have configured the inventory file, and applied the SELinux context, you now need to configure the pulp service.

Procedure

1. With the two mount points set up, shut down the Pulp service to configure `pulpcore.service`:

```
$ systemctl stop pulpcore.service
```

2. Edit `pulpcore.service` using `systemctl`:

```
$ systemctl edit pulpcore.service
```

3. Add the following entry to `pulpcore.service` to ensure that {HubName} services starts only after starting the network and mounting the remote mount points:

```
[Unit]
After=network.target var-lib-pulp.mount
```

4. Enable `remote-fs.target`:

```
$ systemctl enable remote-fs.target
```

5. Reboot the system:

```
$ systemctl reboot
```

Troubleshooting

A bug in the pulpcore SELinux policies can cause the token authentication public/private keys in `etc/pulp/certs/` to not have the proper SELinux labels, causing the pulp process to fail. When this occurs, run the following command to temporarily attach the proper labels:

```
$ chcon system_u:object_r:pulpcore_etc_t:s0
/etc/pulp/certs/token_{private,public}_key.pem
```

Repeat this command to reattach the proper SELinux labels whenever you relabel your system.

Applying the SELinux context

After you have configured the inventory file, you must now apply the context to enable the high availability (HA) deployment of automation hub on SELinux.

Procedure

1. Shut down the Pulp service:

```
$ systemctl stop pulpcore.service
```

2. Unmount `/var/lib/pulp/pulpcore_static`:

```
$ umount /var/lib/pulp/pulpcore_static
```

3. Unmount `/var/lib/pulp/`:

```
$ umount /var/lib/pulp/
```

4. Open `/etc/fstab` using a text editor, then replace the existing value for `/var/lib/pulp` with the following:

```
srv_rhel8:/data /var/lib/pulp nfs
defaults,_netdev,nosharecache,context="system_u:object_r:pulpcore_var_lib_t:s0" 0 0
```

5. Run the mount command:

```
$ mount -a
```

Configuring content signing on private automation hub

To successfully sign and publish {CertifiedName}, you must configure {PrivateHubName} for signing.

Prerequisites

- Your GnuPG key pairs have been securely set up and managed by your organization.
- Your public-private key pair has proper access for configuring content signing on {PrivateHubName}.

Procedure

1. Create a signing script that accepts only a filename.

NOTE	This script acts as the signing service and must generate an ascii-armored detached `gpg` signature for that file using the key specified through the `PULP_SIGNING_KEY_FINGERPRINT` environment variable.

The script prints out a JSON structure with the following format.

```
{"file": "filename", "signature": "filename.asc"}
```

All the file names are relative paths inside the current working directory. The file name must remain the same for the detached signature.

Example:

The following script produces signatures for content:

```
#!/usr/bin/env bash

FILE_PATH=$1
SIGNATURE_PATH="$1.asc"

ADMIN_ID="$PULP_SIGNING_KEY_FINGERPRINT"
PASSWORD="password"

# Create a detached signature
gpg --quiet --batch --pinentry-mode loopback --yes --passphrase \
   $PASSWORD --homedir ~/.gnupg/ --detach-sign --default-key $ADMIN_ID \
   --armor --output $SIGNATURE_PATH $FILE_PATH

# Check the exit status
STATUS=$?
if [ $STATUS -eq 0 ]; then
   echo {\"file\": \"$FILE_PATH\", \"signature\": \"$SIGNATURE_PATH\"}
else
   exit $STATUS
fi
```

After you deploy a {PrivateHubName} with signing enabled to your {PlatformNameShort} cluster, new UI additions are displayed in collections.

2. Review the {PlatformNameShort} installer inventory file for options that begin with `automationhub_*`.

```
[all:vars]
.
.
.
automationhub_create_default_collection_signing_service = True
automationhub_auto_sign_collections = True
```

```
automationhub_require_content_approval = True
automationhub_collection_signing_service_key =
/abs/path/to/galaxy_signing_service.gpg
automationhub_collection_signing_service_script =
/abs/path/to/collection_signing.sh
```

The two new keys (**automationhub_auto_sign_collections** and **automationhub_require_content_approval**) indicate that the collections must be signed and approved after they are uploaded to {PrivateHubName}.

LDAP configuration on {PrivateHubName}

You must set the following six variables in your {PlatformName} installer inventory file to configure your {PrivateHubName} for LDAP authentication:

- `automationhub_authentication_backend`
- `automationhub_ldap_server_uri`
- `automationhub_ldap_bind_dn`
- `automationhub_ldap_bind_password`
- `automationhub_ldap_user_search_base_dn`
- `automationhub_ldap_group_search_base_dn`

If any of these variables are missing, the Ansible Automation installer cannot complete the installation.

Setting up your inventory file variables

When you configure your {PrivateHubName} with LDAP authentication, you must set the proper variables in your inventory files during the installation process.

Procedure

1. Access your inventory file according to the procedure in Editing the {PlatformName} installer inventory file.
2. Use the following example as a guide to set up your {PlatformNameShort} inventory file:

```
automationhub_authentication_backend = "ldap"

automationhub_ldap_server_uri = "ldap://ldap:389"   (for LDAPs use
automationhub_ldap_server_uri = "ldaps://ldap-server-fqdn")
automationhub_ldap_bind_dn = "cn=admin,dc=ansible,dc=com"
automationhub_ldap_bind_password = "GoodNewsEveryone"
automationhub_ldap_user_search_base_dn = "ou=people,dc=ansible,dc=com"
automationhub_ldap_group_search_base_dn = "ou=people,dc=ansible,dc=com"
```

NOTE The following variables will be set with default values, unless you set them with other options.

```
auth_ldap_user_search_scope= `SUBTREE'
auth_ldap_user_search_filter= `(uid=%(user)s)`
auth_ldap_group_search_scope= 'SUBTREE'
auth_ldap_group_search_filter= '(objectClass=Group)`
auth_ldap_group_type_class=
'django_auth_ldap.config:GroupOfNamesType'
```

3. Optional: Set up extra parameters in your {PrivateHubName} such as user groups, super user access, or mirroring. Go to Configuring extra LDAP parameters to complete this optional step.

Configuring extra LDAP parameters

If you plan to set up super user access, user groups, mirroring or other extra parameters, you can create a yaml file that comprises them in your `ldap_extra_settings` dictionary.

Procedure

1. Create a yaml file that contains `ldap_extra_settings`.
   - Example:

     ```
     #ldapextras.yml
     ---
     ldap_extra_settings:
       AUTH_LDAP_USER_ATTR_MAP: '{"first_name": "givenName", "last_name": "sn",
     "email": "mail"}'
     ...
     ```

     Then run `setup.sh -e @ldapextras.yml` during {PrivateHubName} installation.

2. Use this example to set up a superuser flag based on membership in an LDAP group.

   ```
   #ldapextras.yml
   ---
   ldap_extra_settings:
     AUTH_LDAP_USER_FLAGS_BY_GROUP: {"is_superuser": "cn=pah-
   admins,ou=groups,dc=example,dc=com",}
   ...
   ```

   Then run `setup.sh -e @ldapextras.yml` during {PrivateHubName} installation.

3. Use this example to set up superuser access.

   ```
   #ldapextras.yml
   ---
   ldap_extra_settings:
     AUTH_LDAP_USER_FLAGS_BY_GROUP: {"is_superuser": "cn=pah-
   admins,ou=groups,dc=example,dc=com",}
   ```

```
...
```

Then run `setup.sh -e @ldapextras.yml` during {PrivateHubName} installation.

4. Use this example to mirror all LDAP groups you belong to.

```
#ldapextras.yml
---
ldap_extra_settings:
  AUTH_LDAP_MIRROR_GROUPS: True
...
```

Then run `setup.sh -e @ldapextras.yml` during {PrivateHubName} installation.

5. Use this example to map LDAP user attributes (such as first name, last name, and email address of the user).

```
#ldapextras.yml
---
ldap_extra_settings:
  AUTH_LDAP_USER_ATTR_MAP: {"first_name": "givenName", "last_name": "sn", "email":
"mail",}
...
```

Then run `setup.sh -e @ldapextras.yml` during {PrivateHubName} installation.

6. Use the following examples to grant or deny access based on LDAP group membership:

   a. To grant {PrivateHubName} access (for example, members of the `cn=pah-nosoupforyou,ou=groups,dc=example,dc=com` group):

```
#ldapextras.yml
---
ldap_extra_settings:
  AUTH_LDAP_DENY_GROUP: 'cn=pah-nosoupforyou,ou=groups,dc=example,dc=com'
...
```

   b. To deny {PrivateHubName} access (for example, members of the `cn=pah-nosoupforyou,ou=groups,dc=example,dc=com` group):

```
#ldapextras.yml
---
ldap_extra_settings:
  AUTH_LDAP_DENY_GROUP: 'cn=pah-nosoupforyou,ou=groups,dc=example,dc=com'
...
```

c. Then run `setup.sh -e @ldapextras.yml` during {PrivateHubName} installation.

7. Use this example to enable LDAP debug logging.

```
#ldapextras.yml
---
ldap_extra_settings:
  GALAXY_LDAP_LOGGING: True
...
```

Then run `setup.sh -e @ldapextras.yml` during {PrivateHubName} installation.

NOTE	If it is not practical to re-run `setup.sh` or if debug logging is enabled for a short time, you can add a line containing `GALAXY_LDAP_LOGGING: True` manually to the `/etc/pulp/settings.py` file on {PrivateHubName}. Restart both `pulpcore-api.service` and `nginx.service` for the changes to take effect. To avoid failures due to human error, use this method only when necessary.

8. Use this example to configure LDAP caching by setting the variable `AUTH_LDAP_CACHE_TIMEOUT`.

```
#ldapextras.yml
---
ldap_extra_settings:
  AUTH_LDAP_CACHE_TIMEOUT: 3600
...
```

Then run `setup.sh -e @ldapextras.yml` during {PrivateHubName} installation.

Verification

To verify you have set up correctly, confirm you can view all of your settings in the `/etc/pulp/settings.py` file on your {PrivateHubName}.

LDAP referrals

If your LDAP servers return referrals, you might have to disable referrals to successfully authenticate using LDAP on {PrivateHubName}.

If not, the following message is returned:

```
Operation unavailable without authentication
```

To disable the LDAP REFERRALS lookup, set:

```
GALAXY_LDAP_DISABLE_REFERRALS = true
```

This sets `AUTH_LDAP_CONNECTIONS_OPTIONS` to the correct option.

Single controller, single automation hub, and single event-driven ansible controller node with external (installer managed) database

Use this example to populate the inventory file to deploy single instances of {ControllerName} and {HubName} with an external (installer managed) database.

IMPORTANT	This scenario requires a minimum of {ControllerName} 2.4 for successful deployment of {EDAcontroller}.
IMPORTANT	{EDAcontroller} must be installed on a separate server and cannot be installed on the same host as {HubName} and {ControllerName}.

```
[automationcontroller]
controller.example.com

[automationhub]
automationhub.example.com

[automationedacontroller]
automationedacontroller.example.com ansible_connection=local

[database]
data.example.com

[all:vars]
admin_password='<password>'
pg_host='data.example.com'
pg_port='5432'
pg_database='awx'
pg_username='awx'
pg_password='<password>'
pg_sslmode='prefer'  # set to 'verify-full' for client-side enforced SSL

registry_url='registry.redhat.io'
registry_username='<registry username>'
registry_password='<registry password>'

# Automation hub configuration

automationhub_admin_password= <PASSWORD>

automationhub_pg_host='data.example.com'
automationhub_pg_port=5432

automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password=<PASSWORD>
automationhub_pg_sslmode='prefer'
```

```
# Automation EDA controller configuration

automationedacontroller_admin_password='<eda-password>'

automationedacontroller_pg_host='data.example.com'
automationedacontroller_pg_port=5432

automationedacontroller_pg_database='automationedacontroller'
automationedacontroller_pg_username='automationedacontroller'
automationedacontroller_pg_password='<password>'

# Keystore file to install in SSO node
# sso_custom_keystore_file='/path/to/sso.jks'

# This install will deploy SSO with sso_use_https=True
# Keystore password is required for https enabled SSO
sso_keystore_password=''

# This install will deploy a TLS enabled Automation Hub.
# If for some reason this is not the behavior wanted one can
# disable TLS enabled deployment.
#
# automationhub_disable_https = False
# The default install will generate self-signed certificates for the Automation
# Hub service. If you are providing valid certificate via automationhub_ssl_cert
# and automationhub_ssl_key, one should toggle that value to True.
#
# automationhub_ssl_validate_certs = False
# SSL-related variables
# If set, this will install a custom CA certificate to the system trust store.
# custom_ca_cert=/path/to/ca.crt
# Certificate and key to install in Automation Hub node
# automationhub_ssl_cert=/path/to/automationhub.cert
# automationhub_ssl_key=/path/to/automationhub.key

# Certificate and key to install in nginx for the web UI and API
# web_server_ssl_cert=/path/to/tower.cert
# web_server_ssl_key=/path/to/tower.key
# Server-side SSL settings for PostgreSQL (when we are installing it).
# postgres_use_ssl=False
# postgres_ssl_cert=/path/to/pgsql.crt
# postgres_ssl_key=/path/to/pgsql.key

# Boolean flag used to verify Automation Controller's
# web certificates when making calls from Automation EDA Controller.
#
# automationedacontroller_controller_verify_ssl = true

# Certificate and key to install in Automation Hub node
# automationhub_ssl_cert=/path/to/automationhub.cert
```

```
# automationhub_ssl_key=/path/to/automationhub.key
```

Running the {PlatformName} installer setup script

After you update the inventory file with required parameters for installing your {PrivateHubName}, run the installer setup script.

Procedure

- Run the `setup.sh` script

```
$ sudo ./setup.sh
```

Installation of {PlatformName} will begin.

Verifying installation of {ControllerName}

Verify that you installed automation controller successfully by logging in with the admin credentials you inserted in the `inventory` file.

Prerequisite

- Port 443 is available

Procedure

1. Navigate to the IP address specified for the {ControllerName} node in the `inventory` file.
2. Log in with the user ID `admin` and the password credentials you set in the `inventory` file.

NOTE	The {ControllerName} server is accessible from port 80 (**https://<CONTROLLER_SERVER_NAME>/**) but redirects to port 443.

IMPORTANT	If the installation fails and you are a customer who has purchased a valid license for {PlatformName}, contact Ansible through the Red Hat Customer portal.

Upon a successful log in to {ControllerName}, your installation of {PlatformName} {PlatformVers} is complete.

Additional {ControllerName} configuration and resources

See the following resources to explore additional {ControllerName} configurations.

Table 1. Resources to configure {ControllerName}

Resource link	Description
Automation Controller Quick Setup Guide	Set up {ControllerName} and run your first playbook.
Automation Controller Administration Guide	Configure {ControllerName} administration through customer scripts, management jobs, etc.
Configuring proxy support for {PlatformName}	Set up {ControllerName} with a proxy server.
Managing usability analytics and data collection from {ControllerName}	Manage what {ControllerName} information you share with Red Hat.
Automation Controller User Guide	Review {ControllerName} functionality in more detail.

Verifying installation of {HubName}

Verify that you installed your {HubName} successfully by logging in with the admin credentials you inserted into the `inventory` file.

Procedure

1. Navigate to the IP address specified for the {HubName} node in the `inventory` file.
2. Log in with the user ID `admin` and the password credentials you set in the `inventory` file.

IMPORTANT	If the installation fails and you are a customer who has purchased a valid license for {PlatformName}, contact Ansible through the Red Hat Customer portal.

Upon a successful login to {HubName}, your installation of {PlatformName} {PlatformVers} is complete.

Additional {HubName} configuration and resources

See the following resources to explore additional {HubName} configurations.

Table 2. Resources to configure {ControllerName}

Resource link	Description
Managing user access in {PrivateHubName}	Configure user access for {HubName}.
Managing Red Hat Certified, validated, and Ansible Galaxy content in {HubName}	Add content to your {HubName}.
Publishing proprietary content collections in {HubName}	Publish internally developed collections on your {HubName}.

Verifying {EDAcontroller} installation

Verify that you installed {EDAcontroller} successfully by logging in with the admin credentials you

inserted in the inventory file.

Procedure

1. Navigate to the IP address specified for the {EDAcontroller} node in the `inventory` file.
2. Log in with the user ID `admin` and the password credentials you set in the `inventory` file.

IMPORTANT	If the installation fails and you are a customer who has purchased a valid license for {PlatformName}, contact Ansible through the Red Hat Customer portal.

Upon a successful login to {EDAcontroller}, your installation of {PlatformName} {PlatformVers} is complete.