Hardening and compliance

This guide provides recommended practices for various processes needed to install, configure, and maintain Ansible Automation Platform on Red Hat Enterprise Linux in a secure manner.

Providing feedback on Red Hat documentation

If you have a suggestion to improve this documentation, or find an error, you can contact technical support at https://access.redhat.com to open a request.

1. Introduction to hardening Ansible Automation Platform

This document provides guidance for improving the security posture (referred to as “hardening” throughout this guide) of your Red Hat Ansible Automation Platform deployment on Red Hat Enterprise Linux.

The following are not currently within the scope of this guide:

- Other deployment targets for Ansible Automation Platform, such as OpenShift.
- Ansible Automation Platform managed services available through cloud service provider marketplaces.

NOTE

Hardening and compliance for Ansible Automation Platform 2.4 includes additional considerations with regards to the specific *Defense Security Information Agency* (DISA) *Security Technical Implementation Guides* (STIGs) for automation controller, but this guidance does not apply to Ansible Automation Platform 2.5.

This guide takes a practical approach to hardening the Ansible Automation Platform security posture, starting with the planning and architecture phase of deployment and then covering specific guidance for installation, initial configuration, and day 2 operations. As this guide specifically covers Ansible Automation Platform running on Red Hat Enterprise Linux, hardening guidance for Red Hat Enterprise Linux will be covered where it affects the automation platform components. Additional considerations with regards to the DISA STIGs for Red Hat Enterprise Linux are provided for those organizations that integrate the DISA STIGs as a part of their overall security strategy.

NOTE

These recommendations do not guarantee security or compliance of your deployment of Ansible Automation Platform. You must assess security from the unique requirements of your organization to address specific threats and risks and balance these against implementation factors.

1.1. Audience

This guide is written for personnel responsible for installing, configuring, and maintaining Ansible Automation Platform 2.5 when deployed on Red Hat Enterprise Linux. Additional information is provided for security operations, compliance assessment, and other functions associated with related security processes.

1.2. Overview of Ansible Automation Platform

Ansible is an open source, command-line IT automation software application written in Python. You can use Ansible Automation Platform to configure systems, deploy software, and orchestrate advanced workflows to support application deployment, system updates, and more. Ansible's main strengths are simplicity and ease of use. It also has a strong focus on security and reliability, featuring minimal moving parts. It uses secure, well-known communication protocols like SSH, HTTPS, and WinRM for transport and uses a human-readable language that is designed for getting started quickly without extensive training.

Ansible Automation Platform enhances the Ansible language with enterprise-class features, such as *Role-Based Access Controls* (RBAC), centralized logging and auditing, credential management, job scheduling, and complex automation workflows. With Ansible Automation Platform you get certified content from our robust partner ecosystem; added security; reporting, and analytics, as well as life cycle technical support to scale automation across your organization. Ansible Automation Platform simplifies the development and operation of automation workloads for managing enterprise application infrastructure life cycles. It works across multiple IT domains including operations, networking, security, and development, as well as across diverse hybrid environments.

1.2.1. Red Hat Ansible Automation Platform deployment methods

There are three different installation methods for Ansible Automation Platform:

- RPM-based on Red Hat Enterprise Linux
- Container-based on Red Hat Enterprise Linux
- Operator-based on Red Hat OpenShift Container Platform

This document offers guidance on hardening Ansible Automation Platform when installed using either of the first two installation methods (RPM-based or container-based). This document further recommends using the container-based installation method for new deployments, as the RPM-based installation program will be deprecated in a future release.

For further information, see Deprecated features.

Operator-based deployments are not described in this document.

1.2.2. Ansible Automation Platform components

Ansible Automation Platform is a modular platform composed of separate components that can be connected together, including automation controller, platform gateway, automation hub, and

Event-Driven Ansible controller.

Additional resources

For more information about the components provided within Ansible Automation Platform, see Red Hat Ansible Automation Platform components in *Planning your installation*.

2. Hardening Ansible Automation Platform

This guide takes a practical approach to hardening the Ansible Automation Platform security posture, starting with the planning and architecture phase of deployment and then covering specific guidance for the installation phase. As this guide specifically covers Ansible Automation Platform running on Red Hat Enterprise Linux, hardening guidance for Red Hat Enterprise Linux is covered where it affects the automation platform components.

2.1. Planning considerations

Red Hat Ansible Automation Platform is composed of the following primary components:

- Automation controller
- Automation mesh
- Private automation hub
- Event-Driven Ansible controller

A PostgreSQL database is also provided, although a user-provided PostgreSQL database can be used as well. Red Hat recommends that customers always deploy all components of Ansible Automation Platform so that all features and capabilities are available for use without the need to take further action.

For further information, see Red Hat Ansible Automation Platform Architecture

2.1.1. Ansible Automation Platform deployment topologies

Install Ansible Automation Platform 2.5 based on one of the documented tested deployment reference architectures defined in Tested deployment models. Enterprise organizations should use one of the enterprise reference architectures for production deployments to ensure the highest level of uptime, performance, and continued scalability. Organizations or deployments that are resource constrained can use a "growth" reference architecture. Review the Tested deployment models document to determine the reference architecture that best suits your requirements. The reference architecture chosen will include planning information such as an architecture diagram, the number of Red Hat Enterprise Linux servers required, the network ports and protocols used by the deployment, and load balancer information for enterprise architectures.

It is possible to install the Ansible Automation Platform on different infrastructure reference architectures and with different environment configurations. Red Hat does not fully test architectures outside of published deployment models. The following diagram is a tested container enterprise architecture:

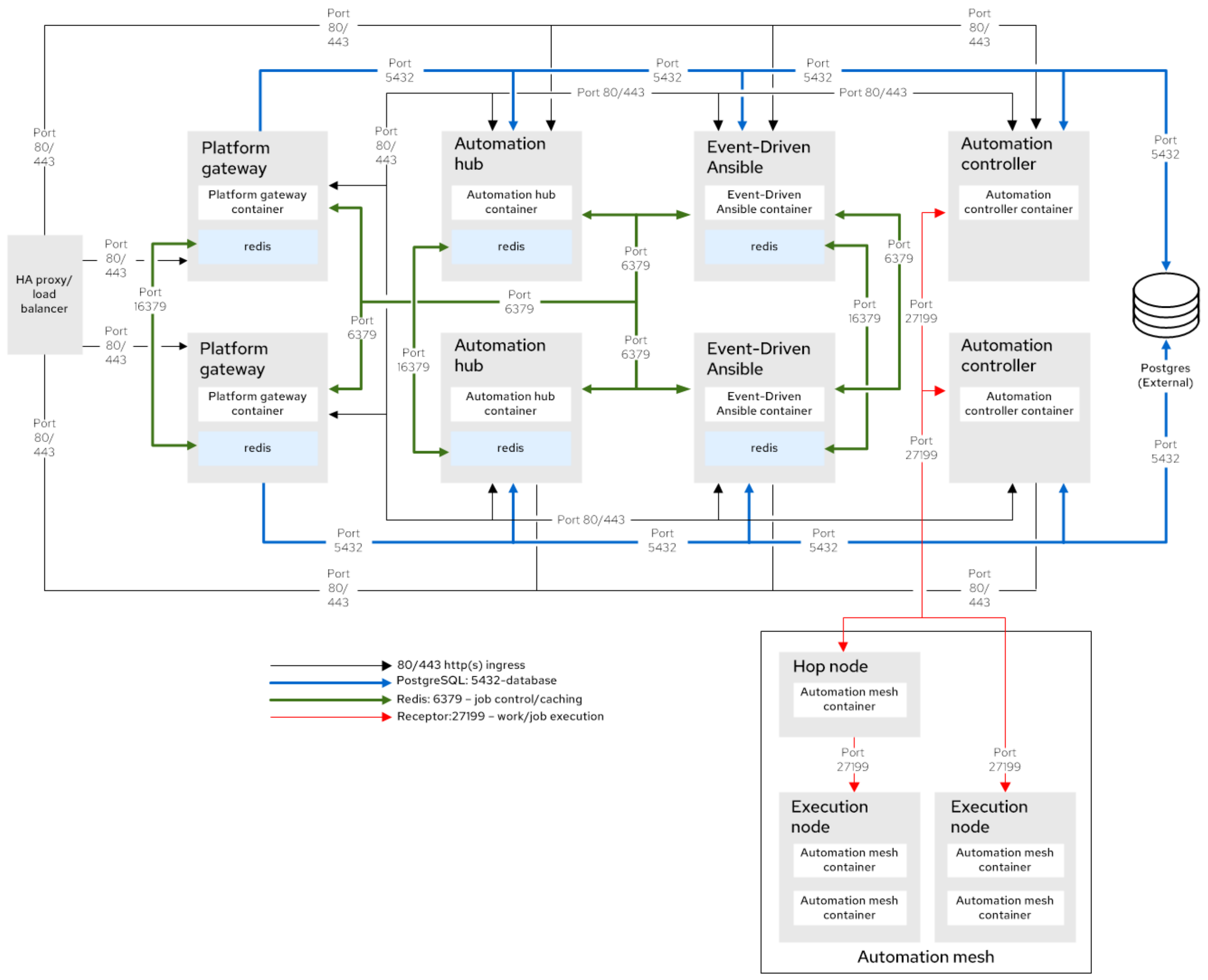

Figure 1. Reference architecture overview

When planning firewall or cloud network security group configurations related to Ansible Automation Platform, see the "Network Ports" section of your chosen topology in Tested deployment models to understand what network ports need to be opened on a firewall or security group.

For internet-connected systems, the Networks and Protocols section of Planning your installation defines the outgoing traffic requirements for services that Ansible Automation Platform can be configured to use, such as Red Hat automation hub, Insights for Ansible Automation Platform, Ansible Galaxy, the registry.redhat.io container image registry, and so on.

Restrict access to the ports used by the Ansible Automation Platform components to protected networks and clients. The following restrictions are highly recommended:

- Restrict the PostgreSQL database port (5432) on the database servers so that only the other Ansible Automation Platform component servers (automation controller, automation hub, Event-Driven Ansible controller) are permitted access.
- Restrict SSH access to the Ansible Automation Platform servers from the installation host and other trusted systems used for maintenance access to the Ansible Automation Platform servers.
- Restrict HTTPS access to the automation controller, automation hub, and Event-Driven Ansible controller from trusted networks and clients.

2.1.2. DNS, NTP, and service planning

DNS

When installing Ansible Automation Platform, the installation program script checks that certain infrastructure servers are defined with a *Fully Qualified Domain Name* (FQDN) in the installer inventory. This guide recommends that all Ansible Automation Platform infrastructure nodes have a valid FQDN defined in DNS which resolves to a routable IP address, and that these FQDNs be used in the installer inventory file.

DNS and load balancing

When using a load balancer with Ansible Automation Platform as described in the deployment topology, an additional FQDN is needed for the load-balancer. For example, an FQDN such as `aap.example.com` might be used for the load balancer which in turn directs traffic to each of the platform gateway components defined in the installation inventory.

NTP

Configure each server in the Ansible Automation Platform infrastructure to synchronize time with an *Network Time Protocol* (NTP) pool or your organization's NTP service. This ensures that logging and auditing events generated by Ansible Automation Platform have an accurate time stamp, and that any scheduled jobs running from the automation controller execute at the correct time. This also enables communication between the systems within Ansible Automation Platform to not reject messages based on timeouts.

For information on configuring the chrony service for NTP synchronization, see Using Chrony in the Red Hat Enterprise Linux documentation.

2.1.3. Credential management planning for Ansible Automation Platform

Red Hat Ansible Automation Platform uses credentials to authenticate requests to jobs against machines, synchronize with inventory sources, and import project content from a version control system.

Automation controller manages three sets of secrets:

- User passwords for **local Ansible Automation Platform users**.
- Secrets for Ansible Automation Platform **operational use** (database password, message bus password, and so on).
- Secrets for **automation use** (SSH keys, cloud credentials, external password vault credentials, and so on).

Implementing a privileged access or credential management solution to protect credentials from compromise is a highly recommended practice. Organizations should audit the use of, and provide additional programmatic control over, access and privilege escalation.

You can further secure automation credentials by ensuring they are unique and stored only in automation controller. Services such as OpenSSH can be configured to allow credentials on connections only from specific addresses. Use different credentials for automation from those used by system administrators to log in to a server. Although direct access should be limited where possible, it can be used for disaster recovery or other ad-hoc management purposes, allowing for easier auditing.

Different automation jobs might need to access a system at different levels. For example, you can have low-level system automation that applies patches and performs security baseline checking, while a higher-level piece of automation deploys applications. By using different keys or credentials for each piece of automation, the effect of any one key vulnerability is minimized. This also allows for easy baseline auditing.

Ansible Automation Platform operational secrets

Ansible Automation Platform uses several secrets (passwords, keys, and so on) operationally. These secrets are stored unencrypted on the various Ansible Automation Platform servers, as each component service must read them at startup. All files are protected by Unix permissions, and restricted to the root user or the appropriate service account user. These files should be routinely monitored to ensure there has been no unauthorized access or modification.

RPM-based installation secrets

The following table provides the location of these secrets for RPM-based installs of Ansible Automation Platform.

Table 1. Ansible Automation Platform operational secrets

Automation controller secrets	
File path	**Description**
`/etc/tower/SECRET_KEY`	A secret key used for encrypting automation secrets in the database. If the `SECRET_KEY` changes or is unknown, no encrypted fields in the database will be accessible.

`/etc/tower/tower.cert`
 	
`/etc/tower/tower.key`
	SSL certificate and key for the automation controller web service.
	
	A self-signed `cert/key` is installed by default; you can provide a locally appropriate certificate and key.
	
	For more information, see Installing with user-provided PKI certificates.

`/etc/tower/conf.d/postgres.py`	Contains the password used by automation controller to connect to the database, unless TLS authentication is used for the database connection.
`/etc/tower/conf.d/channels.py`	Contains the secret used by automation controller for websocket broadcasts.

`/etc/tower/conf.d/gateway.py`	Contains the key used by automation controller to sync state with the platform gateway.
Platform gateway secrets	
File path	**Description**
`/etc/ansible-automation-platform/gateway/SECRET_KEY`	A secret key used for encrypting automation secrets in the database. If the `SECRET_KEY changes` or is unknown, the platform gateway cannot access the encrypted secrets in the database.
`/etc/ansible-automation-platform/gateway/gateway.cert`	SSL certificate for the platform gateway web service. A self-signed certificate is installed by default, although a user-provided certificate and key pair can be used. For more information, see Installing with user-provided PKI certificates.
`/etc/ansible-automation-platform/gateway/gateway.key`	SSL key for the platform gateway web service. A self-signed certificate is installed by default, although a user-provided certificate and key pair can be used. For more information, see Installing with user-provided PKI certificates.
`/etc/ansible-automation-platform/gateway/cache.cert`	SSL certificate used for mutual TLS (mTLS) authentication with the Redis cache used by the platform gateway.
`/etc/ansible-automation-platform/gateway/cache.key`	SSL key used for mutual TLS (mTLS) authentication with the Redis cache used by the platform gateway.
`/etc/ansible-automation-platform/gateway/settings.py`	Contains the password used by the platform gateway to connect to the database, unless TLS authentication is used for the database connection. Also contains the password used to connect to the Redis cache used by the platform gateway.
Automation hub secrets	
File path	**Description**

`/etc/pulp/settings.py`	Contains the password used by automation hub to connect to the database, unless TLS authentication is used for the database connection. Contains the Django secret key used by the automation hub web service.
`/etc/pulp/certs/token_public_key.pem`	OpenSSL public key in PEM format for the automation hub EE token authentication. It is generated by default from the `token_private_key.pem` file.
`/etc/pulp/certs/token_private_key.pem`	OpenSSL private key in PEM format for the automation hub EE token authentication. It is generated by default, although a user can provide their own private key with the ``pulp_token_auth_key` `` installation inventory variable.
`/etc/pulp/certs/pulp_webserver.crt`	SSL certificate for the automation hub web service. A self-signed certificate is installed by default, although a user-provided certificate and key pair can be used. For more information, see Installing with user-provided PKI certificates.
`/etc/pulp/certs/pulp_webserver.key`	SSL key for the automation hub web service. A self-signed certificate is installed by default, although a user-provided certificate and key pair can be used. For more information, see Installing with user-provided PKI certificates.
`/etc/pulp/certs/database_fields.symmetric.key`	A key used for encrypting sensitive fields in the automation hub database table. If the key changes or is unknown, automation hub cannot access the encrypted fields in the database.
Event-Driven Ansible secrets	
File path	**Description**

`/etc/ansible-automation-platform/eda/SECRET_KEY`	A secret key used for encrypting fields in the Event-Driven Ansible controller database table. If the SECRET_KEY changes or is unknown, the Event-Driven Ansible controller cannot access the encrypted fields in the database.
`/etc/ansible-automation-platform/eda/settings.yaml`	Contains the password used by the Event-Driven Ansible gateway to connect to the database, unless TLS authentication is used for the database connection. Contains the password used to connect to the Redis cache used by the Event-Driven Ansible controller. Contains the key used by the Event-Driven Ansible controller to sync state with the platform gateway.
`/etc/ansible-automation-platform/eda/server.cert`	SSL certificate for the Event-Driven Ansible controller web service. A self-signed certificate is installed by default, although a user-provided certificate and key pair can be used. For more information, see Installing with user-provided PKI certificates.
`/etc/ansible-automation-platform/eda/server.key`	SSL key for the Event-Driven Ansible controller web service. A self-signed certificate is installed by default, although a user-provided certificate and key pair can be used. For more information, see Installing with user-provided PKI certificates.
`/etc/ansible-automation-platform/eda/cache.cert`	SSL certificate used for mutual TLS (mTLS) authentication with the Redis cache used by the Event-Driven Ansible controller
`/etc/ansible-automation-platform/eda/cache.key`	SSL key used for mutual TLS (mTLS) authentication with the Redis cache used by the Event-Driven Ansible controller

`/etc/ansible-automation-platform/eda/websocket.cert`	SSL certificate for the Event-Driven Ansible controller websocket endpoint. A self-signed certificate is installed by default, although a user-provided certificate and key pair can be used. For more information, see Installing with user-provided PKI certificates.
`/etc/ansible-automation-platform/eda/websocket.key`	SSL key for the Event-Driven Ansible controller websocket endpoint. A self-signed certificate is installed by default, although a user-provided certificate and key pair can be used. For more information, see Installing with user-provided PKI certificates.
Redis secrets	
File path	**Description**
`/etc/ansible-automation-platform/ca/ansible-automation-platform-managed-ca-cert.crt`	SSL certificate for the internal self-signed certificate authority used by the installation program to generate the default self-signed certificates for each component service.
`/etc/ansible-automation-platform/ca/ansible-automation-platform-managed-ca-cert.key`	SSL key for the internal self-signed certificate authority used by the installer to generate the default self-signed certificates for each component service.

NOTE Some of these file locations reflect the previous product name of automation controller, formerly named Ansible Tower.

Container-based installation secrets

The secrets listed for RPM-based installations are also used in container-based installations, but they are stored in a different manner. Container-based installations of Red Hat Ansible Automation Platform use Podman secrets to store operational secrets. These secrets can be listed using the `podman secret list` command.

By default, Podman stores data in the home directory of the user who installed and runs the containerized Red Hat Ansible Automation Platform services. Podman secrets are stored in the file `$HOME/.local/share/containers/storage/secrets/filedriver/secretsdata.json` as base64-encoded strings, so while they are not in plain text the values are only obfuscated.

The data stored in a Podman secret can be shown using the command `podman secret inspect --showsecret <secret>`.

This file should be routinely monitored to ensure there has been no unauthorized access or modification.

Automation use secrets

Automation controller stores a variety of secrets in the database that are either used for automation or are a result of automation. Automation use secrets include:

- All secret fields of all credential types (passwords, secret keys, authentication tokens, secret cloud credentials).
- Secret tokens and passwords for external services defined in automation controller settings.
- “password” type survey field entries.

You can grant users and teams the ability to use these credentials without actually exposing the credential to the user. This means that if a user moves to a different team or leaves the organization, you don’t have to re-key all of your systems.

Automation controller uses SSH (or the Windows equivalent) to connect to remote hosts. To pass the key from automation controller to SSH, the key must be decrypted before it can be written to a named pipe. Automation controller then uses that pipe to send the key to SSH (so that it is never written to disk). If passwords are used, automation controller handles those by responding directly to the password prompt and decrypting the password before writing it to the prompt.

As an administrator with superuser access, you can define a custom credential type in a standard format using a YAML/JSON-like definition, enabling the assignment of new credential types to jobs and inventory updates. This enables you to define a custom credential type that works in ways similar to existing credential types. For example, you can create a custom credential type that injects an API token for a third-party web service into an environment variable, which your playbook or custom inventory script can consume.

To encrypt secret fields, Ansible Automation Platform uses the *Advanced Encryption Standard* (AES) in *Cipher Block Chaining* (CBC) mode with a 256-bit key for encryption, *Public-Key cryptography Standard* (PKCS7) padding, and *Hash-Based Message Authentication Code* (HMAC) using SHA256 for authentication. The encryption/decryption process derives the AES-256 bit encryption key from the `SECRET_KEY`, the field name of the model field, and the database-assigned auto-incremented record ID. Thus, if any attribute used in the key generation process changes, Ansible Automation Platform fails to correctly decrypt the secret. Ansible Automation Platform is designed such that the `SECRET_KEY` is never readable in playbooks Ansible Automation Platform launches. This means that these secrets are never readable by Ansible Automation Platform users, and no secret field values are ever made available through the Ansible Automation Platform REST API. If a secret value is used in a playbook, you must use `no_log` on the task so that it is not accidentally logged. For more information, see Protecting sensitive data with no log.

Protecting sensitive data with no_log

If you save Ansible output to a log, you expose any secret data in your Ansible output, such as passwords and usernames. To keep sensitive values out of your logs, mark tasks that expose them with the `no_log: True` attribute. However, the `no_log` attribute does not affect debugging output, so be careful not to debug playbooks in a production environment.

2.1.4. User authentication planning

There are two types of user accounts to consider in an Ansible Automation Platform environment:

- Infrastructure accounts: user accounts on the RHEL servers that run the Ansible Automation Platform services.
- Application accounts: user accounts for the Ansible Automation Platform web UI and API.

Infrastructure server account planning

For user accounts on the RHEL servers that run Ansible Automation Platform services, follow your organizational policies to determine if individual user accounts should be local or should use an external authentication source. Only users who have a valid need to perform maintenance tasks on the Ansible Automation Platform components themselves should be granted access to the underlying RHEL servers, as the servers store configuration files that contain sensitive information, such as encryption keys and service passwords. Because these individuals must have privileged access to maintain Ansible Automation Platform services, minimizing access to the underlying RHEL servers is critical. Do not grant sudo access to the root account or local Ansible Automation Platform service accounts (awx, pulp, postgres) to untrusted users.

NOTE

Some local service accounts are created and managed by the RPM-based installation program. These particular accounts on the underlying RHEL hosts cannot come from an external authentication source.

Ansible Automation Platform account planning

Ansible Automation Platform user accounts for accessing the user interface or API can either be local (stored in the Ansible Automation Platform database) or mapped to an external authentication source, such as a *Lightweight Directory Access Protocol* (LDAP) server. This guide recommends that where possible, all primary user accounts should be mapped to an external authentication source. Using external account sources eliminates a source of error when working with permissions in this context and minimizes the amount of time devoted to maintaining a full set of users exclusively within Ansible Automation Platform. This includes accounts assigned to individual persons as well as for non-person entities, such as service accounts used for external application integration. Reserve any local accounts, such as the default “admin” account, for emergency access or “break glass” scenarios where the external authentication mechanism isn’t available.

- LDAP
- SAML
- TACACS+
- Radius
- Azure Active Directory
- Google OAuth
- Generic OIDC
- Keycloak

- GitHub
- GitHub Organization
- GitHub team
- GitHub enterprise
- GitHub enterprise organization
- GitHub enterprise team

Choose an authentication mechanism that adheres to your organization's authentication policies and refer to Access management and authentication to understand the prerequisites for the relevant authentication mechanism. The authentication mechanism used must ensure that the authentication-related traffic between Ansible Automation Platform and the authentication back-end is encrypted when the traffic occurs on a public or non-secure network (for example, LDAPS or LDAP over TLS, HTTPS for OAuth2 and SAML providers, etc.).

In the Ansible Automation Platform UI, any "system administrator" account can edit, change, and update any inventory or automation definition. Restrict these account privileges to the minimum set of users possible for low-level automation controller configuration and disaster recovery.

NOTE

Ansible Automation Platform 2.5 introduces a new central authentication mechanism used by all of the platform components:

- Automation controller
- Private automation hub
- Event-Driven Ansible controller

Before 2.5, each of these components had their own authentication configuration.

2.1.5. Logging and log capture

Visibility and analytics is an important pillar of Enterprise Security and Zero Trust Architecture. Logging is key to capturing actions and auditing. You can manage logging and auditing by using the built-in audit support described in the Auditing the system section of the Security hardening for Red Hat Enterprise Linux guide. Ansible Automation Platform's built-in logging and activity stream logs all change within Red Hat Ansible Automation Platform and automation logs for auditing purposes. More detailed information is available in the Logging and Aggregation section of Configuring automation execution.

This guide recommends that you configure Ansible Automation Platform and the underlying Red Hat Enterprise Linux systems to collect logging and auditing centrally, rather than reviewing it on the local system. Ansible Automation Platform must be configured to use external logging to compile log records from multiple components within the Ansible Automation Platform server. The events occurring must be time-correlated to conduct accurate forensic analysis. This means that the Ansible Automation Platform server must be configured with an NTP server that is also used by the logging aggregator service, as well as the targets of Ansible Automation Platform. The correlation must meet certain industry tolerance requirements. In other words, there might be a varying requirement that time stamps of different logged events must not differ by any amount greater than *x* seconds. This capability should be available in the external logging service.

Another critical capability of logging is the ability to use cryptography to protect the integrity of log tools. Log data includes all information (for example, log records, log settings, and log reports) needed to successfully log information system activity. It is common for attackers to replace the log tools or inject code into the existing tools to hide or erase system activity from the logs. To address this risk, log tools must be cryptographically signed so that you can identify when the log tools have been modified, manipulated, or replaced. For example, one way to validate that the log tool(s) have not been modified, manipulated or replaced is to use a checksum hash against the tool file(s). This ensures the integrity of the tool(s) has not been compromised.

2.1.6. Auditing and incident detection

Ansible Automation Platform should be used to fulfill security policy requirements by applying the NIST Cybersecurity Framework for common use cases, such as:

- Requiring HTTPS for web servers on Red Hat Enterprise Linux.
- Requiring TLS encryption for internal communication between web servers and database servers on Red Hat Enterprise Linux.
- Generating reports showing that the policy is properly deployed.
- Monitoring for drift that violates the policy.
- Automating correction of any policy violation.

This can be done through 5 steps of the cybersecurity framework:

IDENTIFY
Define the requirements to be implemented according to the security policy.

PROTECT
Implement and apply the requirements as an Ansible Playbook.

DETECT
Monitor for drift and generate an audit report.

RESPOND
Explore actions that could be taken when an incident is detected.

RECOVER
Use Ansible to restore the systems to the known good configuration.

2.1.7. Red Hat Enterprise Linux host planning

The security of Ansible Automation Platform relies in part on the configuration of the underlying Red Hat Enterprise Linux servers. For this reason, the underlying Red Hat Enterprise Linux hosts for each Ansible Automation Platform component must be installed and configured in accordance with the Security hardening for Red Hat Enterprise Linux 8 or Security hardening for Red Hat Enterprise Linux 9 (depending on which operating system is used), as well as any security profile requirements (*Center for Internet Security* (CIS), STIG, *Health Insurance Portability and Accountability Act* (HIPAA), and so on) used by your organization. This document recommends Red Hat Enterprise Linux 9 for all new deployments. When using the container-based installation method, Red Hat Enterprise Linux 9 is required.

Ansible Automation Platform and additional software

When installing the Ansible Automation Platform components on Red Hat Enterprise Linux servers, the Red Hat Enterprise Linux servers should be dedicated to that use alone. Additional server capabilities must not be installed in addition to Ansible Automation Platform, as this is an unsupported configuration and might affect the security and performance of the Ansible Automation Platform software.

Similarly, when Ansible Automation Platform is deployed on a Red Hat Enterprise Linux host, it installs software like the nginx web server, the Pulp software repository, and the PostgreSQL database server (unless a user-provided external database is used). This software should not be modified or used in a more generic fashion (for example, do not use nginx to serve additional website content or PostgreSQL to host additional databases) as this is an unsupported configuration and might affect the security and performance of Ansible Automation Platform. The configuration of this software is managed by the Ansible Automation Platform installation program, and any manual changes might be undone when performing upgrades.

2.2. Installation

There are installation-time decisions that affect the security posture of Ansible Automation Platform. The installation process includes setting a number of variables, some of which are relevant to the hardening of the Ansible Automation Platform infrastructure. Before installing Ansible Automation Platform, consider the guidance in the installation section of this guide.

2.2.1. Install from a dedicated installation host

The Ansible Automation Platform installation program can be run from one of the infrastructure servers, such as an automation controller, or from an external system that has SSH access to the Ansible Automation Platform infrastructure servers. The Ansible Automation Platform installation program is also used not just for installation, but for subsequent day-two operations, such as backup and restore, as well as upgrades. This guide recommends performing installation and day-two operations from a dedicated external server, hereafter referred to as the installation host. Doing so eliminates the need to log in to one of the infrastructure servers to run these functions. The installation host must only be used for management of Ansible Automation Platform and must not run any other services or software.

The installation host must be a Red Hat Enterprise Linux server that has been installed and configured in accordance with Security hardening for Red Hat Enterprise Linux and any security profile requirements relevant to your organization (CIS, STIG, and so on). Obtain the Ansible Automation Platform installer as described in the Planning your installation, and create the installer inventory file as described in the Editing the Red Hat Ansible Automation Platform installer inventory file. This inventory file is used for upgrades, adding infrastructure components, and day-two operations by the installation program, so preserve the file after installation for future operational use.

Access to the installation host must be restricted only to those personnel who are responsible for managing the Ansible Automation Platform infrastructure. Over time, it will contain sensitive information, such as the installation inventory (which contains the initial login credentials for Ansible Automation Platform), copies of user-provided PKI keys and certificates, backup files, and so on. The installation host must also be used for logging in to the Ansible Automation Platform infrastructure servers through SSH when necessary for infrastructure management and maintenance.

2.2.2. Security-relevant variables in the installation inventory

The installation inventory file defines the architecture of the Ansible Automation Platform infrastructure and provides a number of variables that can be used to modify the initial configuration of the infrastructure components. For more information on the installer inventory, see About the installer inventory file.

The following table lists a number of security-relevant variables and their recommended values for an RPM-based deployment.

Table 2. Security-relevant inventory variables

RPM deployment variable	Recommended Value	Details
`postgres_use_ssl`	true	The installation program configures the installer-managed Postgres database to accept SSL-based connections when this variable is set. The default for this variable is false which means SSL/TLS is not used for PostgreSQL connections. When set to true, the platform connects to PostgreSQL by using SSL/TLS.

RPM deployment variable	Recommended Value	Details
`pg_sslmode` `automation_gateway_pg_sslmode` `automationhub_pg_sslmode` `automationcontroller_pg_sslmode`	verify-full	These variables control mutual TLS (mTLS) authentication to the database. By default, when each service connects to the database, it tries an encrypted connection, but it is not enforced. Setting this variable to `verify-full` enforces an mTLS negotiation between the service and the database. The `postgres_use_ssl` variable must also be set to `true` for this pg_sslmode to be effective. **NOTE**: If a third-party database is used instead of the installer-managed database, the third-party database must be set up independently to accept mTLS connections.
`nginx_disable_hsts` `automation_gateway_disable_hsts` `automationhub_disable_hsts` `automationcontroller_disable_hsts`	false	If set to `true`, these variables disable HTTPS *strict transport Security* (HSTS) connections to each of the component web services. The default is `false`. If these variables are absent from the installer inventory it is effectively equivalent to defining the variables as `false`.

The following table lists a number of security-relevant variables and their recommended values for a container-based deployment.

Table 3. Security-relevant containerized inventory variables

Container deployment variable	Recommended Value	Details
`postgresql_disable_tls`	false	If set to `true`, this variable disables TLS connections to the installer-managed PostgreSQL database. The default is `false` If this variable is absent from the installer inventory, it is effectively equivalent to defining the variable as `false`.
`controller_pg_sslmode` `gateway_pg_sslmode` `hub_pg_sslmode` `eda_pg_sslmode`	verify-full	These variables control mutual TLS (mTLS) authentication to the database. By default, when each service connects to the database, it tries an encrypted connection, but it is not enforced. Setting this variable to `verify-full` enforces an mTLS negotiation between the service and the database. **NOTE** If a third-party database is used instead of the installer-managed database, the third-party database must be set up independently to accept mTLS connections.

Container deployment variable	Recommended Value	Details
`controller_nginx_disable_https` `gateway_nginx_disable_https` `hub_nginx_disable_https` `da_nginx_disable_https`	`false`	If set to `true`, these variables disable HTTPS connections to each of the component web services. The default is `false`. If these variables are absent from the installer inventory, it is effectively equivalent to defining the variables as `false`.
`controller_nginx_disable_hsts` `gateway_nginx_disable_hsts` `hub_nginx_disable_hsts` `eda_nginx_disable_hsts`	`false`	If set to 'true', these variables disable HTTPS Strict Transport Security (HSTS) connections to each of the component web services. The default is `false`. If these variables are absent from the installer inventory it is effectively equivalent to defining the variables as `false`.

In an enterprise architecture where a load balancer is used in front of multiple platform gateways, SSL client connections can be terminated at the load balancer or passed through to the individual AAP servers. If SSL is being terminated at the load balancer, this guide recommends that the traffic gets re-encrypted from the load balancer to the individual Ansible Automation Platform servers. This ensures that end-to-end encryption is in use. In this scenario, the `*_disable_https` variables listed are set to the default value of `false`.

2.2.3. Installing with user-provided PKI certificates

By default, Ansible Automation Platform creates self-signed *Public Key Infrastructure* (PKI) certificates for the infrastructure components of the platform. Where an existing PKI infrastructure is available, certificates must be generated for the automation controller, private automation hub, Event-Driven Ansible controller, and the postgres database server. Copy the certificate files and their relevant key files to the installation program directory, along with the CA certificate used to verify the certificates.

Use the following inventory variables to configure the infrastructure components with the new certificates.

Table 4. PKI certificate inventory variables

RPM Variable	Containerized Variable	Details

custom_ca_cert	custom_ca_cert	The path to the custom CA certificate file. If set, this will install a custom CA certificate to the system truststore.
web_server_ssl_cert	controller_tls_cert	The file name of the automation controller PKI certificate located in the installer directory.
web_server_ssl_key	controller_tls_key	The file name of the automation controller PKI key located in the installation program directory.
automationhub_ssl_cert	hub_tls_cert	The file name of the private automation hub PKI certificate located in the installation program directory.
automationhub_ssl_key	hub_tls_key	The file name of the private automation hub PKI key located in the installation program directory.
postgres_ssl_cert	postgresql_tls_cert	The file name of the database server PKI certificate located in the installation program directory. This variable is only needed for the installer-managed database server, not if a third-party database is used.
postgres_ssl_key	postgresql_tls_key	The file name of the database server PKI key located in the installation program directory. This variable is only needed for the installer-managed database server, not if a third-party database is used.
automationedacontroller_ssl_cert	eda_tls_cert	The file name of the Event-Driven Ansible controller PKI certificate located in the installation program directory.
automationedacontroller_ssl_key	eda_tls_key	The file name of the Event-Driven Ansible controller PKI key located in the installation program directory.

-	`gateway_tls_cert`	The filename of the platform gateway PKI certificate located in the installation program directory.
-	`gateway_tls_key`	The file name of the platform gateway PKI key located in the installation program directory.

When multiple platform gateways are deployed with a load balancer, `gateway_tls_cert` and `gateway_tls_key` are shared by each platform gateway. To prevent hostname mismatches, the certificate's *Common Name* (CN) must match the DNS FQDN used by the load balancer. If your organizational policies require unique certificates for each service, each certificate requires a *Subject Alt Name* (SAN) that matches the DNS FQDN used for the load-balanced service. To install unique certificates and keys on each platform gateway, the certificate and key variables in the installation inventory file must be defined as per-host variables instead of in the `[all:vars]` section. For example:

```
[automationgateway]
gateway0.example.com gateway_tls_cert=/path/to/cert0 gateway_tls_key=/path/to/key0
gateway1.example.com gateway_tls_cert=/path/to/cert1 gateway_tls_key=/path/to/key1

[automationcontroller]
controller0.example.com web_server_ssl_cert=/path/to/cert0
web_server_ssl_key=/path/to/key0
controller1.example.com web_server_ssl_cert=/path/to/cert1
web_server_ssl_key=/path/to/key1
controller2.example.com web_server_ssl_cert=/path/to/cert2
web_server_ssl_key=/path/to/key2

[automationhub]
hub0.example.com automationhub_ssl_cert=/path/to/cert0
automationhub_ssl_key=/path/to/key0
hub1.example.com automationhub_ssl_cert=/path/to/cert1
automationhub_ssl_key=/path/to/key1
hub2.example.com automationhub_ssl_cert=/path/to/cert2
automationhub_ssl_key=/path/to/key2
```

2.2.4. Sensitive variables in the installation inventory

The installation inventory file contains a number of sensitive variables, mainly those used to set the initial passwords used by Ansible Automation Platform, that are normally kept in plain text in the inventory file. To prevent unauthorized viewing of these variables, you can keep these variables in an encrypted Ansible vault. To do this, go to the installer directory and create a vault file:

- `cd /path/to/ansible-automation-platform-setup-bundle-2.5-1-x86_64`
- `ansible-vault create vault.yml`

You are prompted for a password to the new Ansible vault. Do not lose the vault password because it is required every time you need to access the vault file, including during day-two operations and performing backup procedures. You can secure the vault password by storing it in an encrypted password manager or in accordance with your organizational policy for storing passwords securely.

Add the sensitive variables to the vault, for example:

```
admin_password/controller_admin_password: <secure_controller_password>
pg_password/controller_pg_password: <secure_db_password>
automationhub_admin_password/hub_admin_password: <secure_hub_password>
automationhub_pg_password/hub_pg_password: <secure_hub_db_password>
automationedacontroller_admin_password/eda_admin_password: <secure_eda_password>
automationedacontroller_pg_password/eda_pg_password: <secure_eda_db_password>
-/gateway_admin_password: <secure_gateway_password>
-/gateway_pg_password:<secure_gateway_db_password>
```

Make sure these variables are not also present in the installation inventory file. To use the new Ansible vault with the installation program, run it with the command `./setup.sh -e @vault.yml—--ask-vault-pass`.

2.2.5. Compliance profile considerations

In many environments, Ansible Automation Platform might need to be installed on Red Hat Enterprise Linux systems where security controls have been applied to meet the requirements of a compliance profile such as CIS Critical Security Controls, *Payment Card Industry/Data Security Standard* (PCI/DSS), the DISA STIG, or a similar profile. In these environments, there are a specific set of security controls that might need to be modified for Ansible Automation Platform to run properly. Apply any compliance profile controls to the Red Hat Enterprise Linux servers being used for Ansible Automation Platform before installation, and then modify the following security controls as required.

In environments where these controls are required, discuss waiving the controls with your security auditor.

Fapolicyd

A compliance policy might require the `fapolicyd` daemon to be running. However, Ansible Automation Platform is not currently supported when `fapolicyd` is enforcing policy, as this causes failures during both installation and operation of Ansible Automation Platform. Because of this, the installation program runs a pre-flight check that halts installation if it discovers that `fapolicyd` is enforcing policy. This guide recommends setting `fapolicyd` to permissive mode on Ansible Automation Platform using the following steps:

1. Edit the file `/etc/fapolicyd/fapolicyd.conf` and set "permissive = 1".
2. Restart the service with the command

   `sudo systemctl restart fapolicyd.service`

NOTE

> If this security control is also applied to the installation host, the default `fapolicyd` configuration causes the Ansible Automation Platform installation program to fail. In this case, the recommendation is to set `fapolicyd` to permissive mode on the installation host.

File systems mounted with "noexec"

A compliance profile might require that certain file systems are mounted with the `noexec` option to prevent execution of binaries located in these file systems. The Ansible Automation Platform RPM-based installation program runs a preflight check that fails if any of the following file systems are mounted with the `noexec` option:

- `/tmp`
- `/var`
- `/var/tmp`

To install Ansible Automation Platform, you must re-mount these file systems with the `noexec` option removed. When installation is complete, proceed with the following steps:

1. Reapply the `noexec` option to the `/tmp` and `/var/tmp` file systems.
2. Change the automation controller job execution path from `/tmp` to an alternate directory that does not have the `noexec` option enabled.
3. To make this change, log in to the automation controller UI as an administrator, navigate to Settings and select **Jobs settings**.
4. Change the "Job execution path" setting to the alternate directory.

During normal operations, the file system which contains the `/var/lib/awx` subdirectory (typically `/var`) must not be mounted with the `noexec` option, or the automation controller cannot run automation jobs in execution environments.

In environments where STIG controls are routinely audited, discuss waiving the STIG controls related to file system `noexec` with your security auditor.

User namespaces

A compliance profile might require that the kernel setting `user.max_user_namespaces` is set to "0", to prevent the launch of Linux containers. The DISA STIG, for example, specifically requires this control but only if Linux containers are not required. Because Ansible Automation Platform can be installed and operated in containers and also uses containers as part of its execution environment capability, Linux containers are required and this control must be disabled.

To check the `user.max_user_namespaces` kernel setting, complete the following steps on each Ansible Automation Platform component in the installation inventory:

1. Log in to your automation controller at the command line.
2. Run the command `sudo sysctl user.max_user_namespaces`.
3. If the output indicates that the value is zero, look at the contents of the file `/etc/sysctl.conf` and

all files under `/etc/sysctl.d/`, edit the file containing the `user.max_user_namespaces` setting, and set the value to "65535".

4. To apply this new value, run the command `sudo sysctl -p <file>`, where `<file>` is the file just modified.
5. Re-run the command `sudo sysctl user.max_user_namespaces` and verify that the value is now set to "65535".

Interactive session timeout

A compliance profile might require that an interactive session timeout be enforced. For example, the DISA STIG requires that all users be automatically logged out after 15 minutes of inactivity. The installation process often requires an hour or more to complete, and this control can stop the installation process and log out the user before installation is complete. The same also applies to day two operations such as backup and restore, which in production environments often take longer than the recommended interactive session timeout. During these operations, increase the interactive session timeout to ensure the operation is successful.

There are multiple ways in which this control can be enforced, including shell timeout variables, setting the idle session timeout for `systemd-logind`, or setting SSH connection timeouts, and different compliance profiles can use one or more of these methods. The one that most often interrupts the installation and day two operations is the idle session timeout for `systemd-logind`, which was introduced in the DISA STIG version V2R1 (Red Hat Enterprise Linux 8) and V2R2 (Red Hat Enterprise Linux 9). To increase the idle session timeout for `systemd-logind`, as the root user:

- Edit the file `/etc/systemd/logind.conf`.
- If the `StopIdleSessionSec` setting is set to zero, no change is needed.
- If the `StopIdleSessionSec` setting is non-zero, this indicates that the session will be terminated after that number of seconds.

  Change `StopIdleSessionSec=7200` to increase the timeout, then run `systemctl restart systemd-logind` to apply the change.

- Log out of the interactive session entirely and log back in to ensure the new setting applies to the current login session.

NOTE This change only needs to be made on the installation host, or if an installation host is not used, the host where the Ansible Automation Platform installation program is run.

Sudo and NOPASSWD

A compliance profile might require that all users with sudo privileges must provide a password (that is, the `NOPASSWD` directive must not be used in a sudoers file). The Ansible Automation Platform installation program runs many tasks as a privileged user, and by default expects to be able to elevate privileges without a password. To provide a password to the installer for elevating privileges, append the following options when launching the RPM installer script:

`./setup.sh <setup options> --ask-become-pass`.

For the container-based installation program:

```
ansible-playbook ansible.containerized_installer.install --ask-become-pass
```

When the installation program is run, you are prompted for the user’s password to elevate privileges.

NOTE Using the `--ask-become-pass` option also applies when running the installer for day-two operations such as backup and restore.

2.3. Initial configuration

Granting access to certain parts of the system exposes security vulnerabilities. Apply the following practices to help secure access:

- Minimize access to system administrative accounts. There is a difference between the user interface (web interface) and access to the operating system that the automation controller is running on. A system administrator or super user can access, edit, and disrupt any system application. Anyone with root access to automation controller has the potential ability to decrypt those credentials, and so minimizing access to system administrative accounts is crucial for maintaining a secure system.
- Minimize local system access. Automation controller should not require local user access except for administrative purposes. Non-administrator users should not have access to the automation controller system.
- Enforce separation of duties. Different components of automation might need to access a system at different levels. Use different keys or credentials for each component so that the effect of any one key or credential vulnerability is minimized.
- Restrict automation controller to the minimum set of users possible for low-level automation controller configuration and disaster recovery only. In an automation controller context, any automation controller ‘system administrator’ or ‘superuser’ account can edit, change, and update any inventory or automation definition in automation controller.

2.3.1. Use a configuration as code paradigm

The Red Hat Community of Practice has created a set of automation content available through collections to manage Ansible Automation Platform infrastructure and configuration as code. This enables automation of the platform itself through *Configuration as Code* (CaC). While many of the benefits of this approach are clear, there are security implications to consider.

The following Ansible content collections are available for managing Ansible Automation Platform components using an infrastructure as code methodology, all of which are found on the Ansible Automation Hub:

Table 5. Ansible content collections

Validated Collection	Collection Purpose

`infra.aap_utilities`	Ansible content for automating day 1 and day 2 operations of Ansible Automation Platform, including installation, backup and restore, certificate management, and more.
`infra.aap_configuration`	A collection of roles to manage the creation of Ansible Automation Platform components, including users and groups (RBAC), projects, job templates and workflows, credentials, and more. This collection includes functionality from the older `infra.controller_configuration`, `infra.ah_configuration` and `infra.eda_configuration` and should be used in their place with Ansible Automation Platform 2.5.
`infra.ee_utilities`	A collection of roles for creating and managing execution environment images, or migrating from the older Tower virtualenvs to execution environments.

Many organizations use CI/CD platforms to configure pipelines or other methods to manage this type of infrastructure. However, using Ansible Automation Platform natively, a webhook can be configured to link a Git-based repository natively. In this way, Ansible can respond to Git events to trigger Job Templates directly. This removes the need for external CI components from this overall process and thus reduces the attack surface.

These practices enable version control of all infrastructure and configuration. Apply Git best practices to ensure proper code quality inspection before being synchronized into Ansible Automation Platform. Relevant Git best practices include the following:

- Creating pull requests.
- Ensuring that inspection tools are in place.
- Ensuring that no plain text secrets are committed.
- Ensuring that pre-commit hooks and any other policies are followed.

CaC also encourages using external vault systems which removes the need to store any sensitive data in the repository, or deal with having to individually vault files as needed. This is particularly important when storing Ansible Automation Platform configuration in a source code repository, as automation controller credentials and Event-Driven Ansible credentials must be provided to the collection variables in plain text which should not be committed to a source repository. For more information on using external vault systems, see the External credential vault considerations section in this guide.

Configure centralized logging

Centralized logging is essential to assist in monitoring system security and performing large-scale log analysis. The *Confidentiality, Integrity, and Availability* (CIA) triad, which originated from a combination of ideas from the US military and government, is the model that is the foundation for proper security system development and best practices. Centralized logging falls under the Integrity aspect to assist in identifying if data or systems have been tampered with. The logging to a centralized system enables troubleshooting automation across multiple systems by collecting all logs in the single location therefore making it easier to identify issues, analyze trends and correlate events from different servers, especially in a complex Ansible Automation Platform deployment. Otherwise, manually checking individual machines would be time consuming so this capability is valuable with debugging in addition to meeting security best practices. This ensures the overall system health, stability and assists in identifying potential security threats. In addition to the logging configuration, the failure to log due to storage capacity, hardware failure as well as high availability architecture should be taken into consideration.

There are several additional benefits including:

- The data is sent in JSON format over a HTTP connection using minimal service-specific tweaks engineered in a custom handler or through an imported library. The types of data that are most useful to the controller are job fact data, job events/job runs, activity stream data, and log messages.
- Deeper insights into the automation process by analyzing logs from different parts of the infrastructure, including playbook execution details, task outcomes, and system events.
- Identifying performance bottlenecks and optimizing the Ansible playbooks by analyzing execution times and resource usage from the logs.
- Centralized logging helps meet compliance mandates by providing a single source of truth for auditing purposes.
- Third Party integration with a centralized log management platform like Splunk, Logstash, ElasticSearch, or Loggly to collect and analyze logs.

The logging aggregator service works with the following monitoring and data analysis systems:

- Splunk
- Loggly
- Sumologic
- Elastic stack (formerly ELK stack)

To set up logging to any of the aggregator types for centralized logging follow these steps:

Procedure

1. From the navigation panel, select **Settings › Logging**.
2. On the **Logging settings** page, click **[Edit]**.
3. You can configure the following options:
   - **Logging Aggregator**: Enter the hostname or IP address that you want to send logs to.
   - **Logging Aggregator Port**: Specify the port for the aggregator if it requires one.

NOTE When the connection type is HTTPS, you can enter the hostname as a URL with a port number, after which, you are not required to enter the port again. However, TCP and UDP connections are determined by the hostname and port number combination, rather than URL. Therefore, in the case of a TCP or UDP connection, supply the port in the specified field. If a URL is entered in the **Logging Aggregator** field instead, its hostname portion is extracted as the hostname.

- **Logging Aggregator Type**: Click to select the aggregator service from the list:
- **Logging Aggregator Username**: Enter the username of the logging aggregator if required.
- **Logging Aggregator Password/Token**: Enter the password of the logging aggregator if required.
- **Loggers to Send Data to the Log Aggregator Form**: All four types of data are pre-populated by default. Click the tooltip ? icon next to the field for additional information on each data type. Delete the data types you do not want.
- **Cluster wide unique identifier**: Use this to uniquely identify instances.
- **Logging Aggregator Protocol**: Click to select a connection type (protocol) to communicate with the log aggregator. Subsequent options vary depending on the selected protocol.
- **TCP Connection Timeout**: Specify the connection timeout in seconds. This option is only applicable to HTTPS and TCP log aggregator protocols.
- **Logging Aggregator Level Threshold**: Select the level of severity you want the log handler to report.
- **Maximum number of messages that can be stored in the log action queue**:Defines how large the `rsyslog` action queue can grow in number of messages stored. This can have an impact on memory use. When the queue reaches 75% of this number, the queue starts writing to disk (`queue.highWatermark` in `rsyslog`). When it reaches 90%, `NOTICE`, `INFO`, and `DEBUG` messages start to be discarded (`queue.discardMark` with `queue.discardSeverity=5`).
- **Maximum disk persistence for rsyslogd action queuing (in GB)**: The amount of data to store (in gigabytes) if an `rsyslog` action takes time to process an incoming message (defaults to 1). Equivalent to the `rsyslogd queue.maxdiskspace` setting on the action (e.g. `omhttp`). It stores files in the directory specified by `LOG_AGGREGATOR_MAX_DISK_USAGE_PATH`.
- **File system location for rsyslogd disk persistence**: Location to persist logs that should be retried after an outage of the external log aggregator (defaults to `/var/lib/awx`). Equivalent to the `rsyslogd queue.spoolDirectory` setting.
- **Log Format For API 4XX Errors**: Configure a specific error message. For more information, see API 4XX Error Configuration Set the following options:
- **Log System Tracking Facts Individually**: Click the tooltip ? icon for additional information, such as whether or not you want to turn it on, or leave it off by default.

4. Review your entries for your chosen logging aggregation.
   - **Enable External Logging**: Select this checkbox if you want to send logs to an external log aggregator.
   - **Enable/disable HTTPS certificate verification**: Certificate verification is enabled by default for the HTTPS log protocol. Select this checkbox if you want the log handler to verify the HTTPS certificate sent by the external log aggregator before establishing a connection.

    - **Enable rsyslogd debugging**: Select this checkbox to enable high verbosity debugging for `rsyslogd`. Useful for debugging connection issues for external log aggregation.

5. Click **[Save]** or **[Cancel]** to abandon the changes.

The following steps enable the LDAP logging:

To enable logging for LDAP, use the following procedure.

Procedure

1. Edit the gateway settings file:

    a. On Ansible Automation Platform2.5 Containerized, the file is `~/aap/gateway/etc/settings.py` (as the user running the platform gateway container).

    b. On Ansible Automation Platform2.5 RPM-based, the file is `/etc/ansible-automation-platform/gateway/settings.py`.

```
 (...)
  CACHES['fallback']['LOCATION'] = '/var/cache/ansible-automation-
platform/gateway'

  LOGGING['loggers']['aap']['level'] = 'INFO'
  LOGGING['loggers']['ansible_base']['level'] = 'INFO'
  LOGGING['loggers']['django_auth_ldap']['level'] = 'DEBUG'      ######      add
this line

  (...)
```

2. Restart the platform gateway service or container:

    a. On Ansible Automation Platform2.5 Containerized, restart the platform gateway service so that it restarts the platform gateway container:

> **NOTE**
>
> Ensure that you run `systemctl with the `--user`` flag as follows:
>
> + `$ systemctl --user restart automation-gateway`

    b. On Ansible Automation Platform2.5 RPM-based, use the `automation-gateway-service` command:

```
# automation-gateway-service restart
```

Some of the following examples of meeting compliance requirements come from the US DoD *Security Technical Implementation Guide*, but go back to integrity and security best practices.

Automation controller must use external log providers that can collect user activity logs in independent, protected repositories to prevent modification or repudiation. Automation controller must be configured to use external logging to compile log records from multiple components within the server. The events occurring must be time-correlated in order to conduct accurate forensic analysis. In addition, the correlation must meet certain tolerance criteria.

The following steps implement the security control:

Procedure

1. Log in to automation controller as an administrator.
2. From the navigation panel, select **Settings › Logging**.
3. On the **Logging settings** page, click **[Edit]**.
4. Set the following fields:
   - Set **Logging Aggregator** to `Not configured`. This is the default.
   - Set **Enable External Logging** to `On`.
   - Set **Logging Aggregator Level Threshold** to DEBUG.
   - Set **TCP Connection Timeout** to 5 (the default) or to the organizational timeout.
   - Set **Enable/disable HTTPS certificate verification** to `On`.
5. Click **[Save]**.

Automation controller must allocate log record storage capacity and shut down by default upon log failure (unless availability is an overriding concern). It is critical that when a system is at risk of failing to process logs, it detects and takes action to mitigate the failure. Log processing failures include software/hardware errors, failures in the log capturing mechanisms, and log storage capacity being reached or exceeded. During a failure, the application server must be configured to shut down unless the application server is part of a high availability system. When availability is an overriding concern, other approved actions in response to a log failure are as follows:

1. If the failure was caused by the lack of log record storage capacity, the application must continue generating log records if possible (automatically restarting the log service if necessary), overwriting the oldest log records in a first-in-first-out manner.
2. If log records are sent to a centralized collection server and communication with this server is lost or the server fails, the application must queue log records locally until communication is restored or until the log records are retrieved manually. Upon restoration of the connection to the centralized collection server, action must be taken to synchronize the local log data with the collection server.

   The following steps implement the security control:

   a. Open a web browser and navigate to the logging API, `/api/v2/settings/logging/`

      Ensure that you are authenticated as an automation controller administrator.

   b. In the **Content** section, modify the following values:
      - `LOG_AGGREGATOR_ACTION_MAX_DISK_USAGE_GB` = organization-defined requirement for log buffering.
      - `LOG_AGGREGATOR_MAX_DISK_USAGE_PATH` = `/var/lib/awx`` ..Click **[PUT]**.

Automation controller must generate the appropriate log records. Automation controller's web server must log all details related to user sessions in support of troubleshooting, debugging, and forensic analysis. Without a data logging feature, the organization loses an important auditing and analysis tool for event investigations.

To implement the security control as a System Administrator, for each automation controller host:

Procedure

1. From the navigation panel, select **Settings › System**. The System Settings page is displayed.
2. Click **[Edit]**.
3. Set the following:
    - **Enable Activity Stream** = On
    - **Enable Activity Stream for Inventory Sync** = On
    - **Organization Admins Can Manage Users and Teams** = On
    - **All Users Visible to Organization Admins** = On
4. Click **[Save]**.

Automation controller’s log files must be accessible by explicitly defined privilege. A failure of the confidentiality of automation controller log files would enable an attacker to identify key information about the system that they might not otherwise be able to obtain that would enable them to enumerate more information to enable escalation or lateral movement.

To implement the security control.

Procedure

1. As a system administrator for each automation controller host, set the permissions and owner of the automation controller NGINX log directory:
    - `chmod 770 /var/log/nginx
    - `chown nginx:root /var/log/nginx`
2. Set the permissions and owner of the automation controller log directory:
    - `chmod 770 /var/log/tower`
    - `chown awx:awx /var/log/tower`
3. Set the permissions and owner of the automation controller supervisor log directory:
    - `chmod 770 /var/log/supervisor/`
    - `chown root:root /var/log/supervisor/`

Automation controller must be configured to fail over to another system in the event of log subsystem failure. Automation controller hosts must be capable of failing over to another automation controller host which can handle application and logging functions upon detection of an application log processing failure. This enables continual operation of the application and logging functions while minimizing the loss of operation for the users and loss of log data.

To implement the security control.

- If automation controller is not in an HA configuration, the administrator must reinstall automation controller.

2.3.2. External credential vault considerations

Secrets management is an essential component of maintaining a secure automation platform. We recommend the following secrets management practices:

- Ensure that there are no unauthorized users with access to the system, and ensure that only users who require access are granted it. Automation controller encrypts sensitive information such as passwords and API tokens, but also stores the key to decryption. Authorized users potentially have access to everything.
- Use an external system to manage secrets. In cases where credentials need to be updated, an external system can retrieve updated credentials with less complexity than an internal system. External systems for managing secrets include CyberArk, HashiCorp Vault, Microsoft Azure Key Management, and others. For more information, see the Secret Management System section of Using automation execution.

2.4. Day two operations

Day 2 Operations include Cluster Health and Scaling Checks, including Host, Project, and environment level Sustainment. You must continually analyze configuration and security drift.

2.4.1. RBAC considerations

As an administrator, you can use the *Role-Based Access Controls* (RBAC) built into the platform gateway to delegate access to server inventories, organizations, and more. Administrators can also centralize the management of various credentials, enabling end users to use a needed secret without ever exposing that secret to the end user. RBAC controls allow Ansible Automation Platform to help you increase security and streamline management.

RBAC is the practice of granting roles to users or teams. RBAC is easiest to think of in terms of Roles which define precisely who or what can see, change, or delete an “object” for which a specific capability is being set.

There are a few main concepts that you should become familiar with regarding Ansible Automation Platform’s RBAC design–roles, resources, and users. Users can be members of a role, which gives them certain access to any resources associated with that role, or any resources associated with “descendant” roles.

A role is essentially a collection of capabilities. Users are granted access to these capabilities and the controller’s resources through the roles to which they are assigned or through roles inherited through the role hierarchy.

Roles associate a group of capabilities with a group of users. All capabilities are derived from membership within a role. Users receive capabilities only through the roles to which they are assigned or through roles they inherit through the role hierarchy. All members of a role have all capabilities granted to that role. Within an organization, roles are relatively stable, while users and capabilities are both numerous and may change rapidly. Users can have many roles.

For further detail on Role Hierarchy, access inheritance, Built in Roles, permissions, personas, Role Creation, and so on see Managing access with Role-Based access controls.

The following is an example of an organization with roles and resource permissions:

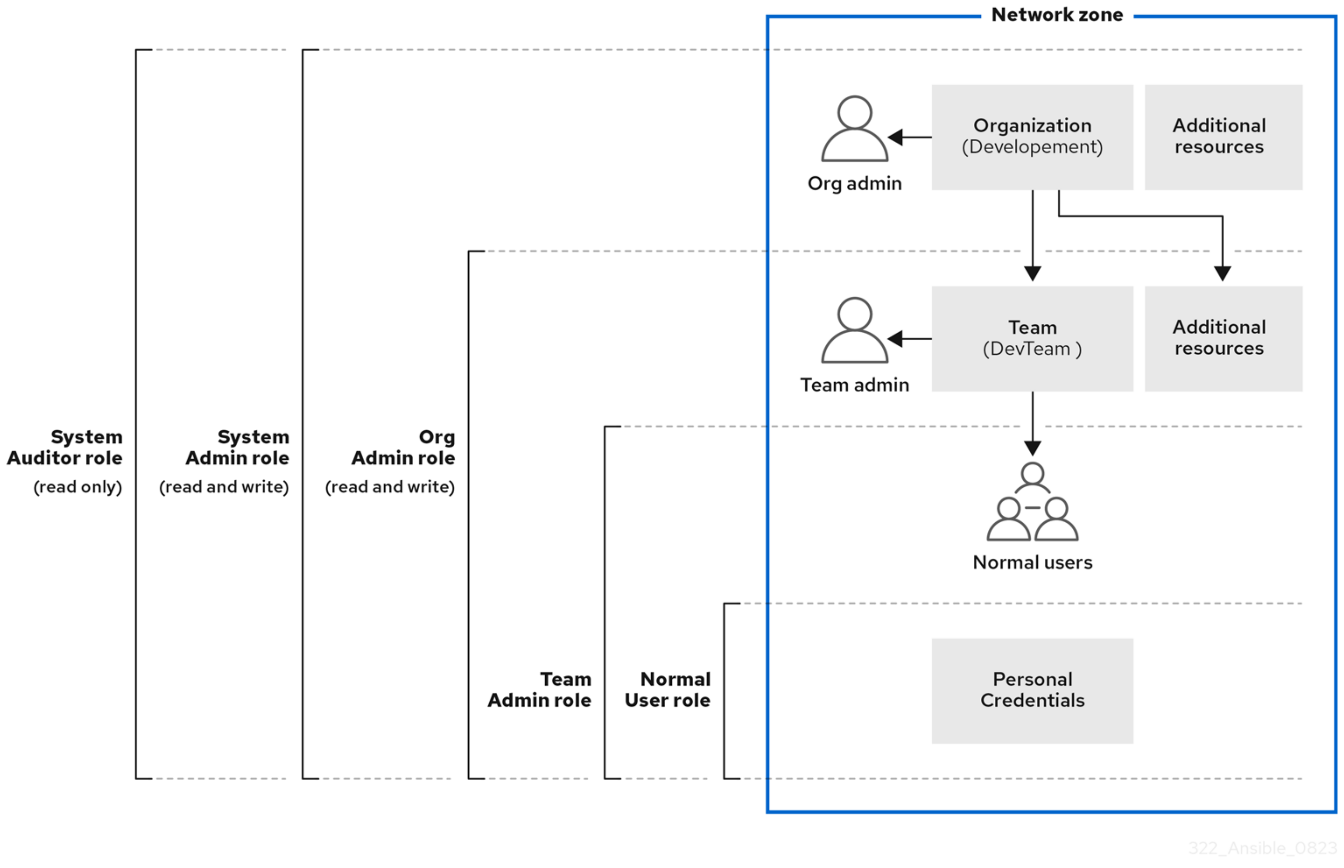

Figure 2. RBAC role scopes within automation controller

User access is based on managing permissions to system objects (users, groups, namespaces) rather than by assigning permissions individually to specific users. You can assign permissions to the groups you create. You can then assign users to these groups. This means that each user in a group has the permissions assigned to that group.

Teams created in automation hub can range from system administrators responsible for governing internal collections, configuring user access, and repository management to groups with access to organize and upload internally developed content to automation hub.

View-only access can be enabled for further lockdown of the private automation hub. By enabling view-only access, you can grant access for users to view collections or namespaces on your private automation hub without the need for them to log in. View-only access allows you to share content with unauthorized users while restricting their ability to only view or download source code, without permissions to edit anything on your private automation hub. Enable view-only access for your private automation hub by editing the inventory file found on your Red Hat Ansible Automation Platform installer.

2.4.2. Updates and upgrades

All upgrades should be no more than two major versions behind what you are currently upgrading to. For example, to upgrade to automation controller 4.3, you must first be on version 4.1.x because there is no direct upgrade path from version 3.8.x or earlier. Refer to Upgrading to Ansible Automation Platform for additional information. To run automation controller 4.3, you must also have Ansible 2.12 or later.

Disaster recovery and continuity of operations

Taking regular backups of Ansible Automation Platform is a critical part of disaster recovery planning. Both backups and restores are performed using the installation program, so these actions should be performed from the dedicated installation host described earlier in this document. Refer to the Backing Up and Restoring section of the automation controller documentation for further details on how to perform these operations.

An important aspect of backups is that they contain a copy of the database as well as the secret key used to decrypt credentials stored in the database, so the backup files should be stored in a secure encrypted location. This means that access to endpoint credentials are protected properly. Access to backups should be limited only to Ansible Automation Platform administrators who have root shell access to automation controller and the dedicated installation host.

The two main reasons an Ansible Automation Platform administrator needs to back up their Ansible Automation Platform environment are:

- To save a copy of the data from your Ansible Automation Platform environment, so you can restore it if needed.
- To use the backup to restore the environment into a different set of servers if you're creating a new Ansible Automation Platform cluster or preparing for an upgrade.

In all cases, the recommended and safest process is to always use the same versions of PostgreSQL and Ansible Automation Platform to back up and restore the environment.

Using some redundancy on the system is highly recommended. If the secrets system is down, the automation controller cannot fetch the information and can fail in a way that would be recoverable once the service is restored. If you believe the SECRET_KEY automation controller generated for you has been compromised and has to be regenerated, you can run a tool from the installer that behaves much like the automation controller backup and restore tool.

To generate a new secret key, perform the following steps:

1. Backup your Ansible Automation Platform database before you do anything else! Follow the procedure described in the Backing Up and Restoring Controller section.
2. Using the inventory from your install (same inventory with which you run backups/restores), run `setup.sh -k`.

A backup copy of the previous key is saved in `/etc/tower/`.