# Oracle Linux Automation Manager 2.2

## Private Automation Hub Installation Guide

G18169-04
November 2024

Oracle Linux Automation Manager 2.2 Private Automation Hub Installation Guide,

G18169-04

# Contents

## 5 Upgrading Private Automation Hub

# Preface

Oracle Linux Automation Manager 2.2: Private Automation Hub Installation Guide describes how to install Oracle Linux Automation Manager Private Automation Hub in single-host deployments.

## Conventions

The following text conventions are used in this document:

| Convention | Meaning |
| --- | --- |
| **boldface** | Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary. |
| *italic* | Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values. |
| `monospace` | Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter. |

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

## Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

## Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

# Preparing to Install Private Automation Hub

This chapter describes the requirements for the systems to be used in an installation of Private Automation Hub.

## Private Automation Hub Hardware Requirements

You can install Private Automation Hub on a single machine in x86-64 Oracle Linux 8 hosts.

Certain operations are memory intensive and require a certain amount of disk space and CPU. A minimum configuration is:

- 4 GB RAM
- 40 GB disk space (170 GB is recommended)
- Two core CPU

These are the minimum requirements to run Private Automation Hub. You must determine any other hardware requirements and capacity based on operational needs.

Hosts must be configured to establish and accept an ssh connection. Consider setting up passwordless login between the deployment host and target hosts to simplify the installation process. For more information, see Set Up Passwordless SSH.

## Installation Options

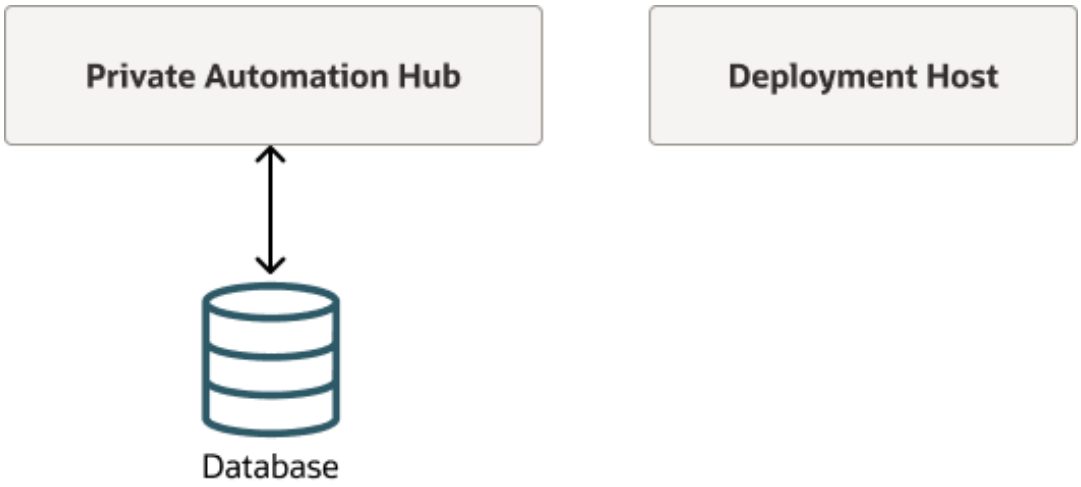
Private Automation Hub provides the following installation options:

- Standalone installation: All components of are on the same host, including the database deployed from the deployment host. The deployment host is a separate system from which you install Private Automation Hub and the database.

**Figure 1-1 Standalone Installation with Local Database**

- Standalone installation with remote database: All components are on the same host, except for the database which is on a remote host. The deployment host is a separate system from which you install Private Automation Hub and the remote database.

**Figure 1-2 Standalone Installation with Remote Database**




# Set Up Passwordless SSH

Set up passwordless SSH connections from the deployment host to the target hosts.

Passwordless SSH is used to copy the CA certificates to the nodes. Set up this feature for the user on the deployment host that installs Private Automation Hub, for example, for the `opc` or `oracle` user.

Set up passwordless SSH between the deployment host and all target hosts.

For example, one way to set up passwordless SSH to the target hosts, is as follows:

1. On the deployment host, use `ssh-keygen` to generate a public and private key pair. For example:

   ```
   ssh-keygen

   Generating public/private rsa key pair.
   Enter file in which to save the key (/home/user/.ssh/id_rsa): <Enter>
   Created directory '/home/user/.ssh'.
   Enter passphrase (empty for no passphrase): <Enter>
   Enter same passphrase again: <Enter>
   ...
   ```

   Press `Enter` each time you're prompted to enter a passphrase.

2. Copy the public key into the `~/.ssh/authorized_keys` file for each target host.
3. To avoid authentication verification prompts, add authenticated hosts to the ~/.ssh/known_hosts file. For example,

   ```
   ssh-keyscan -H <host> >> ~/.ssh/known_hosts
   ```

4. To verify that the deployment host can access the target system without supplying a password, use `ssh` to log each target system. For example:

```
ssh <remote_user>@<host>
```

For more information on setting up passwordless SSH, see Oracle Linux: Connecting to Remote Systems With OpenSSH.

# Enabling Access to the Private Automation Hub Packages

This section contains information on setting up the locations for the operating system on which you want to install the Private Automation Hub software packages.

## Enabling Repositories with the Oracle Linux Yum Server

If you're using the Oracle Linux yum server for system updates, enable the required yum repositories.

To enable the yum repositories:

1. Use the `dnf config-manager` tool to enable the `ol8_baseos_latest` repository.

   ```
   sudo dnf config-manager --enable ol8_baseos_latest
   ```

   > **Note:**
   >
   > This repository is typically enabled by default.

2. Install oraclelinux-automation-manager-release-el8:

   ```
   sudo dnf install oraclelinux-automation-manager-release-el8-2.2
   ```

3. Enable the following yum repositories:

   - `ol8_addons`
   - `ol8_UEKR6` or `ol8_UEKR7`
   - `ol8_appstream`

   Use the `dnf config-manager` tool to enable the yum repositories and do one of the following:

   - If you're using ol8_UEK6, use the following command:

     ```
     sudo dnf config-manager --enable ol8_addons ol8_UEKR6 ol8_appstream
     ```

   - If you're using ol8_UEK7, use the following command:

     ```
     sudo dnf config-manager --enable ol8_addons ol8_UEKR7 ol8_appstream
     ```

4. Ensure that no version of ansible is present on the system. If any are, uninstall them. For example, the following shows that no versions of ansible are installed:

   ```
   rpm -q ansible
   ```

   The following response should appear:

   ```
   package ansible is not installed
   ```

# Enabling Channels with ULN and Setting up a Local Mirror

If you're registered to use ULN, use the ULN web interface to subscribe the system to the appropriate channels.

To subscribe to the ULN channels:

1. Sign in to https://linux.oracle.com with your ULN username and password.
2. On the Systems tab, click the link named for the system in the list of registered machines.
3. Register the deployment and target hosts on ULN. For more information, see Oracle Linux: Managing Software on Oracle Linux.
4. On the System Details page, click **Manage Subscriptions**.
5. On the System Summary page, select each required channel from the list of available channels and click the right arrow to move the channel to the list of subscribed channels. Subscribe the system to the following channels:
   - `ol8_x86_64_automation2.2`
   - `ol8_x86_64_addons`
   - `ol8_x86_64_baseos_latest`
   - `ol8_x86_64_UEKR6` or `ol8_x86_64_UEKR7`
   - `ol8_x86_64_appstream`
6. Click **Save Subscriptions**.
7. Setup a local ULN mirror for the ol8_x86_64_automation2 channel as described in Oracle Linux: Managing Software on Oracle Linux.

> **Note:**
>
> Ensure that you set the `pulp_pkg_repo` variable to point to the location of the ULN mirror. For more information, see Installing Private Automation Hub.

# 2
# Installing Private Automation Hub

This chapter shows you how to set up a host and install the Private Automation Hub software and includes an option for using a remote or local database.

## Setting Up a Remote Database

To setup a remote Postgresql database instance on Oracle Linux 8 for Oracle Linux Automation Manager single host configuration, do the following:

1. Install Oracle Linux 8 on a host.
2. Open the database port in the firewall:

   ```
   sudo firewall-cmd --add-port=5432/tcp --permanent
   sudo firewall-cmd --reload
   ```

3. Enable the `postgresql 13` module stream.

   ```
   sudo dnf module reset postgresql
   sudo dnf module enable postgresql:13
   ```

   > **Note:**
   >
   > For more information about the Postgresql 13 life cycle, see the appendix discussing the application life cycle for stream modules in Oracle Linux: Managing Software on Oracle Linux.

4. Install the database.

   ```
   sudo dnf install postgresql-server postgresql-contrib
   ```

5. Initialize the database:

   ```
   sudo postgresql-setup --initdb
   ```

6. In the `/var/lib/pgsql/data/postgresql.conf` file, switch the password storage mechanism from `md5` to `scram-sha-256`. For example, the following command makes the switch for you:

   ```
   sudo sed -i "s/#password_encryption.*/password_encryption = scram-
   sha-256/"  /var/lib/pgsql/data/postgresql.conf
   ```

7. Start the database using the following command that also ensures that the database restarts in case the host restarts:

   ```
   sudo systemctl enable --now postgresql
   ```

8. Ensure the database is running:

```
sudo systemctl status postgresql
```

9. Create the database user accounts. For example:

```
sudo su - postgres -c "createuser -S -P pulp"
```

10. Enter and confirm the password for the pulp user.

```
Enter password for new role:
Enter it again:
```

11. Create the database.

```
sudo su - postgres -c "createdb -O pulp pulp"
```

12. As the root user, in the `/var/lib/pgsql/data/pg_hba.conf` file add the following line:

```
host  all  all 0.0.0.0/0 scram-sha-256
```

13. As the root user, in the `/var/lib/pgsql/data/postgresql.conf` file in the `# CONNECTIONS AND AUTHENTICATION` section, a line with the text `listen_addresses =` followed by the IP address or host name of the database in single quotes. For example:

```
listen_addresses = '<IP address or host name>'

#listen_addresses = 'localhost'         # what IP address(es) to listen on;
                                        # comma-separated list of
addresses;
                                        # defaults to 'localhost'; use '*'
for all
                                        # (change requires restart)
#port = 5432                            # (change requires restart)
```

In the previous example, *<IP address or hostname>* is the IP address or host name of the database.

14. Calculate and update the memory requirements parameters using the following:

```
max_connections = 1024
shared_buffers = total_mem_mb*0.3
work_mem = total_mem_mb*0.03
maintenance_work_mem = total_mem_mb*0.04
```

In the previous example, *total_mem_mb* is the total memory size in megabytes of the system hosting the database server. For example, if the total available memory on the system were 18 000 MB, then this worksheet would include the following:

```
max_connections = 1024
shared_buffers = 18000*0.3
work_mem = 18000*0.03
maintenance_work_mem = 18000*0.04
```

The final numbers to add are as follows:

```
max_connections = 1024
shared_buffers = 5400MB
work_mem = 540MB
maintenance_work_mem = 720MB
```

15. Add the calculated values to the `/var/lib/pgsql/data/postgresql.conf` file.
16. Restart the database.

    ```
    sudo systemctl restart postgresql
    ```

17. You're now ready to set up hosts as described in Installing on a Single Host.

# Installing on a Single Host

This section provides instructions for installing the Private Automation Hub on a single host where the database is local or on a remote host and assumes that you have setup a passwordless SSH connection.

To set up the host:

1. On the deployment host, login as the user configured with Passwordless SSH to the target host. For more information, see Set Up Passwordless SSH.
2. Ensure python 3.6 is installed on your host. If python 3.6 isn't installed, run the following command:

   ```
   sudo dnf install python36
   ```

3. Install the Private Automation Hub software:

   ```
   sudo dnf install ol-private-automation-hub-installer
   ```

4. Copy the contents of the /single-node folder to a working directory.

   ```
   cp -r /usr/share/ansible/collections/ansible_collections/oraclelinux/
   private_automation_hub/playbooks/single-node/ ~/single_node
   ```

5. From the working directory, create a `hosts` file from the `hosts.singlenode.example`. For example,

   ```
   cd ~/single_node
   cp hosts.singlenode.example hosts
   ```

6. Edit the `hosts` file as follows:

   ```
   all:
     hosts:
       hub:
         ansible_host: <ip_address_or_hostname>
         ansible_user: <username>
   ```

   In the previous example,

- *<ip_address_or_hostname>* is the IP address or host name of the target node where you want to install Private Automation Hub. This host must be reachable using SSH from the deployment host.

  > **Note:**
  >
  > Valid characters for hostnames are a to z, 0 to 9, and the hyphen (-). A hostname may not start with a hyphen.

- *<username>* is the username running the installer playbook commands on the target node where you want to install Private Automation Hub. This user must have sudo privileges.

7. To configure other installation parameters to use during the installation, setup the installation parameter file as described in Configuring the Installation Parameter File.
8. Do one of the following:
   - To install a local database on the same host running Private Automation Hub, run the following command:

     ```
     ansible-playbook single-node-install.yml -i hosts -e
     "olpah_admin_password=<admin_password> olpah_db_password=<db_password>"
     ```

     In the previous example, *<admin_password>* and *<db_password>* are the passwords for the default admin user and the database user account.

     > **Note:**
     >
     > To use the parameter file, add the following to the end of the command:
     >
     > ```
     > -e "@single-node-vars.yml"
     > ```

   - To use an existing database on a remote host, run the following command:
     a. Log into the remote database.
     b. Install the following database extension.

        ```
        sudo dnf install postgresql-contrib
        ```

     c. Restart the database.

        ```
        sudo systemctl restart postgresql
        ```

     d. Create the database user accounts. For example:

        ```
        sudo su - postgres -c "createuser -S -P pulp"
        ```

     e. Enter and confirm the password for the pulp user.

> **Note:**
>
> This must be the same <*db_password*> as specified in the previous step.

```
Enter password for new role:
Enter it again:
```

f. Create the database instance. For example:

```
sudo su - postgres -c "createdb -O pulp pulp"
```

g. From the `single-node-install.yml` file, remove the `pulp_database` role.

h. Set the database hostname or IP address for the remote database (`existing_db_host: <db_hostname_or_ip_address>`) in the `"@single-node-vars.yml"` variables file. For more information about installing using the parameter file, see Configuring the Installation Parameter File.

i. Return to the deployment server and run the following command:

```
ansible-playbook single-node-install.yml -i hosts -e
"olpah_admin_password=<admin_password>
olpah_db_password=<db_password>" -e "@single-node-vars.yml"
```

In the previous example, <*admin_password*> and <*db_password*> are the passwords for the default admin user and the database user account.

9. The host is now ready. Using a browser, you can now log in as the admin user.

```
https://<ip_address_or_hostname>
```

# Configuring the Installation Parameter File

Sometimes, you might want to configure extra parameters when installing Private Automation Hub. If you're configuring extra parameters for a single host installation, edit the `single-node-vars.yml` parameter file.

> **Note:**
>
> You can configure a parameter file before or after you install Private automation Hub. If you do it after installing Private Automation Hub, then you must complete this step and rerun the playbook as described in Installing on a Single Host.

To configure extra installation parameters in a parameter file, do the following:

1. In the parameter file, add the extra parameters you need. For example:

```
existing_db_host: <db_hostname_or_ip_address>
pulp_pkg_repo: "<local_repo_url>"

olpah_require_content_approval: <True or False>
```

```
pulp_api_workers: <Number_of_workers>
connected_olam_controllers: [
 "https://<olam_controller_server_url1>/",
 "https://<olam_controller_server_url2>/",
 ...
]
```

- To use a remote database, add the following parameter to the parameter file. For example,

  ```
  existing_db_host: <db_hostname_or_ip_address>
  ```

  In the previous example, *<db_hostname_or_ip_address>* is the host name or IP address of the remote database.

- To use a remote mirror of ULN or yum repositories, add the following parameter to the parameter file. For example,

  ```
  pulp_pkg_repo: "<local_repo_url>"
  ```

  In the previous example, *<local_repo_url>* is the URL of the remote mirror repository. The URL path might look similar to the following:

  ```
  pulp_pkg_repo: "http://<ip_address>/yum/OracleLinux/OL8/
  automation<version>/$basearch/ol8_x86_64_automation<version>/"
  ```

  In the previous example, *<ip_address>* is the IP address of the repository and *<version>* is the version of the repository.

- To enable the approval process for collection uploads, enable the following parameter in a parameter file.

  ```
  olpah_require_content_approval: True
  ```

  > **Note:**
  >
  > You can do this step before or after you install Private Automation Hub. If you do it after installing Private Automation Hub, then you must complete this step and rerun the playbook as described in this procedure.

- To change the default number of Pulp API workers available for Private Automation Hub, consider setting this value to the same number as the CPU cores as are available on the target instance. For example,

  ```
  pulp_api_workers: 2
  ```

- To link one or more Oracle Linux Automation Manager control servers with Private Automation Hub to enable easier configuration of execution environments in Oracle

Linux Automation Manager, add one or more URL to the following parameter in a parameter file.

```
connected_olam_controllers: [
 "https://<olam_controller_server_url1>/",
 "https://<olam_controller_server_url2>/",
 ...
]
```

In the previous example, *<olam_controller_server_url1>* and *<olam_controller_server_url2>* are the urls to the control servers. You can add more of these URLs depending on the number of control servers you want to make available. The URL must include `https://`. For more information about this feature, see Oracle Linux Automation Manager 2.2: Private Automation Hub User's Guide.

- To integrate Private Automation Hub with an LDAP sever, add the LDAP parameters described in Configuring LDAP Parameters.

# Configuring LDAP Parameters

To configure the LDAP parameters in the parameters file, do the following:

1. Edit the parameter file and add the following required LDAP related parameters to the bottom of the file:

```
#Enable galaxy_ng LDAP Integration
config_ldap: True

# LDAP Binding and Directory Look Up
auth_ldap_server_uri: "<ldap_url>"
auth_ldap_bind_dn: "<ldap_bind>"
auth_ldap_bind_password: "<ldap_bind_password>"
auth_ldap_user_search_base_dn: "cn=users,cn=accounts,dc=example,dc=com"
auth_ldap_user_search_scope: "SUBTREE"
auth_ldap_user_search_filter: "(uid=%(user)s)"
auth_ldap_group_search_base_dn: "cn=groups,cn=accounts,dc=example,dc=com"
auth_ldap_group_search_scope: "SUBTREE"
auth_ldap_group_search_filter: "(objectClass=groupofnames)"
auth_ldap_group_type_class: "django_auth_ldap.config:GroupOfNamesType"
auth_ldap_user_flags_by_group__is_superuser:
"cn=superuserexample,cn=groups,cn=accounts,dc=example,dc=com"
auth_ldap_mirror_groups: False

# LDAP Backend
ldap_logging: True
auth_ldap_start_tls: True
use_galaxy_ldap_self_signed_cert: True
```

In the previous example,

- `config_ldap`
  Set the value of the `config_ldap` parameter to `True` to enable LDAP integration.
- `auth_ldap_server_uri`

Provide the URI to access the LDAP server in the format: `ldap://<host>` where <*host*> is the host name of the LDAP server. This field is required. For example,

```
ldap://ldap1.example.com
```

If the server uses StartTLS functionality, you can set the protocol to ldap within the URI scheme and enable the `auth_ldap_start_tls` option.

- `auth_ldap_bind_dn`

  Provide the Distinguished Name (DN) used to authenticate Oracle Linux Automation Manager against the LDAP server using the Bind operation. This field is required if the LDAP server doesn't allow anonymous access. For example:

  ```
  uid=admin,cn=users,cn=accounts,dc=example,dc=com
  ```

- `auth_ldap_bind_password`
  Provide the Bind password for the Bind DN that you provided before.
- `auth_ldap_user_search_base_dn`

  Provide the DN where your users are listed within the directory.
- `auth_ldap_user_search_scope`

  Provide the scope to use when performing an LDAP search query on the base DN where users are listed. Typically, the scope value is set to either one level deep, ONELEVEL, or to the entire subtree, SUBTREE.
- `auth_ldap_user_search_filter`

  Provide the search filter to be applied when performing an LDAP search query on the base DN where users are listed. You can use the `%(user)s` syntax to match an attribute or key to the username value that a user provided during authentication.
- `auth_ldap_group_search_base_dn`

  Provide the base DN to use when performing an LDAP search query to decide group membership for a user.
- `auth_ldap_group_search_scope`

  Provide the scope to use when performing an LDAP search query on the base DN where groups are listed in the directory. Typically, the scope value is set to either one level deep, ONELEVEL, or to the entire subtree, SUBTREE.
- `auth_ldap_group_search_filter`

  Provide the search filter to be applied when performing an LDAP search query on the base DN where groups are listed in the directory.
- `auth_ldap_group_type_class`

  Provide an appropriate LDAP group type to define how the LDAP server decides group membership for users when performing authorization. LDAP group types map onto the ObjectClasses that are defined for any groups that are listed on an LDAP server and can vary depending on the LDAP server implementation. The values for this parameter are related to the underlying Django framework and the LDAP ObjectClasses that the framework recognizes. Therefore, values are prefixed with `django_auth_ldap.config:`.
- `auth_ldap_user_flags_by_group__is_superuser`

Any user associated with this group has superuser privileges on Private Automation Hub.

> **Caution:**
>
> If you make an error with this value, you can't log into Private Automation Hub after the installation process completes. You must correct the error and run the installation process again before you can log in. If the LDAP server hasn't been configured with the specified superuser group yet, you can't log into Private Automation Hub until the superuser group information has been added to the LDAP server except for the locally defined admin user.

- `auth_ldap_mirror_groups`
  Enabling this feature mirrors LDAP groups associated with a user in Private Automation Hub whenever a user logs into Private Automation Hub. Set this value to True to enable this feature.
- `ldap_logging`

  Set the `ldap_logging` value to True to retain a log of LDAP activity. Logging can help debug authentication issues.
- `auth_ldap_start_tls`

  If the LDAP server uses StartTLS functionality, you can set the protocol to 'ldap' within the URI scheme used in `auth_ldap_server_uri` and set the `auth_ldap_start_tls` value to True.
- `use_galaxy_ldap_self_signed_cert`

  Set the `use_galaxy_ldap_self_signed_cert` value to True if the certificates used for TLS or SSL on the LDAP server are self-signed and you want to disable validation of the certificate against a CA.

# 3
# Installing the Builder Utility

This chapter shows you how to set up the builder utility on a host running x86-64 Oracle Linux 8.

## About the Builder Utility

The Builder utility is an ansible-builder based tool used for generating containers that Oracle Linux Automation Manager can use as execution environments in control or execution nodes to run playbooks. After creating these custom execution environments, you can upload them to Private Automation Hub so that Oracle Linux Automation Manager execution and control nodes can download them when necessary. For more information about creating container environments and uploading them to Private Automation Hub, see Oracle Linux Automation Manager 2.2: Private Automation Hub User's Guide.

## Installing Builder

To install the Builder utility, do the following:

1. On a host running x86-64 Oracle Linux 8, setup the repositories as described in Enabling Access to the Private Automation Hub Packages.

   > **Note:**
   >
   > Don't install the Builder utility on any Oracle Linux Automation Manager host being used as a control node, execution node, or private automation hub nodes.

2. Install the Builder utility.

   ```
   sudo dnf install python3.11-ansible-builder
   ```

   The Builder utility is now installed. For more information about setting up and using the Builder utility to create new container, see Oracle Linux Automation Manager 2.2: Private Automation Hub User's Guide.

4

# Backing up and Restoring Private Automation Hub

The following chapter provides information about backing up and restoring Private Automation Hub. Perform an offline backup where all relevant services are stopped and the data isn't changing at the time of the backup; this backup is consistent by definition.

## Offline Backing up Private Automation Hub

To do an offline backup Private Automation Hub, do the following:

1. Create a backup directories. Ensure you have enough disk space and consider making the directory persistent.

> **Note:**
>
> For the purposes of this document, we use a folder in the home directory, which isn't secure.

```
sudo mkdir -p ~/backup/etc/pulp ~/backup/var/lib/pulp ~/backup/etc/nginx/
pulp ~/backup/var/lib/pgsql/data
```

2. Stop the following services in the following order:

```
sudo systemctl stop pulpcore
sudo systemctl stop nginx
sudo systemctl stop postgresql
```

3. Copy the following folders into the backup directory.

> **Note:**
>
> cp option r ensures that the backup includes all subdirectories and p ensures that all permissions are preserved.

```
sudo cp -rp /etc/pulp/ ~/backup/etc/
sudo cp -rp /var/lib/pulp ~/backup/var/lib/
sudo cp -rp /etc/nginx/pulp ~/backup/etc/nginx/
sudo cp -rp /var/lib/pgsql/data ~/backup/var/lib/pgsql/
```

4. Restart the services in the following order:

```
sudo systemctl start postgresql
sudo systemctl start pulpcore* --all
sudo systemctl start nginx
```

> **Note:**
>
> Consider testing the backup to ensure it works as expected. For more information, see Offline Restoring Private Automation Hub.

# Offline Backing up Private Automation Hub with a Remote Database

To do an offline backup Private Automation Hub with a remote database, do the following:

1. On the Private Automation Hub server, create a backup directories. Ensure you have enough disk space and consider making the directory persistent.

> **Note:**
>
> For the purposes of this document, we use a folder in the home directory, which isn't secure.

```
sudo mkdir -p ~/backup/etc/pulp ~/backup/var/lib/pulp ~/backup/etc/nginx/
pulp
```

2. On the database server, create a backup directory for the following folder. Ensure you have enough disk space and consider making the directory persistent.

```
sudo mkdir -p ~/backup/var/lib/pgsql/data
```

3. Stop the following services on the Private Automation Hub server in the following order:

```
sudo systemctl stop pulpcore
sudo systemctl stop nginx
```

4. Stop the following service on the database server:

```
sudo systemctl stop postgresql
```

5. Copy the following folders into the backup directory from the Private Automation Hub server.

> **Note:**
>
> cp option r ensures that the backup includes all subdirectories and p ensures that all permissions are preserved.

```
sudo cp -rp /etc/pulp/ ~/backup/etc/
sudo cp -rp /var/lib/pulp ~/backup/var/lib/
sudo cp -rp /etc/nginx/pulp ~/backup/etc/nginx/
```

6. Copy the following folders into the backup directory from the database server.

```
sudo cp -rp /var/lib/pgsql/data ~/backup/var/lib/pgsql/
```

7. Restart the services on the database Server:

```
sudo systemctl start postgresql
```

8. Restart the services on Private Automation Hub in the following order:

```
sudo systemctl start pulpcore* --all
sudo systemctl start nginx
```

> **Note:**
>
> Consider testing the backup to ensure it works as expected. For more information, see Offline Restoring Private Automation Hub.

# Offline Restoring Private Automation Hub

To do an offline restore of Private Automation Hub, do the following:

> **Note:**
>
> All data entered after taking an offline backup is lost when the backup is restored.

1. If running, stop the following services in the following order:

```
sudo systemctl stop pulpcore
sudo systemctl stop nginx
sudo systemctl stop postgresql
```

2. Copy the following folders from the backup directory to the Private Automation Hub server.

> **Note:**
>
> cp option r ensures that the backup includes all subdirectories and p ensures that all permissions are preserved.

```
sudo cp -rp ~/backup/etc/pulp /etc/
sudo cp -rp ~/backup/var/lib/pulp /var/lib/
sudo cp -rp ~/backup/etc/nginx/pulp /etc/nginx/
sudo cp -rp ~/backup/var/lib/pgsql/data /var/lib/pgsql/
```

3. Restart the services in the following order:

```
sudo systemctl daemon-reload
sudo systemctl start postgresql
sudo systemctl start pulpcore* --all
sudo systemctl start nginx
```

# Offline Restoring Private Automation Hub with a Remote Database

To do an offline restore of Private Automation Hub with a remote database, do the following:

> **Note:**
>
> All data entered after taking an offline backup is lost when the backup is restored.

1. Stop the following services on the Private Automation Hub server in the following order:

```
sudo systemctl stop pulpcore
sudo systemctl stop nginx
```

2. Stop the following service on the database server:

```
sudo systemctl stop postgresql
```

3. Copy the following folders from the backup directory to the Private Automation Hub server.

> **Note:**
>
> cp option r ensures that the backup includes all subdirectories and p ensures that all permissions are preserved.

```
sudo cp -rp ~/backup/etc/pulp /etc/
sudo cp -rp ~/backup/var/lib/pulp /var/lib/
sudo cp -rp ~/backup/etc/nginx/pulp /etc/nginx/
```

4. Copy the following folders from the backup directory to the database Server.

```
sudo cp -rp ~/backup/var/lib/pgsql/data /var/lib/pgsql/
```

5. Restart the services on the database Server:

```
sudo systemctl daemon-reload
sudo systemctl start postgresql
```

6. Restart the services on Private Automation Hub in the following order:

```
sudo systemctl daemon-reload
sudo systemctl start pulpcore* --all
sudo systemctl start nginx
```

# Offline Restoring Private Automation Hub to a New Host

To do an offline restore of Private Automation Hub to a new host, do the following:

> **Note:**
>
> All data entered after taking an offline backup is lost when the backup is restored.

1. Ensure the new host is installed with the same configuration and settings as the original host where the backup was taken. This includes running identical installer playbooks and the software versions (for example, same database version and Private Automation Hub version). The host and IP address are the only parameters that need to change when running the playbook. For more information, see Installing Private Automation Hub.
2. If running, stop the following services on the original host in the following order:

```
sudo systemctl stop pulpcore
sudo systemctl stop nginx
sudo systemctl stop postgresql
```

3. Copy the following folders from the backup directory to the new Private Automation Hub server using whatever method you need. Copy the backup files to the following directories:

> **Note:**
>
> cp option r ensures that the backup includes all subdirectories and p ensures that all permissions are preserved.

```
sudo cp -rp ~/backup/etc/pulp /etc/
sudo cp -rp ~/backup/var/lib/pulp /var/lib/
sudo cp -rp ~/backup/etc/nginx/pulp /etc/nginx/
sudo cp -rp ~/backup/var/lib/pgsql/data /var/lib/pgsql/
```

4. Update the /etc/pulp/settings.local.py on the new host with the following parameters:

```
DATABASES = {'default': {'HOST': '<IP address or host name>', 'ENGINE':
'django.db.backends.postgresql', 'NAME': 'pulp', 'USER': 'pulp',
'PASSWORD': '<db_password>'}}
TOKEN_SERVER = 'https://<<IP address or host name>>/token/'
and
CONTENT_ORIGIN = 'https://<IP address or host name>'
```

   In the previous example, <IP address or host name> is the IP address or host name of the standalone Private Automation Hub server and <db_password> is the password for the database.

5. Restart the services in the following order:

```
sudo systemctl daemon-reload
sudo systemctl start postgresql
sudo systemctl start pulpcore* --all
sudo systemctl start nginx
```

# Offline Restoring Private Automation Hub to a New Host with Remote Database

To do an offline restore of Private Automation Hub to a new host with remote database, do the following:

> **Note:**
>
> All data entered after taking an offline backup is lost when the backup is restored.

1. Ensure the new host is installed with the same configuration and settings as the original host and remote database where the backup was taken. This includes running identical installer playbooks and the software versions (for example, same database version and Private Automation Hub version). The host and IP address are the only parameters that need to change when running the playbook. For more information, see Installing Private Automation Hub.
2. If running, stop the following services on the Private Automation Hub server in the following order:

```
sudo systemctl stop pulpcore
sudo systemctl stop nginx
```

3. If running, stop the following service on the database server:

```
sudo systemctl stop postgresql
```

4. Copy the following folders from the backup directory to the new Private Automation Hub server using whatever method you need. Copy the backup files to the following directories:

> **Note:**
>
> cp option r ensures that the backup includes all subdirectories and p ensures that all permissions are preserved.

```
sudo cp -rp ~/backup/etc/pulp /etc/
sudo cp -rp ~/backup/var/lib/pulp /var/lib/
sudo cp -rp ~/backup/etc/nginx/pulp /etc/nginx/
```

5. Update the `/etc/pulp/settings.local.py` on the new Private Automation Hub server with the following parameters:

```
DATABASES = {'default': {'HOST': '<database IP address or host name>',
'ENGINE': 'django.db.backends.postgresql', 'NAME': 'pulp', 'USER': 'pulp',
'PASSWORD': '<db_password>'}}
TOKEN_SERVER = 'https://<<IP address or host name>>/token/'
and
CONTENT_ORIGIN = 'https://<IP address or host name>'
```

   In the previous example, <database IP address or host name> is the IP address or host name of the database, <IP address or host name> is the IP address or host name of the standalone Private Automation Hub server and <db_password> is the password for the database.

6. Copy the following folders from the backup directory to the new database server using whatever method you need. Copy the backup files to the following directory:

```
sudo cp -rp ~/backup/var/lib/pgsql/data /var/lib/pgsql/
```

7. Restart the services on the database Server:

```
sudo systemctl daemon-reload
sudo systemctl start postgresql
```

8. Restart the services on Private Automation Hub in the following order:

```
sudo systemctl daemon-reload
sudo systemctl start pulpcore* --all
sudo systemctl start nginx
```

# 5
# Upgrading Private Automation Hub

The following chapter provides information about upgrading Private Automation Hub.

## Upgrading 2.1 to 2.2

To upgrade Private Automation Hub from 2.1 to 2.2, do the following:

1. Consider performing an OS backup. Backing up a system is good practice so that the system can be restored to its former state if the upgrade fails.
2. Create a backup of Private Automation Hub. For more information, see Backing up and Restoring Private Automation Hub.
3. On the system running Private Automation Hub, stop all the Private Automation Hub services:

   ```
   sudo systemctl stop pulpcore
   ```

4. Check the status to ensure all services are stopped:

   ```
   sudo systemctl status pulpcore* -all
   ```

5. On the local or remote database host, log in to the user account that controls the database.

   ```
   sudo su - postgres
   ```

6. Verify which version of `postgresql` is running.

   ```
   rpm -q postgresql
   ```

   For example, the following response shows `postgresql` version 10:

   ```
   postgresql-10.23-4.0.1.module+el8.9.0+90164+87901204.x86_64
   ```

7. Do one of the following:
   - If version 13 is installed, go to step 17.
   - If version 10, 11, or 12 is installed, go to the next step.
8. Export the database. Exporting the database creates a script file containing all the necessary SQL commands and input data to restore the databases. For example, this command creates the `hub.dump` file in the database home directory:

   ```
   pg_dumpall > hub.dump
   ```

9. Exit the database user account:

   ```
   exit
   ```

10. Stop the database server:

```
sudo systemctl stop postgresql
```

11. Remove ( and optionally backup ) existing database data directory. For example, the following command removes and creates a backup file in the home directory:

```
sudo mv /var/lib/pgsql/data/ ~/data.old
```

12. Remove the current version of the database:

```
sudo dnf remove postgresql
```

13. Reinstall the Database as described in Setting Up a Remote Database and follow the procedure up to and including the step to restart the database and return to this step but ignore steps 9 to 11 create user and database pulp as these objects are already in the dump file.
14. After the update completes, set up the database by importing the `hub.dump` file. Run the following commands:

```
sudo su - postgres
psql -d postgres -f hub.dump
exit
```

15. On the Private Automation Hub server, restart the private automation hub service and NGINX processes:

```
sudo systemctl start pulpcore* -all
sudo systemctl start nginx
```

16. Test the Private Automation Hub functionality to ensure everything is working as expected.
17. On the Private Automation Hub server, run the following command:

```
sudo dnf install oraclelinux-automation-manager-release-el8-2.2
sudo dnf config-manager --disable ol_automation2
```

18. On the Private Automation Hub deployment server, run the following command:

```
sudo dnf install oraclelinux-automation-manager-release-el8-2.2
sudo mv /etc/yum.repos.d/oraclelinux-automation-manager-ol8.repo /etc/
yum.repos.d/oraclelinux-automation-manager-ol8.repo.OLD
sudo mv /etc/yum.repos.d/oraclelinux-automation-manager-
ol8.repo.rpmnew /etc/yum.repos.d/oraclelinux-automation-manager-ol8.repo
```

19. Update the OS on the Private Automation Hub server and the deployment server:

```
sudo dnf update -y --refresh
```

20. On the Private Automation Hub server, run the collect static command:

```
sudo PULP_SETTINGS=/etc/pulp/settings.py /usr/bin/pulpcore-manager
collectstatic --clear --noinput
```

21. On the deployment host, run the installer for Private Automation Hub with the same user account originally used to install Private Automation Hub.

```
ansible-playbook -i hosts single-node-install.yml -e
"olpah_admin_password=<admin_password> olpah_db_password=<db_password>"
```

> **Note:**
>
> If you used a parameters file when you first created the server, you must use it again when running this command. For example, add the following to the end of the command line `-e "@single-node-vars.yml"`.

# Upgrading Builder Utility 2.1 to 2.2

To upgrade Private Automation Hub Builder utility from 2.1 to 2.2, do the following:

1. On the host you installed the previous version of the Builder utility, run the following commands:

```
sudo dnf install oraclelinux-automation-manager-release-el8-2.2
sudo mv /etc/yum.repos.d/oraclelinux-automation-manager-ol8.repo /etc/
yum.repos.d/oraclelinux-automation-manager-ol8.repo.OLD
sudo mv /etc/yum.repos.d/oraclelinux-automation-manager-
ol8.repo.rpmnew /etc/yum.repos.d/oraclelinux-automation-manager-ol8.repo
sudo dnf update python39-ansible-builder
```

2. Update any `execution-environment.yml` files to the 2.2 images of `olam-ee` and `olam-builder`. For example, find the following entries:

```
base_image:
name: container-registry.oracle.com/oracle_linux_automation_manager/olam-
ee:latest
builder_image:
name: container-registry.oracle.com/oracle_linux_automation_manager/olam-
builder:latest
```

   And change `latest` to `2.2`:

```
images:
base_image:
name: container-registry.oracle.com/oracle_linux_automation_manager/olam-
ee:2.2
builder_image:
name: container-registry.oracle.com/oracle_linux_automation_manager/olam-
builder:2.2
```

3. If using Private Automation Hub to store `olam-ee` or `olam-builder` images then update the images to `2.2` and point the images in `execution-environment.yml` to the new `2.2` images.
4. Ensure that any execution environments created in previous releases that are intended to be used in 2.2 are evaluated for changing dependencies. For example, if previous

execution environments were created using older versions of `ansible-core`, confirm that collections or modules used in these older execution environments continue to function with later versions of `ansible-core` running on Oracle Linux Automation Manager 2.2 nodes.