Tutorial: orcharhino in a box

Introduction

- Introduction
  - Structure
  - orcharhino-in-a-box Test Infastructure
    - How to access your orcharhino-in-a-box
- orcharhino Features
  - Provisioning
    - Compute Resources
    - Compute Profiles
    - Host Groups
      - (Different Ways for Provisioning a Host)
    - Hands-On: Create a Host
  - Release Management
    - Hands-On
      - Create a new version of a Content Viewc
      - Promote Lifecycle to new version
      - Create a new Content View
      - Create an Activation Key (optional)
  - Patch Management
    - Hands-On
      - Package Updates
      - Errata Updates
  - Remote Execution
    - Hands-On
  - Ansible Roles
    - Workflow for working with Ansible and orcharhino
    - Hands-On: Import Roles from orcharhino host
    - Assigning Roles to Hosts or Host Groups
    - Managing Ansible Variables
    - Hands-On: Configure a Role for a Host/ Host Group
  - User Management
    - Overview
    - Hands-On: Add a user with restricted permissions
    - Hands-On: Inspect Audit logs

This document is designed to provide an overview of how orcharhino works and to allow users to explore and understand its key features.

Structure

This document is structured into two parts: **explanations** and **exercises**. For each feature of orcharhino that we discuss, there is a section in the explanations part and a corresponding

exercise in the exercise part. In the explanation you will learn about the relationships, terminology, and background of the feature. How this works in practice is then demonstrated in the respective exercises directly within orcharhino.

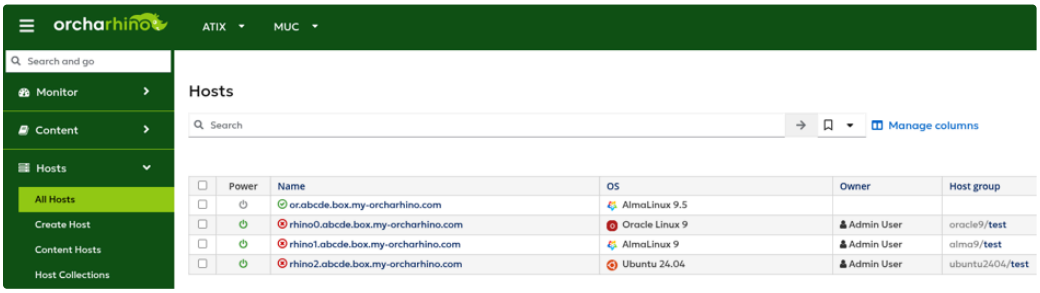
Features are further grouped into **basic features** and **advanced features**. We recommend that you try out all of the basic features in order. This allows us to reduce repetition in the explanations and give you a more coherent learning experience. For the advanced features, these are largely independent, feel free to look into them in any order and depth that you would like.

If you want to get a quick overview, you can also try out the exercises without reading the full explanations. In case you have any questions, you can always come back to the explanations to gain a deeper understanding of the things you are doing in the exercises.

orcharhino-in-a-box Test Infastructure

For the exercises, you have access to a dedicated instance of orcharhino with which you can perform all the exercises and play around in order to understand how orcharhino works. This test system is referred to as **orcharhino-in-a-box** in the following.

This is a basic orcharhino installation where AlmaLinux 9, Oracle Linux 9, and Ubuntu 24.04 are already setup, and VMs for these operating systems are already available for testing:



How to access your orcharhino-in-a-box

You should have a username and password with which you can log into the orcharhino UI of the orcharhino-in-a-box at its FQDN. You should also have received an email with a test license attached (the file name ends with the extension `.osk`). When you log into the orcharhino, you be greeted with a dialog that reminds you to add a valid license. This is also reflected by the color of the rhino at the top left of the UI, which is red, and will turn green once a valid license has been supplied. In order to do that, navigate to Administer → orcharhino subscription, press “Upload subscription key” and upload the `.osk` file you received via e-mail. The rhino in the top right should of the UI should become green again and you are ready to work with your orcharhino-in-a-box.

orcharhino Features

Provisioning

Compute Resources

Compute Resource is what hypervisors are called in orcharhino. When you create a VM, you can specify which of your compute resources that are connected to orcharhino you want to use for this. orcharhino supports the following providers of compute resources: EC2, Google, Libvirt, OpenStack, oVirt, Proxmox, VMware.

Compute Profiles

Host Groups

Usually, you will have a number of different scenarios for which you will need a varying number of hosts. Instead of always specifying how the a new host should be created manually, you can template these settings by creating so-called host groups.

An example would be that you need to regularly create medium-sized Ubuntu 24.04 VMs that are used as test systems for developers. You could then create a host group named `Ubuntu 24.04 Testing` which specifies things like the compute profile (number of CPUs, memory etc.), accessible package versions (via content views and life cycles), network settings, ansible roles, host provisioning method and so on. When creating a new test system, you just need to fill in the host group and all configuration options are automatically set. If you want to later on get an overview of how many test systems you have, or in case you want to change some aspect of all test systems, you can conveniently refer to these hosts via their host groups. When creating a new host group, you can use another host group as parent, which will make it inherit all settings from its parent. This way you can separate concerns, e.g. have a host group `Ubuntu 24.04` that contains settings that are important for all `Ubuntu 24.04` machines, and then have a child host group `Ubuntu 24.04/Test System` that sets all the aspects that are important for a test system.

(Different Ways for Provisioning a Host)

With orcharhino, you can provision cloud instances, virtual machines, and bare metal machines. In this tutorial, we will focus on virtual machines
orcharhino offers 3 main ways in which a virtual host can be deployed:

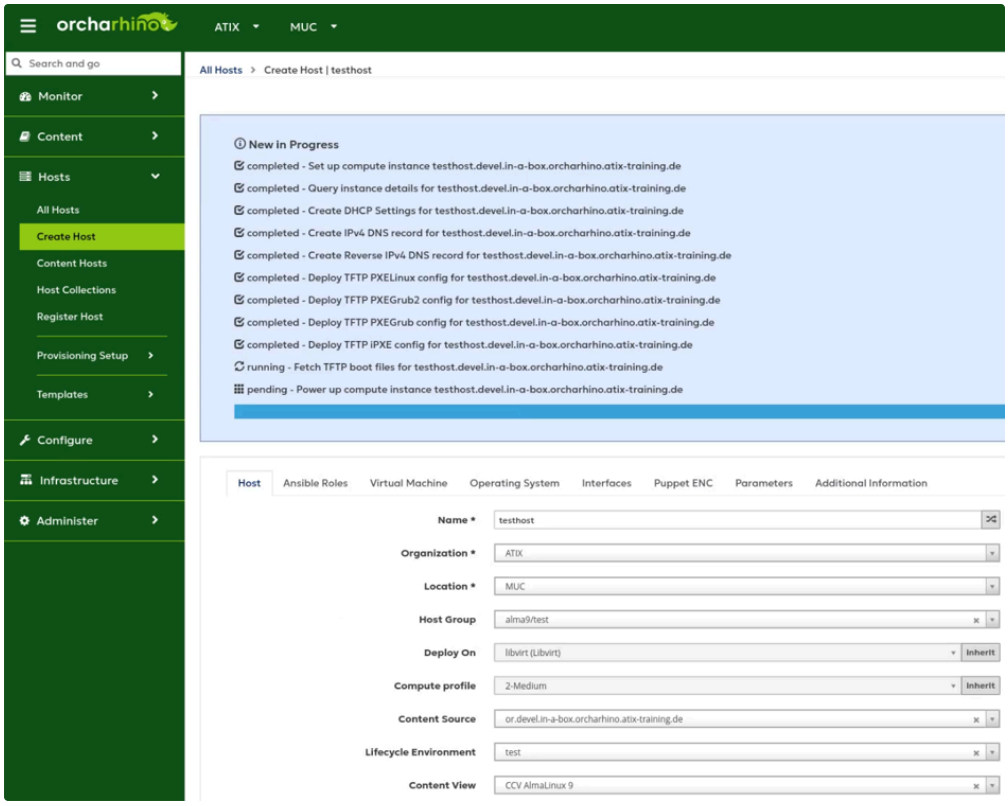
- ***Network Based Provisioning:*** the host boots over network and boots via PXE/iPXE
- ***Boot Disk Based Provisioning:*** the host starts with a boot disk inserted into the VM, which contains information about the network configuration and then loads everything else

- **Image Based Provisioning:** in case the compute resource supports cloning VMs from templates, this mechanism is used to quickly create a VM from a pre-existing template/image

All methods except the boot disk based provisioning require a working DHCP (orcharhino can either provide a DHCP itself or needs access to an already existing DHCP). For more overview, please have a look at

Hands-On: Create a Host

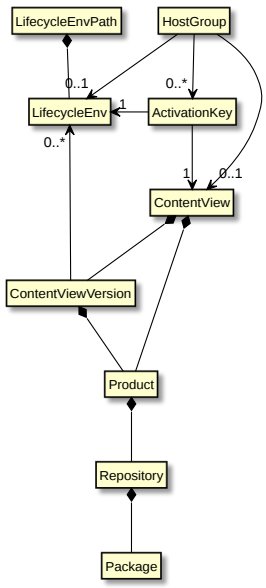
1. Go to Hosts → Create Host
2. In the Host Creation Form you can:
   a. Enter the hostname for your new host (shortname not FQDN), e.g. `testhost` or keep the randomly generated preset
   b. Select the host group `alma9/test` → this will fill out most of the required parameters, e.g. Content Source, Lifecycle Environment and Content View on this tab, but also on other tabs of the Create Host page; “Deploy On” will be set to libvirt, which is the compute resource on the test system
   c. Make sure to set a Root Password in the “Operating System” tab (at least 8 characters)
   d. Feel free to check the other Tabs and prefilled values
3. You can now press submit to create this host or you can optionally override some definitions of the host group:
   a. You could decide to override the Compute profile to adjust the sizing of the VM
   b. You could change the size of the harddisk in the VM-Section
   c. You could override the host parameter “package_upgrade” and set it to “false” in the parameter section to install all packages only from the installation medium without updating → with that you can reuse this host to test the patching mechanism
4. After Submission, orcharhino will briefly list the individual steps that are performed to create the VM including things like DNS, DHCP, and TFTP entries. Then end in the host details for your created host



Steps performed when host creation form is submitted (only visible briefly, so a screenshot is included here for reference)

5. Optional: Select “Webconsole” in the host details menu (three dots in the right top corner) to watch installation

Release Management



Relation between the various concepts inside orcharhino that are used for release management

In Release Management, we want to have control over which package versions end up at the hosts, such that we can perform package upgrades with well-defined package states. We will have a look through the terminology of orcharhino using the overview picture on the left, starting from bottom to top.

At the bottom we have the familiar concept of **packages** (think of apt or rpm packages), which migsht occur in different **package versions**. These are provided inside a **repository** (for example on EL you would usually see at least the repositories BaseOS, AppStream, and Extras).

Now inside orcharhino, we bundle multiple repositories into a so-called **product**. This makes sense in practice, since we can then refer to all repositories that belong to e.g. AlmaLinux 9 by adding a single product “AlmaLinux 9” that contains the right repositories.

You can also create products that are not related to the operating system. For example assume your company has a company-internal software named “FancyTool” which is available as package inside a apt repository. Then you can also add a product “FancyTool” that includes that internal apt repository.

Now we are ready to talk about **content views**. A content view is a way to define the set of products that we want to version. Let us assume that our company wants to have quarterly releases where new package versions are installed to the hosts. Then each quarter, you would create a new snapshot of the repositories inside a content view. This snapshot is called a **content view version**, because it is a version number that counts up for the corresponding content view. For each content view there can be multiple content view versions, such that you later have different hosts using different snapshots of the package repositories. You can also create hierarchies by forming so-called **composite content views**, i.e., content views that contain other content views instead of products.

So far we looked at how orcharhino organizes different package versions for controlled releases. Now we need to talk about how these packages end up at the hosts, so we will go from the top of the diagram until we reached the content view and content view version. Since it would be prone to errors if we define this for each host individually, we instead group hosts in so-called **host groups**. For each host group, we can set at most one content view.

Now, how do we specify which content view version the hostgroup gets from the content view? In our example, we might want to advance the package states every two weeks. The latest content view version shall be assigned to hosts in the “development” category, followed by hosts in the “test” category, which one version earlier than “development”, followed by hosts in the “production” category, which get one version earlier than “test“. There might be more stages to the release process, the important take-away message is that it is a predetermined sequence.

In orcharhino, these categories are called **lifecycle environments**, and the sequence they form is called **lifecycle environment path**. Each of these paths starts with the implicit lifecycle environment “library” (meaning take the most recent package versions). So in our example we would have a lifecycle path containing the lifecycle environments library, development, test, and production.

An **activation key** is a way to specify that a host from a hostgroup is mapped to a specific lifecycle environment. Now how do we determine which version of a content view the host will see?
To achieve that, each lifecycle environment can be assigned to at most one content view version inside the same content view.

Hands-On

Create a new version of a Content Viewc

1. Go to Content → Lifecycle → Content Views
2. Click on a Content View for which you want to create an new version. the content view should have CV in the name (not CCV), e.g. AlmaLinux 9_CV or Ubuntu 24.04_CV
3. Create a new version using the “Publish new version” button. You can give a description but it is optional. In this case leave the Promote disabled, click on Next and then Finish; after some time, the new content view version should be successfully created
4. As the pre-configured content views are organized into composite content views to have the possibility of modular updates, you need to publish a new version of the composite content view (CCV in the name) as well.
5. Go to Content → Lifecycle → Content Views and click on the corresponding composite content view (CCV AlmaLinux 9 or CCV Ubuntu 24.04)
6. You should see a small arrow symbol next to the most recent version of the composite content view, indicating that content views in this composite content view have been updated
7. Publish a new version of the composite content view and observe that the arrow symbol is gone afterwards

Promote Lifecycle to new version

1. Go to Content → Lifecycle → Content Views
2. Select the Composite Content View, in which you published a new version previously.
3. For the version you want to connect with a new lifecycle, select “Promote” in the menu (three dots on the right) in the list of versions or click on the version and use the blue “Promote”-Button.

4. Select the lifecycle environment(s) you want to link to this version and click on promote.
5. Optional: If you have promoted a newer version of repositories in a lifecycle, which is in use of hosts, you should also see that new updates are available (unless your orcharhino-in-a-box was created recently, in which case new kagespackages might simply not be available yet)

Create a new Content View

1. Go to Content → Lifecycle → Content Views
2. “Create content view” - select either a “Regular content view” (versioned) or “Rolling content view” (always latest sychronized repository) and give it a name.
3. After creation click on “Show repositories” or search for the repositories you want to add and add all repositories you want to combine in this content view.
4. If you created a regular content view you have to publish your first version (you can also directly promote it to other lifecycles) - in case of a rolling content view this is not needed.

Create an Activation Key (optional)

1. Go to Content → Lifecycle → Activation Keys
2. Click on “Create Activation Key”
3. Select a name, lifecycle and content view (for rolling content view the lifecycle environment has to be “Library”)
4. Go to “Repository Sets” in your activation key and decide which repositories should be enabled (Select Action → Override to Enabled)
5. You can now use this activation key to register new hosts or configure it in a (new) hostgroup. Machines will have the repositories of this activation key available right after provisioning.

Patch Management

For patch management we want to have control which packages are installed, which packages are updated on selected machines and also an overview, where we have security relevant updates (=errata) available including the possibility to apply them.

All hosts registered to orcharhino are able to report their installed packages to orcharhino, and this allows the hosts to obtain the repositories from orcharhino. With that you can check for every host which packages are installed and in which version. As orcharhino provides the repositories, it can calculate and show which updated versions of those packages are available in the connected repositories for this system. Using remote execution you can trigger jobs to install, update, remove packages on your managed systems. Those jobs can be executed immediately or can be scheduled to be executed at a time you choose. Configuring and setting up remote jobs - orcharhino documentation

Errata contain the information on known issues - mostly security-related - including the CVE-IDs and the list of packages and affected version numbers. With that orcharhino can identify which systems have those packages installed in a lower version and will highlight that those systems are affected by security issues, which can be fixed. By applying an erratum the remote execution is used to trigger an update of those affected packages to a version, which fixes the described issues. Managing errata - orcharhino documentation

Instead of doing this for every host separately, you can also manage those actions for multiple hosts at once. You can select multiple hosts from the host list or use the search mode in the remote execution job to target your desired selection. If you often have to manage the same group of hosts, you can add them to a host collection, which is a group you can easily reuse for patching, errata-management etc. However, it is recommended not to mix operating systems and major versions in those remote execution jobs, as their packages and versions may differ. Configuring host collections - orcharhino documentation

Hands-On

Package Updates

1. Go to Hosts → All Hosts and go to the host page of a host with updates available (click on the host's FQDN)
   Tip: The number of installable updates per type (security, bug fix, enhancement, regular) can be seen on the "All Hosts" page if you go to "Manage Columns" and check "Installable Updates" in the "Content"-section
2. on the host page go to the Content-Tab → Packages. There you can see all installed packages and where updates are available. (You can filter for upgradable packages only in the selection showing "Status")
3. Select packages you want to upgrade (or all packages) and click on "Upgrade" to trigger the installation of the patches immediately
4. Optional: Select "Upgrade via customized remote execution" in the drop-down menu of the "Upgrade" button. In the Job settings go to "4. Schedule" and select "Future execution" to plan the execution in future. The exact time can then be set under Schedule → Future execution

Errata Updates

1. Find a host with installable security updates (see Package Upgrades Step 1).
2. In the host details go to the Content-Tab → Errata. This will show you a list of errata affecting this host. If you click on the errata-ID, you will get a description, usually CVE-IDs and a list of packages in which the described issues are solved

3. Select one, multiple or all installable errata and “Apply” them - similar to the regular patching.
4. Optional: you can click on “Go to job details” once you started the job, then click on the host’s FQDN on the bottom left to read through the standard output of the update command. You can also find the list of executed jobs under Monitor → Jobs

Remote Execution

The remote execution in orcharhino allows users to run jobs against the registered machines. In the background SSH, Ansible or an agent-based connection on base of the MQTT protocol is used. Configuring and setting up remote jobs - orcharhino documentation

For such jobs, orcharhino is shipped with templates for common things like updating packages, restarting services etc. However, you can create your own jobs using those templates to have the possibility to reuse jobs everytime you need them.

Depending on the tasks you can also benefit from the scheduling possibilities. The default is an immediate execution, but it is also possible to schedule it to start it at a certain time (together with a “starts before” time, where it will not start new jobs on hosts if they did not start up to that time) or as a recurring job - to run it daily, weekly or according to a cronline.

Hands-On

1. Go to host details of a host of your choice (not the orcharhino itself)
2. Select “Schedule a job”
3. You could select a template from different categories but in this case keep the category “Commands” and the template “Run Command - Script Default” - click on “next”
4. In the next screen you can enter a command like “uptime”
5. You can check the other pages but those can be skipped by clicking on “5 Review Details” and then “Submit”
6. Optional: you can schedule the job for future execution under “4 Schedule”
7. “Run on selected hosts” to trigger the job
8. In the job overview you can see the result
9. If you click on the hostname in the bottom part of the job details, you can see the output of the script/command - like the output of “uptime”

Ansible Roles

Workflow for working with Ansible and orcharhino

orcharhino can be combined with Ansible in several ways. The most common way (which is also described in our documentation: Configuring hosts by using Ansible - orcharhino documentati

on) is that you manage your hosts' roles and their Ansible variable assignment via orcharhino. You install your roles locally on the orcharhino, and then assign the roles to the hosts (or hostgroups) using the orcharhino (UI). You can adjust the values for Ansible variables in general (by setting a default) and for individualization you can also set different values depending on criteria like hostsname/hostgroup/fqdn with so-called matchers (more on that later). The assignment of roles to hosts/hostgroups and adjustment of variables inside the orcharhino UI effectively replaces what you would otherwise specify in your Ansible inventory.

Roles are automatically executed upon host provisioning(see Provisioning chapter), manually executed whenever required, or executed according to a scheduled job inside the orcharhino. Other ways to combine orcharhino with Ansible exist, for example you can use orcharhino to generate a (dynamic) inventory for use with Ansible. This is useful if you already have your ansible workflow and want to stick with this for now. There is no guide on this yet, if you are interested in this please let us know.

Yet another way to combine Ansible and orcharhino is to run dedicated playbooks as remote execution jobs via orcharhino. This is described in the section for Remote Execution.

Hands-On: Import Roles from orcharhino host

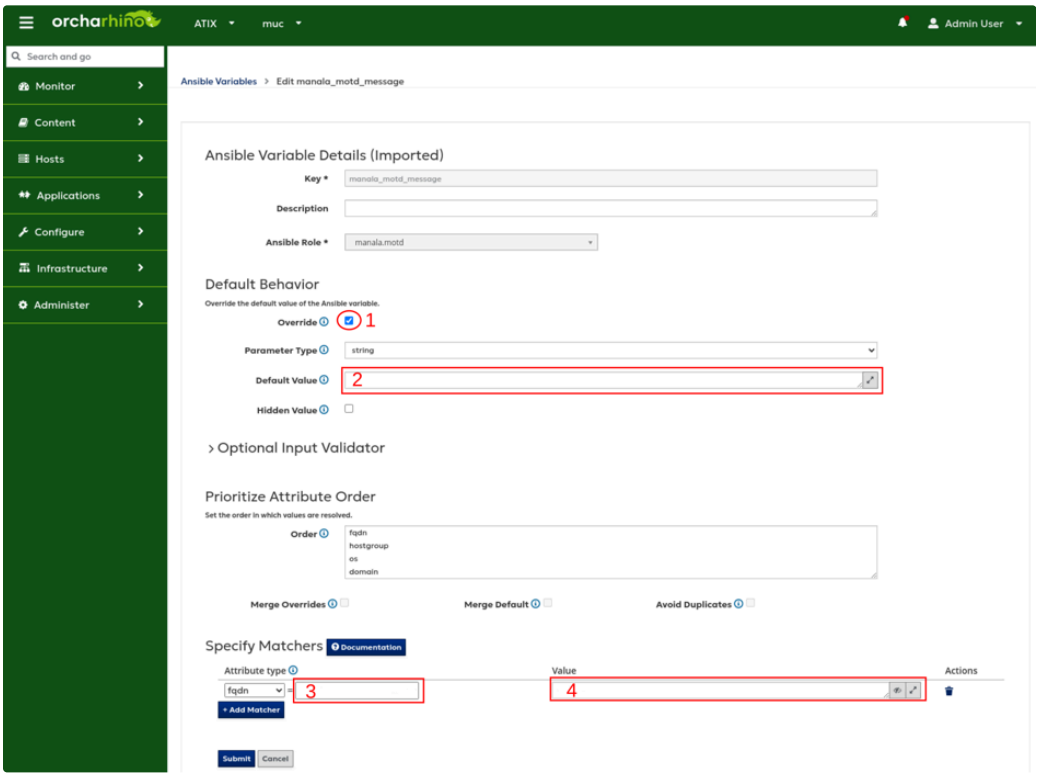
orcharhino can import roles from its host (the machine on which orcharhino runs) automatically via the UI. For this exercise, we already placed some roles to `/etc/ansible/roles` on the orcharhino's host. Go to Configure→Ansible-Roles, where there should be only the role `manala.motd` listed. Click on "Import from <orcharhino-fqdn>", after which the roles `geerlingguy.swap` and `manala.motd` should be listed, as these where found on the orcharhino host. Import `geerlingguy.swap` by clicking the checkbox next to it and click "Submit". Note that orcharhino automatically detected variables for the role. Click on one of the variable counts for one of the imported roles to get a listing of the variables that have been imported for that role.

Assigning Roles to Hosts or Host Groups

- How it works:
  - Roles can be assigned per host group and also individually for each host
  - execution is done in a configurable order upon host provisioning
  - roles can be executed again via a job either manually or scheduled
  - role execution uses the remote execution user
  - role variables can be adjusted via matchers for specific hosts/host groups or other patterns
  - pull-based execution is also possible (cf. documentation)

Managing Ansible Variables

In orcharhino, variables for imported ansible roles are automatically listed under Configure → Ansible → Variables. Here you can adjust the default values and also add so-called **matchers** to make custom exceptions from the default values to ensure all hosts get the variables set to exactly the values you want them to have. For example, you have a role that sets the NTP server for a host. You have a default NTP server in your organization, so you set its FQDN as the default value for the corresponding variable of that role. Now let's assume you have a special hostgroup `DMZ` where the hosts are in a special subnet from which that default NTP server is not available. Then you would add a matcher that overrides this variable to point to a different NTP server for all hosts where `hostgroup == DMZ`. Instead of matching for the hostgroup, you could also match (parts of) the hostname, the operating system, or the domain of the host. You can also add a variable directly to a host or host group, in which case



Workflow for setting variables in orcharhino: once you mark the checkbox (1), the other input options become available

Hands-On: Configure a Role for a Host/ Host Group

In this exercise we will add a role to a host via its host group and adjust the role variables such that we can address the host individually. Once this is done we will let orcharhino run the role and verify that the host's configuration is indeed how we specified it. We will use the host `rhino1` and the host group `alma9` (and its sub host group ending in `/test`) in this

hands-on section, feel free to use `rhino0` and `oracle9` or `rhino2` and `ubuntu2404` instead.

Add a Role to a Host Group:

- Configure → Host Groups → Click on the `alma9` host group → Tab "Ansible Roles"
  - Select `geerlingguy.swap` and `manala.motd`
    - Observe that you can change the order of the selected roles on the right via drag and drop (in case role execution order matters)
    - press submit

Run a role for a Host in the Host `Group`:

- The pre-existing test host (`rhino1`) for AlmaLinux 9 should already be part of the host group `alma9` (or rather `alma9/test`), so it should inherit the roles we just assigned to the Host Group. We will use this host to demonstrate how roles are configured and run. (alternatively to you can create a new host in the hostgroup `alma9/test` as explained in the section how to create a host)
- Verify that the roles are correctly inherited from the Host Group: click on the host `rhino1` for the `alma9/test` host group in the Hosts → All Hosts overview, go to the Ansible tab and click on “View inherited roles”
- So far, roles were not configured, so they use their default variable values and either do nothing (in the case of `manala.motd`) or follow their default behavior (in the case of `geerlingguy.swap` , which adds a 512MB swapfile to `/swapfile`
- We want to observe that `geerlingguy.swap` does by default what it is supposed to, so we need to run the roles on the host
  - In the Hosts → All Hosts overview, click the checkbox next to the `rhino1` host
  - Click on “Select Action” on the top right and then on the drop-down arrow next to Schedule a job → select "Run all Ansible roles"
  - In the job overview that is now shown, click on the host’s FQDN at the bottom to see the job’s (ansible) output
  - observe the ansible tasks that are performed in order to setup the swap
- Optional: to test if swap is really enabled using a remote execution command (as outlined in the Remote Execution section):
  - In the Hosts → All Hosts overview, click the checkbox next to the `rhino1` host

    - Click on “Select Action” on the top right and then on the drop-down arrow next to Schedule a job → select "Schedule Remote Job"
    - Click on “Next” and enter the following command under “command”: `swapon -s`
    - Click on “Run on selected hosts”
    - Inspect the job output like before; it should list the kB equivalent of 512MB swap in a swap file at `/swapfile`
- Now we want to change the configuration using ansible variables
- Go to Configure → Ansible → Variables → Click on the variable `swap_file_size_mb`
    - Tick the checkbox "Override", some new configuration options appear
    - We want to create a matcher such that the value will only be set for our host group
        - Click on Add Matcher
        - for Attribute Type select `hostgroup` = `alma9/test`
        - As value we use `1024`
        - Click submit to save the changes
- Run the configured roles on the host (again):
    - In the Hosts → All Hosts overview, click the checkbox next to the `rhino1` host
    - Click on “Select Action” on the top right and then on the drop-down arrow next to Schedule a job → select "Run all Ansible roles"
- Check the output of the job as before, and optionally run a job with the command `swapon -s` again to confirm that the size of the swapfile really changed

User Management

An orcharhino is usually a very central piece of infrastructure. A user with admin privileges can perform a wide range of changes to the infrastructure. Especially from a security standpoint, the following topics are important and will be addressed as exercises:

1. Add a users with restricted permissions
2. Audit changes made in orcharhino
3. manage login credentials (use external authentication service such as LDAP, add 2FA)

We will start with a short overview and address these topics afterwards. For 1. and 2. we will also have a short exercise, while an exercise for 3. is beyond the scope of our test system.

Overview

We can add users either locally on the orcharhino (=**internal)** or via using some authentication source like LDAP (=**LDAP**), or via some other external service (= **external**). External authentication also makes it possible to add things like 2FA to orcharhino.

Permissions in orcharhino are bundled in so-called **roles**. You never assign permissions directly, as this would be error-prone, instead you use (pre- or user-defined) roles. Users can be added to ***groups***, which makes it easy to adjust the roles for multiple users in one place. orcharhino offers many pre-defined roles for common use cases, which can be assigned either to the user individually or inherited from groups that the user is assigned to.

Ideally, users log into orcharhino via their own account (not a generic admin account), which can be nicely managed via external authentication. A **audit log** is available in the orcharhino UI that makes it easy to determine which action was issues by which user, which helps understanding the chain of events e.g. in case of an accidental misconfiguration.

For more in-depth information, you can have a look at our official documentation:

- Managing Users and Roles:
  Administering orcharhino - orcharhino documentation
- Configuring external and LDAP authentication:
  Configuring authentication for orcharhino users - orcharhino documentation
- Maintaining audit logs (more technical, only if you want to find out how to clean up audit logs):
  Administering orcharhino - orcharhino documentation

Hands-On: Add a user with restricted permissions

Add a restricted user:

- Administer → Users → Click on Create User
- Fill: Username→ "testuser", Authorized by: "INTERNAL", Password: "testuser", Verify: "testuser"
- Make sure Location and Organization are set in the corresponding tabs
- In the Roles tab, select "View hosts" and "Remote Execution user"
- Click Submit

to see which permissions we just granted to our user in detail, do the following:

- Click Administer → Roles → Click on the "Filters" button for the "Remote Execution User" entry
- We can observe that several of permissions are needed for allowing remote execution, especially also the "view_hosts" permission, so our initial role selection was indeed redundant

(the same permission is also present in the "View hosts" role)

Now we verify that the permissions are indeed restricted for our test user. Log in as "testuser" (password "testuser"). Alternatively you can also impersonate the user by clicking the pull-down menu in the Actions column of the user list (Administer → Users) and selecting “Impersonate”. You can exit the impersonation again by clicking on the eye symbol next to the user name on the top right.
Now as testuser try the following:

- Click on Hosts
- Observe that "Create Hosts" is not shown
- Click on “View Hosts”: host list should be visible
- Tick the checkbox next to the Ubuntu 24.04 host and then click "Select Action"
- Observe that the dropdown menu does not show Delete Hosts as it would be the case for the admin user
- Schedule Remote Job is still available for the testuser, click it
- Fill: Job category: "Packages"; Job template: "Package Action - Script Default"
- Click Next; then action: install; package: "fonts-noto-mono"
- Click "run on selected hosts"
- Watch the job overview, it should show 100% after a short time
- Click on the host name in the bottom to observe the job output; package should have been installed
- Verify that the package is installed: click on the host name at the top off the command output listing (where it says `Target: <hostname>`)
- Click on the Content Tab → Packages Tab
- Search for the package that was just installed, there should be a search result for that package and the "Installed version" column listing the exact version
- Optional: run a job that removes this package and verify that it was indeed removed

Hands-On: Inspect Audit logs

Assuming you just performed previous hands-on exercise, log in back as admin and have a look at the
Monitor→Audits section in the orcharhino UI. You should see some audit logs for the actions you performed in the previous exercise, marked as performed by that specific user.

Now we perform some action ourselves and see whether it ends up in the audits list. Go to Content→Sync Status and trigger a synchronization for an arbitrary repository in an arbitrary

product (check one of the checkboxes and press “Synchronize Now”. If you go to the audit log, you will see that this particular action has not been logged in the audit log. This is because you triggered an **orcharhino task**, which is tracked under Monitor→orcharhino Tasks→Tasks. As it is a normal mode of operation to synchronize repositories and does not change the configuration of orcharhino or the hosts itself, it is not logged into the audits.

Now we perform some action that will go into the audit log. Go to Administer→Settings and change the setting “Entries per page” in the “General” tab to a different value, e.g. 30.

Go back to Monitors→Audits and observe that a new entry logging this activity has been created in the audit log. We can also filter for similar events. For this type the following in the search bar:

```
action = create and type = setting
```

While typing, you should have noticed that there are auto-completion hints and suggestions for things you might want to filter for. In the filtered view you should only see entries of the audit log that are about changes of settings.