

Contents

[Introduction 3](#_Toc156389980)

[Prerequisites 3](#_Toc156389981)

[Overview of k3s\_cluster\_install.yaml Playbook 4](#_Toc156389982)

[K3s Master Installation 4](#_Toc156389983)

[Additional Playbook Imports 4](#_Toc156389984)

[Overall Functionality 4](#_Toc156389985)

[Benefits 5](#_Toc156389986)

[Ansible Playbook for Harbor Deployment 6](#_Toc156389987)

[Explanation of the harbor\_deployment.yaml Ansible Playbook 7](#_Toc156389988)

[Overall Functionality 8](#_Toc156389989)

[Preparing for Playbook Execution with Required Files 9](#_Toc156389990)

[Preparing for the k3s\_cluster\_install.yaml Playbook Execution 10](#_Toc156389991)

[Explanation of the helm\_install.yaml Playbook 12](#_Toc156389992)

[Explanation of the private-registry.yaml Ansible Playbook 14](#_Toc156389993)

[Essential Resources and Download Links 16](#_Toc156389994)

# Introduction

**Kubernetes and K3s**

* **Kubernetes:** An open-source platform designed for automating deployment, scaling, and operations of application containers across clusters of hosts. It simplifies the process of managing containerized applications, making it efficient and scalable.
* **K3s:** A lightweight, easy-to-install Kubernetes distribution, perfect for edge computing, IoT (Internet of Things), and CI/CD (Continuous Integration and Continuous Deployment) environments. K3s offers the essential features of Kubernetes with a smaller footprint.

**Harbor**

* **Overview:** Harbor is an advanced, open-source container image registry that provides secure storage and management of container images. It integrates security practices by enforcing policies and role-based access control, ensuring that container images are scanned for vulnerabilities and are secure for deployment.

**Ansible**

* **Functionality:** Ansible is a powerful open-source automation tool used for software provisioning, configuration management, and application deployment. It is known for its simplicity and ease of use, employing YAML syntax for defining automation tasks.

**Purpose of this Document**

* **Objective:** This document serves as a comprehensive guide for deploying Harbor on a K3s cluster using Ansible. It is crafted to provide clear, step-by-step instructions suitable for users with various levels of experience in these technologies.

## Prerequisites

**Software and Tools**

1. **Ansible:** Essential for automating the deployment process.
2. **K3s:** Selected as the Kubernetes distribution for its lightweight nature.
3. **Helm:** A tool for managing Kubernetes applications, easing the process of installing and updating software in the Kubernetes environment.
4. **Access to a Kubernetes Cluster:** Either a local setup or a cloud-based solution, depending on your deployment needs.

**Setting Up the Environment**

Initial Setup Steps

1. **Install and configure Ansible:** Begin by following the comprehensive Ansible Installation Guide to set up Ansible on your system.
2. **Make sure you have the required files.**

# Overview of k3s\_cluster\_install.yaml Playbook

This Ansible playbook is crafted for the comprehensive setup of a K3s Kubernetes cluster. The setup includes the master node, worker nodes, and essential components like Helm, a private Harbor registry, and firewalld configurations.

## K3s Master Installation

* **Target Hosts:** Focuses on the master node(s) of the cluster.
* **Key Tasks:**
  + **Creating Image Directory:** Ensures **/var/lib/rancher/k3s/agent/images/** exists with the correct permissions for image storage.
  + **Setting K3s Executable:** Places the K3s binary in **/usr/local/bin/** and sets it as executable.
  + **Transferring Air-Gap Images:** Moves a tarball of K3s air-gap images to a designated directory for offline usage.
  + **Install Script Configuration:** Adds the K3s installation script to the home directory and makes it executable.
  + **Handling K3s SELinux File:** Transfers the K3s SELinux RPM package for installation.
  + **SELinux Package Installation:** Installs the K3s SELinux package using Yum.
  + **Script Execution:** Executes the K3s installation script, omitting the download step.
  + **Node Token Retrieval and Storage:** Fetches the node token from the K3s server and stores it for later use.

## Additional Playbook Imports

* **Worker Node Setup:** Executes **K3s\_worker\_install.yaml** to configure worker nodes.
* **Firewalld Configuration:** Applies network security rules via **firewalld.yaml**.
* **Helm Setup on Master Node:** Installs and configures Helm using **helm\_install.yaml**.
* **Harbor Private Registry Configuration:** Sets up Harbor as a private Docker registry with **private-registry.yaml**.
* **Harbor Deployment:** Facilitates the deployment of the Harbor registry using **harbor\_local\_deploy.yaml**.

## Overall Functionality

* The playbook begins by preparing the master node with essential binaries, images, and installation scripts.
* It then proceeds to set up the worker nodes, implement firewalld for secure network traffic, and install Helm for efficient Kubernetes package management.
* Finally, it concludes with the setup and deployment of Harbor, creating a private Docker image registry within the cluster.

## Benefits

* **Automation Efficiency:** This playbook streamlines the process of establishing a fully functional Kubernetes environment using K3s, ensuring all components are correctly installed and configured.
* **Error Minimization:** Particularly beneficial in scenarios where manual configuration is not only time-consuming but also prone to errors.

# Ansible Playbook for Harbor Deployment

**Loading Harbor Images**

1. **Directory Creation:** Establish directories on each node for storing Harbor images.
2. **Copying Resources:** Transfer the Harbor Helm chart and related container images to the appropriate directories.
3. **Image Loading:** Utilize K3s's **ctr** command to load the images into the Kubernetes cluster.

**Deploying Harbor with Helm**

* **Installation:** Perform the installation of Harbor on the master node of the Kubernetes cluster using Helm, a Kubernetes package manager that simplifies software deployment.

**Executing the Playbooks**

1. **First Playbook Execution:** Run the playbook designed to load Harbor images onto all nodes within the cluster.
2. **Second Playbook Execution:** Execute the playbook responsible for installing Harbor on the master node using Helm.

**Verification and Troubleshooting**

* **Image Verification:** Use **k3s ctr images list** to confirm that the Harbor images have been successfully loaded onto the nodes.
* **Harbor Installation Check:** Verify the successful deployment of Harbor by executing relevant Helm commands.

# Explanation of the harbor\_deployment.yaml Ansible Playbook

The **harbor\_deployment.yaml** playbook is designed to deploy Harbor, a cloud-native container registry, on a K3s Kubernetes cluster using Ansible. The playbook consists of two main parts: loading Harbor images and installing Harbor using Helm. Here's a breakdown of its functionality:

**Part 1: Load Harbor Images Using K3s ctr**

* **Hosts:** All nodes in the K3s cluster.
* **Tasks:**
  1. **Create Directory for Harbor Images:** Makes a directory at **/home/harbor\_images** on each node for storing Harbor container images.
  2. **Copy Helm Chart:** Unarchives the Harbor Helm chart (**harbor-1.13.1.tgz**) to **/home/** on all nodes.
  3. **Copy Mirrored Pause Image:** Transfers the **mirrored-pause.tar** image file to **/home/** on all nodes. This image is typically used in Kubernetes as a placeholder for maintaining network namespaces.
  4. **Copy Harbor Images:** Unarchives a tarball containing Harbor container images to **/home/** on all nodes.
  5. **Define Harbor Images List:** Sets a list of Harbor-related image names as an Ansible fact for later use.
  6. **Load Harbor Images:** Uses K3s's container runtime (**ctr**) to import each listed Harbor image. The images are imported from the previously created directory.

**Part 2: Execute Helm Chart**

* **Hosts:** Only the master node of the K3s cluster.
* **Tasks:**
  1. **Install Harbor Using Helm:** Executes a Helm command to install the Harbor application on the Kubernetes cluster. The command uses the **--generate-name** flag to automatically generate a release name for Harbor and specifies the K3s kubeconfig file for Kubernetes cluster access.

## Overall Functionality

* The playbook first ensures that all necessary Harbor images and the Helm chart are available on all nodes in the cluster. This step is crucial, especially in air-gapped environments where direct internet access is not available.
* It then proceeds to load these images into the container runtime of each node, preparing them for the actual deployment of Harbor.
* Finally, Harbor is installed on the Kubernetes cluster (specifically on the master node) using Helm, a package manager for Kubernetes, which simplifies the deployment and management of applications like Harbor.

By automating these steps with Ansible, the playbook provides a streamlined and repeatable process for deploying Harbor in a K3s environment. This approach is particularly beneficial for environments where manual deployment would be complex and error-prone.

# Explanation of the monitoring.yaml Ansible Playbook

The **monitoring.yaml** playbook uses Ansible to setup and deploy the applications required to monitor the Kubernetes cluster. This includes Prometheus, along with its add-ons, and Grafana. The persistent volumes required for storing monitored data are also part of this setup.

# Installation

**Host:** Masternode of the k3s cluster

* **Host:** Master node of the k3s cluster.
* **Tasks:**
  + **Copy YAML Files To Master:** Copies over the YAML files to the master node and puts them in the **/home/** folder.
  + **Create Monitoring Namespace:** Creates the monitoring namespace to separate the monitoring apps from the rest of the Kubernetes cluster. **Note: If the “monitoring” namespace already exists, this step will fail and cause the process to stop.**
  + **Deploying Components:** Deploys the monitoring applications on the k3s cluster using kubectl apply.

After executing this playbook the monitoring applications on the k3s cluster are set up and ready to be configured further according to the user’s needs.

# Grafana Dashboard Setup

After the monitoring.yaml playbook is used to setup the Grafana application, the dashboard must be configured inside of the Grafana application itself, the process of which is described in this setup. In this setup we will be importing the dashboard from the included grafana-dashboard.json file.

# Instructions

1. **Access the Grafana Application:** First to access the Grafana application you must enter the IP of your cluster’s master node in your browser followed by the port configured for Grafana (Default: **32000**).
2. **Authenticate User:** After this you have to log into the application using the username and password which are both **“admin”**. It is highly recommended to immediately change the password to something more secure using the following prompt.
3. **Access the Dashboards Menu:** Once you see the Home screen of Grafana you must access the side menu by clicking the hamburger button on the left hand side of the screen. In this side menu you must then click on **“Dashboards”** to access the dashboards menu.
4. **Head to the Import Menu:** In the dashboards menu you can head to the import menu by clicking on the “New” button to open a dropdown menu. In this menu you can find the Import button.
5. **Import the JSON file:** In the import menu you can now import the **grafana-dashboard.json** file. This can be done in one of two ways:
   1. Upload the file directly to the Grafana application.
   2. Copy and paste the contents of the file to the textbox shown in the import menu.

Both ways will have the same results, but one might be more convenient than the other depending on your situation.

1. **Finalize import:** The menu that should follow from the previous step will show you some options for the dashboard. In here you must select the Prometheus data source, so that the dashboard knows where to pull the data from. You can also change the name of the dashboard if desired.

After following these steps you should have a functioning dashboard in Grafana. On the dashboard screen you can change the time range of the data by using the clock button in the top right corner.

## Dashboard Functionality

* **Display of CPU Usage:** The dashboard shows the viewer a detailed breakdown of the CPU usage in multiple different categories: Per namespace, per deployment, per pod, over time and the total current usage. This should give the viewer a better grasp of potential problems that might be occurring in their cluster and what segments could be causing them.
* **Display of RAM Usage:** Along with the CPU usage this dashboard also shows the RAM usage in the same way. This might also be helpful in determining where problems could be coming from as some issues and/or slowdown might not be apparent from CPU Usage statistics alone.

## Preparing for Playbook Execution with Required Files

**Required Files and Their Sources**

Before executing the playbook, ensure that the following essential files are downloaded and available in the specified source locations:

1. **Harbor Helm Chart (harbor-1.13.1.tgz):**
   * **Description:** A packaged Helm chart for Harbor. Helm charts are collections of pre-configured Kubernetes resources that facilitate the deployment of applications like Harbor.
   * **Download Source:** Acquire the Harbor Helm chart from the official Harbor releases page on GitHub. Be sure to download version 1.13.1 or the version specified in your playbook.
2. **Rancher/Mirrored-Pause Image (mirrored-pause.tar):**
   * **Description:** A container image used by Kubernetes as a placeholder for maintaining network namespaces during various operations.
   * **Download Source:** This image is usually available from container image registries like Docker Hub. If a specific version aligned with Rancher is needed, it may be found in Rancher repositories or documentation.
3. **Harbor Container Images (harbor-images.tar):**
   * **Description:** The Docker container images for the Harbor application, essential for running Harbor on Kubernetes nodes.
   * **Download Source:** Manually pull all necessary images from Docker Hub and package them accordingly.
4. **Monitoring Applications Container Images:**
   * **Description:** The Docker container images for the various applications relating to monitoring of the Kubernetes cluster.
   * **Download Source:** As with the Harbor images, manually pull all the different necessary images and package them accordingly.

**Preparation Steps**

* **File Organization:** Download the required files and store them in a shared directory, such as **/share/harbor/**, ensuring it's accessible by the Ansible control machine.
* **Playbook Update:** Modify the playbook to reflect the correct file paths, especially if your files are stored in a location different from the default paths.

**Executing the Playbook**

With the prerequisites met and the necessary files in place, you're now set to run the playbook. The playbook will manage the creation of directories and handle the copying and unpacking of these files to their designated locations on your K3s nodes.

## Preparing for the k3s\_cluster\_install.yaml Playbook Execution

**Required Files and Their Sources**

To ensure a smooth execution of the k3s\_cluster\_install.yaml playbook, certain files must be pre-downloaded and available. These include:

1. **K3s Binary (k3s):**
   * **Description:** The executable binary for K3s, essential for running the Kubernetes cluster.
   * **Download Source:** Obtain the binary from the official K3s GitHub repository. Select the version that aligns with your deployment needs.
2. **K3s Air-Gap Images (k3s-airgap-images-amd64.tar):**
   * **Description:** Container images required by K3s, especially packaged for environments without internet access (air-gapped).
   * **Download Source:** Available on the K3s releases page on GitHub. Choose the images corresponding to your specific K3s version.
3. **K3s Install Script (install.sh):**
   * **Description:** A convenient script provided by K3s for streamlined installation.
   * **Download Source:** Typically found in the K3s GitHub repository, alongside the binary and air-gap images.
4. **K3s SELinux RPM Package (k3s-selinux-1.4-1.el9.noarch.rpm):**
   * **Description:** An SELinux policy package tailored for K3s.
   * **Download Source:** Can be sourced from the K3s repository or other websites hosting K3s-related files. Ensure compatibility with your K3s version.

**Preparation Steps**

* **File Placement:** Post-download, place these files in a shared directory, such as /share/k3s/, where they are accessible to the Ansible control machine.
* **Path Verification:** Confirm that the paths specified in the playbook (src attributes) match the locations of the stored files.

**Additional Playbooks**

The k3s\_cluster\_install.yaml playbook integrates several additional playbooks, which must also be accessible:

* **K3s\_worker\_install.yaml**
* **firewalld.yaml**
* **helm\_install.yaml**
* **private-registry.yaml**
* **harbor\_local\_deploy.yaml**

Make sure these playbooks are in your playbook directory or adjust the paths in the main playbook to correspond with their locations.

**Execution Readiness**

With all prerequisites and files correctly set up, you're ready to execute the k3s\_cluster\_install.yaml playbook. This will automate the configuration of the K3s on your master node, the setup of worker nodes, and the installation of essential components like Helm and Harbor, creating a robust and efficient Kubernetes environment.

# Explanation of the helm\_install.yaml Playbook

This playbook is specifically designed for installing Helm on the master node of a Kubernetes cluster managed by K3s. Helm, a key tool for managing Kubernetes applications, is essential for deploying and managing apps efficiently in a Kubernetes environment.

**Target and Tasks:**

1. **Hosts:**
   * The playbook specifically targets the master node(s) of the K3s cluster.
2. **Key Tasks:**
   * **Copy Helm Binary:**
     + **Objective:** Transfers the Helm binary to **/usr/local/bin/** on the master node, a common directory in the system's PATH. This placement makes the Helm command readily accessible.
     + **Requirements:** The Helm binary needs to be located at **/share/helm/linux-amd64/helm** on the Ansible control machine or an equivalent shared location.
     + **Permissions:** Sets the binary's file mode to 0755, ensuring it is executable.
   * **Export Kubeconfig:**
     + **Purpose:** Executes a command to set the **KUBECONFIG** environment variable, guiding Helm and other Kubernetes tools to the Kubernetes API configuration file.
     + **Configuration:** Sets **KUBECONFIG** to **/etc/rancher/k3s/k3s.yaml**, the default kubeconfig file location for K3s.
     + **Consideration:** This task configures the environment variable temporarily during task execution. For persistent changes, consider adding the export command to the user's profile or bashrc file.

**Overall Functionality**

* The playbook is primarily designed to facilitate a straightforward installation of Helm on the master node.
* It ensures the Helm binary is appropriately placed and configured to interact seamlessly with the K3s cluster.

**Additional Notes**

* **Prerequisites:** Verify the specified source path for the Helm binary (**/share/helm/linux-amd64/helm**) and ensure the binary is present before executing the playbook.
* **Verification:** Post-execution, validate the installation by running **helm version** on the master node.
* **Scope:** This playbook focuses exclusively on installing Helm, without importing additional playbooks or roles.

**Key Benefit**

By automating the Helm setup, this playbook significantly streamlines the deployment and management process of Kubernetes applications, enhancing the overall efficiency of operating within a Kubernetes cluster.

# Explanation of the private-registry.yaml Ansible Playbook

The **private-registry.yaml** playbook is tailored for integrating a private registry within a K3s Kubernetes environment. It ensures all nodes are configured to utilize this registry, crucial for environments with limited internet access or specific in-house container image requirements.

K3s Private Registry Configuration

1. **Targets (Hosts):**
   * Applies to all nodes within the K3s cluster.
2. **Primary Tasks:**
   * **Create K3s Configuration Directory:**
     + Ensures **/etc/rancher/k3s/** exists on all nodes for storing K3s configuration files.
     + Sets directory mode to 0755 for appropriate access permissions.
   * **Create Registries Configuration File:**
     + Generates a **registries.yaml** file within **/etc/rancher/k3s/**. This file is pivotal for configuring the private registry within K3s.
     + Configuration details:
       - **Mirrors:** Defines the URL of the private registry (e.g., **harbor.core.local**).
       - **Configs:** Establishes TLS configurations, including **insecure\_skip\_verify: true** for handling self-signed certificates or non-HTTPS registries.

Restarting K3s Services

1. **For Master Node:**
   * **Scope:** Targets exclusively the master node(s).
   * **Tasks:** Restarts K3s service to enforce new registry configurations, ensuring the daemon reloads with updated settings.
2. **For Worker Nodes:**
   * **Scope:** Focuses solely on the worker nodes.
   * **Tasks:** Restarts K3s-agent service to apply registry configuration changes, mirroring the master node's process.

Overall Functionality

* **Uniformity:** Standardizes container registry settings across all nodes, allowing each to pull images from the designated private registry.
* **Service Restart Importance:** Vital to ensure all changes are applied effectively.

Additional Considerations

* **Preparation:** Confirm the private registry (like Harbor) is operational and accessible at the specified URL (**harbor.core.local**).
* **Customization:** Tailor the settings in **registries.yaml** to align with your specific environmental and registry configurations.
* **Verification:** Post-execution, ensure K3s nodes can successfully pull images from the private registry.

Key Benefit

This playbook streamlines the integration of a private registry with a K3s Kubernetes cluster, enhancing the security and management efficiency of container images within your Kubernetes setup.

# Essential Resources and Download Links

**K3s Resources and Binaries**

* K3s Documentation: <https://ranXcher.com/docs/k3s/latest/en/>
* K3s GitHub Repository: <https://github.com/k3s-io/k3s>
* K3s Installation Guide: <https://rancher.com/docs/k3s/latest/en/installation/>

**Helm Resources and Binaries**

* Helm Documentation: <https://helm.sh/docs/>
* Helm GitHub Repository: <https://github.com/helm/helm/releases>

**Harbor Resources and Binaries**

* Harbor Documentation: <https://goharbor.io/docs/>
* Harbor GitHub Repository: <https://github.com/goharbor/harbor/releases>

**Rancher/Mirrored-Pause Image**

* Docker Hub: <https://hub.docker.com/>

**General Kubernetes Resources**

* Kubernetes Documentation: <https://kubernetes.io/docs/>