



## Red Hat Quay 3.8

# Vulnerability reporting with Clair on Red Hat Quay

Vulnerability reporting with Clair on Red Hat Quay



# Red Hat Quay 3.8 Vulnerability reporting with Clair on Red Hat Quay

---

Vulnerability reporting with Clair on Red Hat Quay

## Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

Get started with Red Hat Quay

# Table of Contents

|  |           |
|--|-----------|
| <b>PREFACE</b> .....   | <b>4</b>  |
| <b>PART I. VULNERABILITY REPORTING WITH CLAIR ON RED HAT QUAY OVERVIEW</b> .....           | <b>5</b>  |
| <b>CHAPTER 1. CLAIR FOR RED HAT QUAY</b> .....   | <b>6</b>  |
| 1.1. CLAIR VULNERABILITY DATABASES .....   | 6         |
| <b>CHAPTER 2. CLAIR CONCEPTS</b> .....   | <b>7</b>  |
| 2.1. CLAIR IN PRACTICE .....   | 7         |
| 2.1.1. Indexing .....  | 7         |
| 2.1.1.1. Content addressability .....  | 7         |
| 2.1.2. Matching .....  | 7         |
| 2.1.2.1. Remote matching .....   | 8         |
| 2.1.3. Notifications .....   | 8         |
| 2.1.3.1. Webhook delivery .....  | 9         |
| 2.1.3.2. AMQP delivery .....   | 9         |
| 2.1.3.2.1. AMQP direct delivery .....  | 9         |
| 2.1.3.3. Notifier testing and development mode .....                                       | 9         |
| 2.1.3.4. Deleting notifications .....  | 10        |
| 2.2. CLAIR AUTHENTICATION .....  | 10        |
| 2.3. CLAIR UPDATERS .....  | 10        |
| 2.3.1. Configuring updaters .....  | 10        |
| 2.3.1.1. Updater sets .....  | 11        |
| 2.3.1.2. Selecting updater sets .....  | 11        |
| 2.3.1.3. Filtering updater sets .....  | 11        |
| 2.3.1.4. Configuring specific updaters .....   | 12        |
| 2.3.1.5. Disabling the Clair Updater component .....                                       | 12        |
| 2.3.2. Clair updater URLs .....  | 12        |
| <b>CHAPTER 3. ABOUT CLAIR</b> .....  | <b>13</b> |
| 3.1. CLAIR RELEASES .....  | 13        |
| 3.2. CLAIR SUPPORTED LANGUAGES .....   | 13        |
| 3.3. CLAIR CONTAINERS .....  | 13        |
| 3.4. CVE RATINGS FROM THE NATIONAL VULNERABILITY DATABASE .....                            | 13        |
| 3.5. FEDERAL INFORMATION PROCESSING STANDARD (FIPS) READINESS AND COMPLIANCE .....         | 13        |
| <b>PART II. CLAIR ON RED HAT QUAY</b> .....  | <b>15</b> |
| <b>CHAPTER 4. SETTING UP CLAIR ON STANDALONE RED HAT QUAY DEPLOYMENTS</b> .....            | <b>16</b> |
| <b>CHAPTER 5. CLAIR ON OPENSIFT CONTAINER PLATFORM</b> .....                               | <b>19</b> |
| <b>CHAPTER 6. TESTING CLAIR</b> .....  | <b>20</b> |
| <b>PART III. ADVANCED CLAIR CONFIGURATION</b> .....  | <b>22</b> |
| <b>CHAPTER 7. UNMANAGED CLAIR CONFIGURATION</b> .....                                      | <b>23</b> |
| 7.1. RUNNING A CUSTOM CLAIR CONFIGURATION WITH AN UNMANAGED CLAIR DATABASE .....           | 23        |
| 7.2. CONFIGURING A CUSTOM CLAIR DATABASE WITH AN UNMANAGED CLAIR DATABASE .....            | 23        |
| <b>CHAPTER 8. RUNNING A CUSTOM CLAIR CONFIGURATION WITH A MANAGED CLAIR DATABASE</b> ..... | <b>26</b> |
| 8.1. SETTING A CLAIR DATABASE TO MANAGED .....   | 26        |
| 8.2. CONFIGURING A CUSTOM CLAIR DATABASE WITH A MANAGED CLAIR CONFIGURATION .....          | 26        |
| <b>CHAPTER 9. CLAIR IN DISCONNECTED ENVIRONMENTS</b> .....                                 | <b>29</b> |

---

|  |           |
|--|-----------|
| 9.1. SETTING UP CLAIR IN A DISCONNECTED OPENSIFT CONTAINER PLATFORM CLUSTER  | 29        |
| 9.1.1. Installing the clairctl command line utility tool for OpenShift Container Platform deployments                        | 29        |
| 9.1.2. Retrieving and decoding the Clair configuration secret for Clair deployments on OpenShift Container Platform          | 29        |
| 9.1.3. Exporting the updaters bundle from a connected Clair instance   | 30        |
| 9.1.4. Configuring access to the Clair database in the disconnected OpenShift Container Platform cluster                     | 30        |
| 9.1.5. Importing the updaters bundle into the disconnected OpenShift Container Platform cluster                              | 32        |
| 9.2. SETTING UP A SELF-MANAGED DEPLOYMENT OF CLAIR FOR A DISCONNECTED OPENSIFT CONTAINER PLATFORM CLUSTER                    | 32        |
| 9.2.1. Installing the clairctl command line utility tool for a self-managed Clair deployment on OpenShift Container Platform | 32        |
| 9.2.2. Deploying a self-managed Clair container for disconnected OpenShift Container Platform clusters                       | 32        |
| 9.2.3. Exporting the updaters bundle from a connected Clair instance   | 33        |
| 9.2.4. Configuring access to the Clair database in the disconnected OpenShift Container Platform cluster                     | 34        |
| 9.2.5. Importing the updaters bundle into the disconnected OpenShift Container Platform cluster                              | 35        |
| 9.3. ENABLING CLAIR CRDA   | 35        |
| 9.4. MAPPING REPOSITORIES TO COMMON PRODUCT ENUMERATION INFORMATION  | 36        |
| 9.4.1. Mapping repositories to Common Product Enumeration example configuration  | 36        |
| <b>CHAPTER 10. CLAIR CONFIGURATION OVERVIEW</b> .....  | <b>37</b> |
| 10.1. CLAIR CONFIGURATION REFERENCE  | 37        |
| 10.2. CLAIR GENERAL FIELDS   | 38        |
| 10.3. CLAIR INDEXER CONFIGURATION FIELDS   | 39        |
| 10.4. CLAIR MATCHER CONFIGURATION FIELDS   | 40        |
| 10.5. CLAIR MATCHERS CONFIGURATION FIELDS  | 42        |
| 10.6. CLAIR UPDATERS CONFIGURATION FIELDS  | 43        |
| 10.7. CLAIR NOTIFIER CONFIGURATION FIELDS  | 44        |
| 10.8. CLAIR AUTHORIZATION CONFIGURATION FIELDS   | 48        |
| 10.9. CLAIR TRACE CONFIGURATION FIELDS   | 49        |
| 10.10. CLAIR METRICS CONFIGURATION FIELDS  | 50        |



## PREFACE

The contents within this guide provide an overview of Clair for Red Hat Quay, running Clair on standalone Red Hat Quay and Operator deployments, and advanced Clair configuration.



## **PART I. VULNERABILITY REPORTING WITH CLAIR ON RED HAT QUAY OVERVIEW**

The content in this guide explains the key purposes and concepts of Clair on Red Hat Quay. It also contains information about Clair releases and the location of official Clair containers.

## CHAPTER 1. CLAIR FOR RED HAT QUAY

Clair v4 (Clair) is an open source application that leverages static code analyses for parsing image content and reporting vulnerabilities affecting the content. Clair is packaged with Red Hat Quay and can be used in both standalone and Operator deployments. It can be run in highly scalable configurations, where components can be scaled separately as appropriate for enterprise environments.

### 1.1. CLAIR VULNERABILITY DATABASES

Clair uses the following vulnerability databases to report for issues in your images:

- Ubuntu Oval database
- Debian Oval database
- Red Hat Enterprise Linux (RHEL) Oval database
- SUSE Oval database
- Oracle Oval database
- Alpine SecDB database
- VMWare Photon OS database
- Amazon Web Services (AWS) UpdateInfo
- Pyup.io (Python) database

For information about how Clair does security mapping with the different databases, see [ClairCore Severity Mapping](#).

---

## CHAPTER 2. CLAIR CONCEPTS

The following sections provide a conceptual overview of how Clair works.

### 2.1. CLAIR IN PRACTICE

A Clair analysis is broken down into three distinct parts: indexing, matching, and notification.

#### 2.1.1. Indexing

Clair's indexer service is responsible for indexing a manifest. In Clair, manifests are representations of a container image. The indexer service is the component that Clair uses to understand the contents of layers. Clair leverages the fact that Open Container Initiative (OCI) manifests and layers are content addressed to reduce duplicate work.

Indexing involves taking a manifest representing a container image and computing its constituent parts. The indexer tries to discover what packages exist in the image, what distribution the image is derived from, and what package repositories are used within the image. When this information is computed, it is persisted into an **IndexReport**.

The **IndexReport** is stored in Clair's database. It can be fed to a **matcher** node to compute the vulnerability report.

##### 2.1.1.1. Content addressability

Clair treats all manifests and layers as *content addressable*. In the context of Clair, content addressable means that when a specific manifest is indexed, it is not indexed again unless it is required; this is the same for individual layers.

For example, consider how many images in a registry might use **ubuntu:artful** as a base layer. If the developers prefer basing their images off of Ubuntu, it could be a large majority of images. Treating the layers and manifests as content addressable means that Clair only fetches and analyzes the base layer one time.

In some cases, Clair should re-index a manifest. For example, when an internal component such as a package scanner is updated, Clair performs the analysis with the new package scanner. Clair has enough information to determine that a component has changed and that the **IndexReport** might be different the second time, and as a result it re-indexes the manifest.

A client can track Clair's **index\_state** endpoint to understand when an internal component has changed, and can subsequently issue re-indexes. See the Clair API guide to learn how to view Clair's API specification.

#### 2.1.2. Matching

With Clair, a matcher node is responsible for matching vulnerabilities to a provided **IndexReport**.

Matchers are responsible for keeping the database of vulnerabilities up to date. Matchers will typically run a set of updaters, which periodically probe their data sources for new content. New vulnerabilities are stored in the database when they are discovered.

The matcher API is designed to be used often. It is designed to always provide the most recent **VulnerabilityReport** when queried. The **VulnerabilityReport** summarizes both a manifest's content and any vulnerabilities affecting the content.

### 2.1.2.1. Remote matching

A remote matcher acts similar to a matcher, however remote matchers use API calls to fetch vulnerability data for a provided **IndexReport**. Remote matchers are useful when it is impossible to persist data from a given source into the database.

The CRDA remote matcher is responsible for fetching vulnerabilities from Red Hat Code Ready Dependency Analytics (CRDA). By default, this matcher serves 100 requests per minute. The rate limiting can be lifted by requesting a dedicated API key, which is done by submitting [the API key request form](#).

To enable CRDA remote matching, see "Enabling CRDA for Clair".

### 2.1.3. Notifications

Clair uses a notifier service that keeps track of new security database updates and informs users if new or removed vulnerabilities affect an indexed manifest.

When the notifier becomes aware of new vulnerabilities affecting a previously indexed manifest, it uses the configured methods in your **config.yaml** file to issue notifications about the new changes. Returned notifications express the most severe vulnerability discovered because of the change. This avoids creating excessive notifications for the same security database update.

When a user receives a notification, it issues a new request against the matcher to receive an up to date vulnerability report.

The notification schema is the JSON marshalled form of the following types:

```
// Reason indicates the catalyst for a notification
type Reason string
const (
    Added Reason = "added"
    Removed Reason = "removed"
    Changed Reason = "changed"
)
type Notification struct {
    ID          uuid.UUID      `json:"id"`
    Manifest    claircore.Digest `json:"manifest"`
    Reason      Reason          `json:"reason"`
    Vulnerability VulnSummary    `json:"vulnerability"`
}
type VulnSummary struct {
    Name          string          `json:"name"`
    Description    string          `json:"description"`
    Package       *claircore.Package `json:"package,omitempty"`
    Distribution   *claircore.Distribution `json:"distribution,omitempty"`
    Repo          *claircore.Repository `json:"repo,omitempty"`
    Severity      string          `json:"severity"`
    FixedInVersion string          `json:"fixed_in_version"`
    Links         string          `json:"links"`
}
```

You can subscribe to notifications through the following mechanics:

- Webhook delivery

- AMQP delivery
- STOMP delivery

Configuring the notifier is done through the Clair YAML configuration file.

### 2.1.3.1. Webhook delivery

When you configure the notifier for webhook delivery, you provide the service with the following pieces of information:

- A target URL where the webhook will fire.
- The callback URL where the notifier might be reached, including its API path. For example, <http://clair-notifier/notifier/api/v1/notifications>.

When the notifier has determined an updated security database has been changed the affected status of an indexed manifest, it delivers the following JSON body to the configured target:

```
{
  "notification_id": {uuid_string},
  "callback": {url_to_notifications}
}
```

On receipt, the server can browse to the URL provided in the callback field.

### 2.1.3.2. AMQP delivery

The Clair notifier also supports delivering notifications to an AMQP broker. With AMQP delivery, you can control whether a callback is delivered to the broker or whether notifications are directly delivered to the queue. This allows the developer of the AMQP consumer to determine the logic of notification processing.



#### NOTE

AMQP delivery only supports AMQP 0.x protocol (for example, RabbitMQ). If you need to publish notifications to AMQP 1.x message queue (for example, ActiveMQ), you can use STOMP delivery.

#### 2.1.3.2.1. AMQP direct delivery

If the Clair notifier's configuration specifies **direct: true** for AMQP delivery, notifications are delivered directly to the configured exchange.

When **direct** is set, the **rollup** property might be set to instruct the notifier to send a maximum number of notifications in a single AMQP. This provides balance between the size of the message and the number of messages delivered to the queue.

### 2.1.3.3. Notifier testing and development mode

The notifier has a testing and development mode that can be enabled with the **NOTIFIER\_TEST\_MODE** parameter. This parameter can be set to any value.

... NOTIFIER\_TEST\_MODE ...

When the **NOTIFIER\_TEST\_MODE** parameter is set, the notifier begins sending fake notifications to the configured delivery mechanism every **poll\_interval** interval. This provides an easy way to implement and test new or existing deliverers.

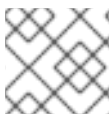
The notifier runs in **NOTIFIER\_TEST\_MODE** until the environment variable is cleared and the service is restarted.

#### 2.1.3.4. Deleting notifications

To delete the notification, you can use the **DELETE** API call. Deleting a notification ID manually cleans up resources in the notifier. If you do not use the **DELETE** API call, the notifier waits a predetermined length of time before clearing delivered notifications from its database.

## 2.2. CLAIR AUTHENTICATION

In its current iteration, Clair v4 (Clair) handles authentication internally.



### NOTE

Previous versions of Clair used JWT Proxy to gate authentication.

Authentication is configured by specifying configuration objects underneath the **auth** key of the configuration. Multiple authentication configurations might be present, but they are used preferentially in the following order:

1. PSK. With this authentication configuration, Clair implements JWT-based authentication using a pre-shared key.
2. Configuration. For example:

```
auth:  
  psk:  
    key: >-  
      MDQ4ODBINDAtNDc0ZC00MWUxLThhMzAtOTk0MzEwMGQwYTMxCg==  
    iss: 'issuer'
```

In this configuration the **auth** field requires two parameters: **iss**, which is the issuer to validate all incoming requests, and **key**, which is a base64 coded symmetric key for validating the requests.

## 2.3. CLAIR UPDATERS

Clair uses **Go** packages called *updaters* that contain the logic of fetching and parsing different vulnerability databases.

Updaters are usually paired with a matcher to interpret if, and how, any vulnerability is related to a package. Administrators might want to update the vulnerability database less frequently, or not import vulnerabilities from databases that they know will not be used.

### 2.3.1. Configuring updaters

Updaters can be configured by the **updaters** key at the top of the configuration. If updaters are being run automatically within the matcher process, which is the default setting, the period for running updaters is configured under the matcher's configuration field.

### 2.3.1.1. Updater sets

The following sets can be configured with Clair updaters:

- **alpine**
- **aws**
- **debian**
- **enricher/cvss**
- **libvuln/driver**
- **oracle**
- **photon**
- **pyupio**
- **rhel**
- **rhel/rhcc**
- **suse**
- **ubuntu**
- **updater**

### 2.3.1.2. Selecting updater sets

Specific sets of updaters can be selected by the **sets** list. For example:

```
updaters:
  sets:
    - rhel
```

If the **sets** field is not populated, it defaults to using all sets.

### 2.3.1.3. Filtering updater sets

To reject an updater from running without disabling an entire set, the **filter** option can be used.

In the following example, the string is interpreted as a Go **regexp** package. This rejects any updater with a name that does not match.



#### NOTE

This means that an empty string matches any string. It does not mean that it matches no strings.

```
updaters:
  filter: '^$'
```

### 2.3.1.4. Configuring specific updaters

Configuration for specific updaters can be passed by putting a key underneath the **config** parameter of the **updaters** object. The name of an updater might be constructed dynamically, and users should examine logs to ensure updater names are accurate. The specific object that an updater expects should be covered in the updater's documentation.

In the following example, the **rhel** updater fetches a manifest from a different location:

```
updaters:
  config:
    rhel:
      url: https://example.com/mirror/oval/PULP_MANIFEST
```

### 2.3.1.5. Disabling the Clair Updater component

In some scenarios, users might want to disable the Clair updater component. Disabling updaters is required when running Red Hat Quay in a disconnected environment.

In the following example, Clair updaters are disabled:

```
matcher:
  disable_updaters: true
```

## 2.3.2. Clair updater URLs

The following are the HTTP hosts and paths that Clair will attempt to talk to in a default configuration. This list is non-exhaustive. Some servers issue redirects and some request URLs are constructed dynamically.

- <https://secdb.alpinelinux.org/>
- [http://repo.us-west-2.amazonaws.com/2018.03/updates/x86\\_64/mirror.list](http://repo.us-west-2.amazonaws.com/2018.03/updates/x86_64/mirror.list)
- [https://cdn.amazonlinux.com/2/core/latest/x86\\_64/mirror.list](https://cdn.amazonlinux.com/2/core/latest/x86_64/mirror.list)
- <https://www.debian.org/security/oval/>
- <https://linux.oracle.com/security/oval/>
- [https://packages.vmware.com/photon/photon\\_oval\\_definitions/](https://packages.vmware.com/photon/photon_oval_definitions/)
- <https://github.com/pyupio/safety-db/archive/>
- <https://catalog.redhat.com/api/containers/>
- <https://www.redhat.com/security/data/>
- <https://support.novell.com/security/oval/>
- <https://people.canonical.com/~ubuntu-security/oval/>



## CHAPTER 3. ABOUT CLAIR

The content in this section highlights Clair releases, official Clair containers, and information about CVSS enrichment data.

### 3.1. CLAIR RELEASES

New versions of Clair are regularly released. The source code needed to build Clair is packaged as an archive and attached to each release. Clair releases can be found at [Clair releases](#).

Release artifacts also include the **clairctl** command line interface tool, which obtains updater data from the internet by using an open host.

### 3.2. CLAIR SUPPORTED LANGUAGES

Clair supports the following languages: \* Python \* Java (CRDA must be enabled)

### 3.3. CLAIR CONTAINERS

Official downstream Clair containers bundled with Red Hat Quay can be found on the [Red Hat Ecosystem Catalog](#).

Official upstream containers are packaged and released as a container at [Quay.io/projectquay/clair](#). The latest tag tracks the Git development branch. Version tags are built from the corresponding release.

### 3.4. CVE RATINGS FROM THE NATIONAL VULNERABILITY DATABASE

As of Clair v4.2, Common Vulnerability Scoring System (CVSS) enrichment data is now viewable in the Red Hat Quay UI. Additionally, Clair v4.2 adds CVSS scores from the National Vulnerability Database for detected vulnerabilities.

With this change, if the vulnerability has a CVSS score that is within 2 levels of the distribution score, the Red Hat Quay UI present's the distribution's score by default. For example:

DESCRIPTION

The SUSE coreutils-118n.patch for GNU coreutils allows context-dependent attackers to cause a denial of service (segmentation fault and crash) via a long string to the uniq command, which triggers a stack-based buffer overflow in the alloca function.

▼ CVE-2015-4041 Unknown \* coreutils 8.30-3 ADD rootfs.tar / # buildkit

SEVERITY NOTE

Note that this vulnerability was originally given a CVSSv3 score of 7.8 by NVD but was subsequently reclassified as Unknown by Unknown

VECTORS

| Attack Vector   | Attack Complexity  | Privileges Required  | User Interaction  | Scope   | Confidentiality Impact   | Integrity Impact   | Availability Impact  |
|---|--|--|---|---|--|--|--|
| <input type="radio"/> Network<br><input type="radio"/> Adjacent Network<br><input checked="" type="radio"/> Local<br><input type="radio"/> Physical | <input checked="" type="radio"/> Low<br><input type="radio"/> High | <input type="radio"/> None<br><input checked="" type="radio"/> Low<br><input type="radio"/> High | <input checked="" type="radio"/> None<br><input type="radio"/> Required | <input checked="" type="radio"/> Unchanged<br><input type="radio"/> Changed | <input checked="" type="radio"/> High<br><input type="radio"/> Low<br><input type="radio"/> None | <input checked="" type="radio"/> High<br><input type="radio"/> Low<br><input type="radio"/> None | <input checked="" type="radio"/> High<br><input type="radio"/> Low<br><input type="radio"/> None |

DESCRIPTION

The keycompare\_mb function in sort.c in sort in GNU Coreutils through 8.23 on 64-bit platforms performs a size calculation without considering the number of bytes occupied by multibyte characters, which allows attackers to cause a denial of service (heap-based buffer overflow and application crash) or possibly have unspecified other impact via long UTF-8 strings.

This differs from the previous interface, which would only display the following information:

▼ CVE-2015-4041 Unknown coreutils 8.30-3 ADD rootfs.tar / # buildkit

DESCRIPTION

The keycompare\_mb function in sort.c in sort in GNU Coreutils through 8.23 on 64-bit platforms performs a size calculation without considering the number of bytes occupied by multibyte characters, which allows attackers to cause a denial of service (heap-based buffer overflow and application crash) or possibly have unspecified other impact via long UTF-8 strings.

### 3.5. FEDERAL INFORMATION PROCESSING STANDARD (FIPS) READINESS AND COMPLIANCE

The Federal Information Processing Standard (FIPS) developed by the National Institute of Standards and Technology (NIST) is regarded as the highly regarded for securing and encrypting sensitive data, notably in highly regulated areas such as banking, healthcare, and the public sector. Red Hat Enterprise Linux (RHEL) and OpenShift Container Platform support the FIPS standard by providing a *FIPS mode*, in which the system only allows usage of specific FIPS-validated cryptographic modules like **openssl**. This ensures FIPS compliance.

Red Hat Quay supports running on FIPS-enabled RHEL and OpenShift Container Platform environments from Red Hat Quay version 3.5.0.

## PART II. CLAIR ON RED HAT QUAY

This guide contains procedures for running Clair on Red Hat Quay in both standalone and OpenShift Container Platform Operator deployments.

## CHAPTER 4. SETTING UP CLAIR ON STANDALONE RED HAT QUAY DEPLOYMENTS

For standalone Red Hat Quay deployments, you can set up Clair manually.

### Procedure

1. In your Red Hat Quay installation directory, create a new directory for the Clair database data:

```
$ mkdir /home/<user-name>/quay-poc/postgres-clairv4
```

2. Set the appropriate permissions for the **postgres-clairv4** file by entering the following command:

```
$ setfacl -m u:26:-wx /home/<user-name>/quay-poc/postgres-clairv4
```

3. Deploy a Clair Postgres database by entering the following command:

```
$ sudo podman run -d --name postgresql-clairv4 \
  -e POSTGRESQL_USER=clairuser \
  -e POSTGRESQL_PASSWORD=clairpass \
  -e POSTGRESQL_DATABASE=clair \
  -e POSTGRESQL_ADMIN_PASSWORD=adminpass \
  -p 5433:5433 \
  -v /home/<user-name>/quay-poc/postgres-clairv4:/var/lib/pgsql/data:Z \
  {postgresimage}
```

4. Install the Postgres **uuid-osp** module for your Clair deployment:

```
$ podman exec -it postgresql-clairv4 /bin/bash -c 'echo "CREATE EXTENSION IF NOT EXISTS \"uuid-osp\"" | psql -d clair -U postgres'
```

### Example output

```
CREATE EXTENSION
```



#### NOTE

Clair requires the **uuid-osp** extension to be added to its Postgres database. For users with proper privileges, creating the extension will automatically be added by Clair. If users do not have the proper privileges, the extension must be added before start Clair.

If the extension is not present, the following error will be displayed when Clair attempts to start: **ERROR: Please load the "uuid-osp" extension. (SQLSTATE 42501).**

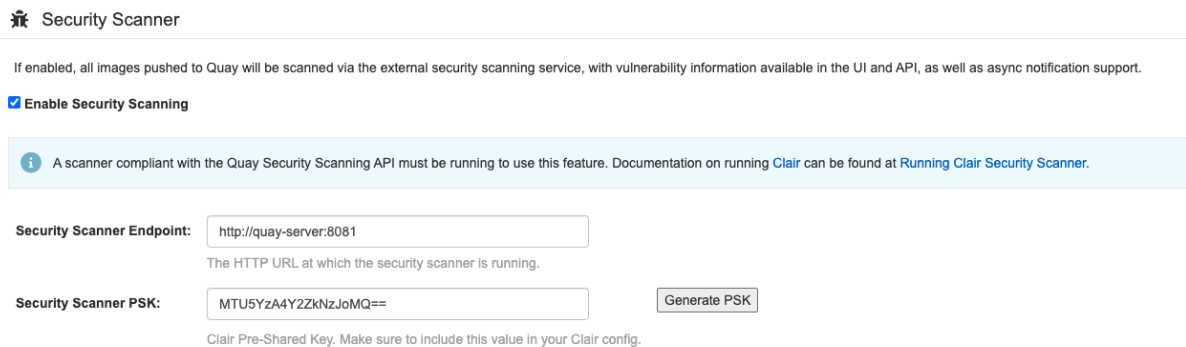
5. Stop the **Quay** container if it is running and restart it in configuration mode, loading the existing configuration as a volume:

```
$ sudo podman run --rm -it --name quay_config \
  -p 80:8080 -p 443:8443 \
```

```
-v $QUAY/config:/conf/stack:Z \
{productrepo}/{quayimage}:{productminv} config secret
```

- Log in to the configuration tool and click **Enable Security Scanning** in the **Security Scanner** section of the UI.
- Set the HTTP endpoint for Clair using a port that is not already in use on the **quay-server** system, for example, **8081**.
- Create a pre-shared key (PSK) using the **Generate PSK** button.

## Security Scanner UI



**Security Scanner**

If enabled, all images pushed to Quay will be scanned via the external security scanning service, with vulnerability information available in the UI and API, as well as async notification support.

**Enable Security Scanning**

*i* A scanner compliant with the Quay Security Scanning API must be running to use this feature. Documentation on running Clair can be found at [Running Clair Security Scanner](#).

**Security Scanner Endpoint:**   
The HTTP URL at which the security scanner is running.

**Security Scanner PSK:**    
Clair Pre-Shared Key. Make sure to include this value in your Clair config.

- Validate and download the **config.yaml** file for Red Hat Quay, and then stop the **Quay** container that is running the configuration editor.
- Extract the new configuration bundle into your Red Hat Quay installation directory, for example:

```
$ tar xvf quay-config.tar.gz -d /home/<user-name>/quay-poc/
```

- Create a folder for your Clair configuration file, for example:

```
$ mkdir /etc/opt/clairv4/config/
```

- Change into the Clair configuration folder:

```
$ cd /etc/opt/clairv4/config/
```

- Create a Clair configuration file, for example:

```
http_listen_addr: :8081
introspection_addr: :8088
log_level: debug
indexer:
  connstring: host=quay-server.example.com port=5433 dbname=clair user=clairuser
  password=clairpass sslmode=disable
  scanlock_retry: 10
  layer_scan_concurrency: 5
  migrations: true
matcher:
  connstring: host=quay-server.example.com port=5433 dbname=clair user=clairuser
  password=clairpass sslmode=disable
  max_conn_pool: 100
run: ""
```

```

migrations: true
indexer_addr: clair-indexer
notifier:
  connstring: host=quay-server.example.com port=5433 dbname=clair user=clairuser
  password=clairpass sslmode=disable
  delivery_interval: 1m
  poll_interval: 5m
  migrations: true
auth:
  psk:
    key: "MTU5YzA4Y2ZkNzJoMQ=="
    iss: ["quay"]
# tracing and metrics
trace:
  name: "jaeger"
  probability: 1
  jaeger:
    agent_endpoint: "localhost:6831"
    service_name: "clair"
metrics:
  name: "prometheus"

```

For more information about Clair's configuration format, see [Clair configuration reference](#).

14. Start Clair by using the container image, mounting in the configuration from the file you created:

```

$ sudo podman run -d --name clairv4 \
-p 8081:8081 -p 8088:8088 \
-e CLAIR_CONF=/clair/config.yaml \
-e CLAIR_MODE=combo \
-v /etc/opt/clairv4/config:/clair:Z \
registry.redhat.io/quay/clair-rhel8:v3.8.15

```



#### NOTE

Running multiple Clair containers is also possible, but for deployment scenarios beyond a single container the use of a container orchestrator like Kubernetes or OpenShift Container Platform is strongly recommended.

## CHAPTER 5. CLAIR ON OPENSIFT CONTAINER PLATFORM

To set up Clair v4 (Clair) on a Red Hat Quay deployment on OpenShift Container Platform, it is recommended to use the Red Hat Quay Operator. By default, the Red Hat Quay Operator will install or upgrade a Clair deployment along with your Red Hat Quay deployment and configure Clair automatically.

## CHAPTER 6. TESTING CLAIR

Use the following procedure to test Clair on either a standalone Red Hat Quay deployment, or on an OpenShift Container Platform Operator-based deployment.

### Prerequisites

- You have deployed the Clair container image.

### Procedure

1. Pull a sample image by entering the following command:

```
$ podman pull ubuntu:20.04
```

2. Tag the image to your registry by entering the following command:

```
$ sudo podman tag docker.io/library/ubuntu:20.04 <quay-server.example.com>/<user-name>/ubuntu:20.04
```

3. Push the image to your Red Hat Quay registry by entering the following command:

```
$ sudo podman push --tls-verify=false quay-server.example.com/quayadmin/ubuntu:20.04
```

4. Log in to your Red Hat Quay deployment through the UI.
5. Click the repository name, for example, **quayadmin/ubuntu**.
6. In the navigation pane, click **Tags**.

### Report summary

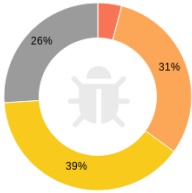
| TAG   | LAST MODIFIED | SECURITY SCAN       | SIZE    | EXPIRES | MANIFEST            |
|-------|---------------|---------------------|---------|---------|---------------------|
| 18.04 | 9 days ago    | 6 High · 82 fixable | 25.5 MB | Never   | SHA256 b58746c8a899 |
| 19.04 | 10 days ago   | Passed              | 26.4 MB | Never   | SHA256 61844ceb1dd5 |

7. Click the image report, for example, **45 medium**, to show a more detailed report:

### Report details



← clairv4-org/ubuntu **b58746c8a899**



Quay Security Scanner has detected **146** vulnerabilities.  
Patches are available for **82** vulnerabilities.

- ▲ **6** High-level vulnerabilities.
- ▲ **45** Medium-level vulnerabilities.
- ▲ **57** Low-level vulnerabilities.
- ▲ **38** Negligible-level vulnerabilities.

---

**Vulnerabilities** Filter Vulnerabilities...  Only show fixable

| CVE                                | SEVERITY                                | PACKAGE       | CURRENT VERSION  | FIXED IN VERSION  | INTRODUCED IN LAYER  |
|------------------------------------|---|---------------|------------------|-------------------|--|
| ▶ CVE-2019-3462 <a href="#">🔗</a>  | <span style="color: red;">▲</span> High | apt           | 1.6.12           | 🟢 1.7.0ubuntu0.1  | <span style="background-color: black; color: white; padding: 2px;">ADD</span> file:c3e6bb316dfa6b81dd4478aaa310df532883... |
| ▶ CVE-2019-3462 <a href="#">🔗</a>  | <span style="color: red;">▲</span> High | libapt-pkg5.0 | 1.6.12           | 🟢 1.7.0ubuntu0.1  | <span style="background-color: black; color: white; padding: 2px;">ADD</span> file:c3e6bb316dfa6b81dd4478aaa310df532883... |
| ▶ CVE-2018-16864 <a href="#">🔗</a> | <span style="color: red;">▲</span> High | libudev1      | 237-3ubuntu10.39 | 🟢 239-7ubuntu10.6 | <span style="background-color: black; color: white; padding: 2px;">ADD</span> file:c3e6bb316dfa6b81dd4478aaa310df532883... |

## PART III. ADVANCED CLAIR CONFIGURATION

Use this section to configure advanced Clair features.

## CHAPTER 7. UNMANAGED CLAIR CONFIGURATION

Red Hat Quay users can run an unmanaged Clair configuration with the Red Hat Quay OpenShift Container Platform Operator. This feature allows users to create an unmanaged Clair database, or run their custom Clair configuration without an unmanaged database.

An unmanaged Clair database allows the Red Hat Quay Operator to work in a geo-replicated environment, where multiple instances of the Operator must communicate with the same database. An unmanaged Clair database can also be used when a user requires a highly-available (HA) Clair database that exists outside of a cluster.

### 7.1. RUNNING A CUSTOM CLAIR CONFIGURATION WITH AN UNMANAGED CLAIR DATABASE

Use the following procedure to set your Clair database to unmanaged.

#### Procedure

- In the Quay Operator, set the **clairpostgres** component of the **QuayRegistry** custom resource to **managed: false**:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: quay370
spec:
  configBundleSecret: config-bundle-secret
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: clairpostgres
      managed: false
```

### 7.2. CONFIGURING A CUSTOM CLAIR DATABASE WITH AN UNMANAGED CLAIR DATABASE

The Red Hat Quay Operator for OpenShift Container Platform allows users to provide their own Clair database.

Use the following procedure to create a custom Clair database.



#### NOTE

The following procedure sets up Clair with SSL/TLS certifications. To view a similar procedure that does not set up Clair with SSL/TSL certifications, see "Configuring a custom Clair database with a managed Clair configuration".

#### Procedure

1. Create a Quay configuration bundle secret that includes the **clair-config.yaml** by entering the following command:

```
$ oc create secret generic --from-file config.yaml=./config.yaml --from-file extra_ca_cert_rds-ca-2019-root.pem=./rds-ca-2019-root.pem --from-file clair-config.yaml=./clair-config.yaml --from-file ssl.cert=./ssl.cert --from-file ssl.key=./ssl.key config-bundle-secret
```

### Example Clair config.yaml file

```
indexer:
  connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb
  password=quayrdsdb sslrootcert=/run/certs/rds-ca-2019-root.pem sslmode=verify-ca
  layer_scan_concurrency: 6
  migrations: true
  scanlock_retry: 11
log_level: debug
matcher:
  connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb
  password=quayrdsdb sslrootcert=/run/certs/rds-ca-2019-root.pem sslmode=verify-ca
  migrations: true
metrics:
  name: prometheus
notifier:
  connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb
  password=quayrdsdb sslrootcert=/run/certs/rds-ca-2019-root.pem sslmode=verify-ca
  migrations: true
```

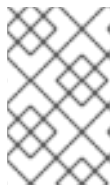


### NOTE

- The database certificate is mounted under **/run/certs/rds-ca-2019-root.pem** on the Clair application pod in the **clair-config.yaml**. It must be specified when configuring your **clair-config.yaml**.
- An example **clair-config.yaml** can be found at [Clair on OpenShift config](#).

2. Add the **clair-config.yaml** file to your bundle secret, for example:

```
apiVersion: v1
kind: Secret
metadata:
  name: config-bundle-secret
  namespace: quay-enterprise
data:
  config.yaml: <base64 encoded Quay config>
  clair-config.yaml: <base64 encoded Clair config>
  extra_ca_cert_<name>: <base64 encoded ca cert>
  ssl.crt: <base64 encoded SSL certificate>
  ssl.key: <base64 encoded SSL private key>
```

**NOTE**

When updated, the provided **clair-config.yaml** file is mounted into the Clair pod. Any fields not provided are automatically populated with defaults using the Clair configuration module.

3. You can check the status of your Clair pod by clicking the commit in the **Build History** page, or by running **oc get pods -n <namespace>**. For example:

```
$ oc get pods -n <namespace>
```

**Example output**

```
NAME                                READY STATUS  RESTARTS  AGE
f192fe4a-c802-4275-bcce-d2031e635126-9l2b5-25lg2  1/1  Running  0         7s
```

## CHAPTER 8. RUNNING A CUSTOM CLAIR CONFIGURATION WITH A MANAGED CLAIR DATABASE

In some cases, users might want to run a custom Clair configuration with a managed Clair database. This is useful in the following scenarios:

- When a user wants to disable specific updater resources.
- When a user is running Red Hat Quay in a disconnected environment. For more information about running Clair in a disconnected environment, see [Configuring access to the Clair database in the air-gapped OpenShift cluster](#).



### NOTE

- If you are running Red Hat Quay in a disconnected environment, the **airgap** parameter of your **clair-config.yaml** must be set to **true**.
- If you are running Red Hat Quay in a disconnected environment, you should disable all updater components.

### 8.1. SETTING A CLAIR DATABASE TO MANAGED

Use the following procedure to set your Clair database to managed.

#### Procedure

- In the Quay Operator, set the **clairpostgres** component of the **QuayRegistry** custom resource to **managed: true**:

```
apiVersion: quay.redhat.com/v1
kind: QuayRegistry
metadata:
  name: quay370
spec:
  configBundleSecret: config-bundle-secret
  components:
    - kind: objectstorage
      managed: false
    - kind: route
      managed: true
    - kind: tls
      managed: false
    - kind: clairpostgres
      managed: true
```

### 8.2. CONFIGURING A CUSTOM CLAIR DATABASE WITH A MANAGED CLAIR CONFIGURATION

The Red Hat Quay Operator for OpenShift Container Platform allows users to provide their own Clair database.

Use the following procedure to create a custom Clair database.

## Procedure

1. Create a Quay configuration bundle secret that includes the **clair-config.yaml** by entering the following command:

```
$ oc create secret generic --from-file config.yaml=./config.yaml --from-file extra_ca_cert_rds-ca-2019-root.pem=./rds-ca-2019-root.pem --from-file clair-config.yaml=./clair-config.yaml config-bundle-secret
```

### Example Clair config.yaml file

```
indexer:
  connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb
  password=quayrdsdb sslmode=disable
  layer_scan_concurrency: 6
  migrations: true
  scanlock_retry: 11
log_level: debug
matcher:
  connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb
  password=quayrdsdb sslmode=disable
  migrations: true
metrics:
  name: prometheus
notifier:
  connstring: host=quay-server.example.com port=5432 dbname=quay user=quayrdsdb
  password=quayrdsdb sslmode=disable
  migrations: true
```



#### NOTE

- The database certificate is mounted under **/run/certs/rds-ca-2019-root.pem** on the Clair application pod in the **clair-config.yaml**. It must be specified when configuring your **clair-config.yaml**.
- An example **clair-config.yaml** can be found at [Clair on OpenShift config](#).

2. Add the **clair-config.yaml** file to your bundle secret, for example:

```
apiVersion: v1
kind: Secret
metadata:
  name: config-bundle-secret
  namespace: quay-enterprise
data:
  config.yaml: <base64 encoded Quay config>
  clair-config.yaml: <base64 encoded Clair config>
```



#### NOTE

- When updated, the provided **clair-config.yaml** file is mounted into the Clair pod. Any fields not provided are automatically populated with defaults using the Clair configuration module.

3. You can check the status of your Clair pod by clicking the commit in the **Build History** page, or by running **oc get pods -n <namespace>**. For example:

```
$ oc get pods -n <namespace>
```

#### Example output

```
NAME                                READY STATUS  RESTARTS  AGE
f192fe4a-c802-4275-bcce-d2031e635126-9l2b5-25lg2  1/1   Running  0         7s
```



## CHAPTER 9. CLAIR IN DISCONNECTED ENVIRONMENTS

Clair uses a set of components called *updaters* to handle the fetching and parsing of data from various vulnerability databases. Updaters are set up by default to pull vulnerability data directly from the internet and work for immediate use. However, some users might require Red Hat Quay to run in a disconnected environment, or an environment without direct access to the internet. Clair supports disconnected environments by working with different types of update workflows that take network isolation into consideration. This works by using the **clairctl** command line interface tool, which obtains updater data from the internet by using an open host, securely transferring the data to an isolated host, and then importing the updater data on the isolated host into Clair.

Use this guide to deploy Clair in a disconnected environment.



### NOTE

Currently, Clair enrichment data is CVSS data. Enrichment data is currently unsupported in disconnected environments.

For more information about Clair updaters, see "Clair updaters".

## 9.1. SETTING UP CLAIR IN A DISCONNECTED OPENSIFT CONTAINER PLATFORM CLUSTER

Use the following procedures to set up an OpenShift Container Platform provisioned Clair pod in a disconnected OpenShift Container Platform cluster.

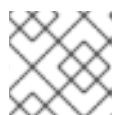
### 9.1.1. Installing the clairctl command line utility tool for OpenShift Container Platform deployments

Use the following procedure to install the **clairctl** CLI tool for OpenShift Container Platform deployments.

#### Procedure

1. Install the **clairctl** program for a Clair deployment in an OpenShift Container Platform cluster by entering the following command:

```
$ oc -n quay-enterprise exec example-registry-clair-app-64dd48f866-6ptgw -- cat /usr/bin/clairctl > clairctl
```



### NOTE

Unofficially, the **clairctl** tool can be downloaded

2. Set the permissions of the **clairctl** file so that it can be executed and run by the user, for example:

```
$ chmod u+x ./clairctl
```

### 9.1.2. Retrieving and decoding the Clair configuration secret for Clair deployments on OpenShift Container Platform

Use the following procedure to retrieve and decode the configuration secret for an OpenShift Container Platform provisioned Clair instance on OpenShift Container Platform.

### Prerequisites

- You have installed the **clairctl** command line utility tool.

### Procedure

1. Enter the following command to retrieve and decode the configuration secret, and then save it to a Clair configuration YAML:

```
$ oc get secret -n quay-enterprise example-registry-clair-config-secret -o "jsonpath={$.data['config.yaml']}" | base64 -d > clair-config.yaml
```

2. Update the **clair-config.yaml** file so that the **disable\_updaters** and **airgap** parameters are set to **true**, for example:

```
---
indexer:
  airgap: true
---
matcher:
  disable_updaters: true
---
```

## 9.1.3. Exporting the updaters bundle from a connected Clair instance

Use the following procedure to export the updaters bundle from a Clair instance that has access to the internet.

### Prerequisites

- You have installed the **clairctl** command line utility tool.
- You have retrieved and decoded the Clair configuration secret, and saved it to a Clair **config.yaml** file.
- The **disable\_updaters** and **airgap** parameters are set to **true** in your Clair **config.yaml** file.

### Procedure

- From a Clair instance that has access to the internet, use the **clairctl** CLI tool with your configuration file to export the updaters bundle. For example:

```
$ ./clairctl --config ./config.yaml export-updaters updates.gz
```

## 9.1.4. Configuring access to the Clair database in the disconnected OpenShift Container Platform cluster

Use the following procedure to configure access to the Clair database in your disconnected OpenShift Container Platform cluster.

## Prerequisites

- You have installed the **clairctl** command line utility tool.
- You have retrieved and decoded the Clair configuration secret, and saved it to a Clair **config.yaml** file.
- The **disable\_updaters** and **airgap** parameters are set to **true** in your Clair **config.yaml** file.
- You have exported the updaters bundle from a Clair instance that has access to the internet.

## Procedure

1. Determine your Clair database service by using the **oc** CLI tool, for example:

```
$ oc get svc -n quay-enterprise
```

### Example output

```
NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP  PORT(S)
AGE
example-registry-clair-app          ClusterIP     172.30.224.93 <none>
80/TCP,8089/TCP                    4d21h
example-registry-clair-postgres    ClusterIP     172.30.246.88 <none>      5432/TCP
4d21h
...
```

2. Forward the Clair database port so that it is accessible from the local machine. For example:

```
$ oc port-forward -n quay-enterprise service/example-registry-clair-postgres 5432:5432
```

3. Update your Clair **config.yaml** file, for example:

```
indexer:
  connstring: host=localhost port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable ❶
  scanlock_retry: 10
  layer_scan_concurrency: 5
  migrations: true
scanner:
  repo:
    rhel-repository-scanner: ❷
    repo2cpe_mapping_file: /data/cpe-map.json
  package:
    rhel_containerscanner: ❸
    name2repos_mapping_file: /data/repo-map.json
```

- ❶ Replace the value of the **host** in the multiple **connstring** fields with **localhost**.
- ❷ For more information about the **rhel-repository-scanner** parameter, see "Mapping repositories to Common Product Enumeration information".
- ❸ For more information about the **rhel\_containerscanner** parameter, see "Mapping repositories to Common Product Enumeration information".

## 9.1.5. Importing the updaters bundle into the disconnected OpenShift Container Platform cluster

Use the following procedure to import the updaters bundle into your disconnected OpenShift Container Platform cluster.

### Prerequisites

- You have installed the **clairctl** command line utility tool.
- You have retrieved and decoded the Clair configuration secret, and saved it to a Clair **config.yaml** file.
- The **disable\_updaters** and **airgap** parameters are set to **true** in your Clair **config.yaml** file.
- You have exported the updaters bundle from a Clair instance that has access to the internet.
- You have transferred the updaters bundle into your disconnected environment.

### Procedure

- Use the **clairctl** CLI tool to import the updaters bundle into the Clair database that is deployed by OpenShift Container Platform. For example:

```
$ ./clairctl --config ./clair-config.yaml import-updaters updates.gz
```

## 9.2. SETTING UP A SELF-MANAGED DEPLOYMENT OF CLAIR FOR A DISCONNECTED OPENSIFT CONTAINER PLATFORM CLUSTER

Use the following procedures to set up a self-managed deployment of Clair for a disconnected OpenShift Container Platform cluster.

### 9.2.1. Installing the clairctl command line utility tool for a self-managed Clair deployment on OpenShift Container Platform

Use the following procedure to install the **clairctl** CLI tool for self-managed Clair deployments on OpenShift Container Platform.

#### Procedure

1. Install the **clairctl** program for a self-managed Clair deployment by using the **podman cp** command, for example:

```
$ sudo podman cp clairv4:/usr/bin/clairctl ./clairctl
```

2. Set the permissions of the **clairctl** file so that it can be executed and run by the user, for example:

```
$ chmod u+x ./clairctl
```

### 9.2.2. Deploying a self-managed Clair container for disconnected OpenShift Container Platform clusters

Use the following procedure to deploy a self-managed Clair container for disconnected OpenShift Container Platform clusters.

### Prerequisites

- You have installed the **clairctl** command line utility tool.

### Procedure

1. Create a folder for your Clair configuration file, for example:

```
$ mkdir /etc/clairv4/config/
```

2. Create a Clair configuration file with the **disable\_updaters** parameter set to **true**, for example:

```
---
indexer:
  airgap: true
---
matcher:
  disable_updaters: true
---
```

3. Start Clair by using the container image, mounting in the configuration from the file you created:

```
$ sudo podman run -it --rm --name clairv4 \
-p 8081:8081 -p 8088:8088 \
-e CLAIR_CONF=/clair/config.yaml \
-e CLAIR_MODE=combo \
-v /etc/clairv4/config:/clair:Z \
registry.redhat.io/quay/clair-rhel8:v3.8.15
```

### 9.2.3. Exporting the updaters bundle from a connected Clair instance

Use the following procedure to export the updaters bundle from a Clair instance that has access to the internet.

### Prerequisites

- You have installed the **clairctl** command line utility tool.
- You have deployed Clair.
- The **disable\_updaters** and **airgap** parameters are set to **true** in your Clair **config.yaml** file.

### Procedure

- From a Clair instance that has access to the internet, use the **clairctl** CLI tool with your configuration file to export the updaters bundle. For example:

```
$ ./clairctl --config ./config.yaml export-updaters updates.gz
```

## 9.2.4. Configuring access to the Clair database in the disconnected OpenShift Container Platform cluster

Use the following procedure to configure access to the Clair database in your disconnected OpenShift Container Platform cluster.

### Prerequisites

- You have installed the **clairctl** command line utility tool.
- You have deployed Clair.
- The **disable\_updaters** and **airgap** parameters are set to **true** in your Clair **config.yaml** file.
- You have exported the updaters bundle from a Clair instance that has access to the internet.

### Procedure

1. Determine your Clair database service by using the **oc** CLI tool, for example:

```
$ oc get svc -n quay-enterprise
```

#### Example output

```
NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP  PORT(S)
AGE
example-registry-clair-app          ClusterIP     172.30.224.93 <none>
80/TCP,8089/TCP                    4d21h
example-registry-clair-postgres    ClusterIP     172.30.246.88 <none>      5432/TCP
4d21h
...
```

2. Forward the Clair database port so that it is accessible from the local machine. For example:

```
$ oc port-forward -n quay-enterprise service/example-registry-clair-postgres 5432:5432
```

3. Update your Clair **config.yaml** file, for example:

```
indexer:
  connstring: host=localhost port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable ❶
scanlock_retry: 10
layer_scan_concurrency: 5
migrations: true
scanner:
  repo:
    rhel-repository-scanner: ❷
    repo2cpe_mapping_file: /data/cpe-map.json
  package:
    rhel_containerscanner: ❸
    name2repos_mapping_file: /data/repo-map.json
```

- ❶ Replace the value of the **host** in the multiple **connstring** fields with **localhost**.

- 2 For more information about the **rhel-repository-scanner** parameter, see "Mapping repositories to Common Product Enumeration information".
- 3 For more information about the **rhel\_containerscanner** parameter, see "Mapping repositories to Common Product Enumeration information".

### 9.2.5. Importing the updaters bundle into the disconnected OpenShift Container Platform cluster

Use the following procedure to import the updaters bundle into your disconnected OpenShift Container Platform cluster.

#### Prerequisites

- You have installed the **clairctl** command line utility tool.
- You have deployed Clair.
- The **disable\_updaters** and **airgap** parameters are set to **true** in your Clair **config.yaml** file.
- You have exported the updaters bundle from a Clair instance that has access to the internet.
- You have transferred the updaters bundle into your disconnected environment.

#### Procedure

- Use the **clairctl** CLI tool to import the updaters bundle into the Clair database that is deployed by OpenShift Container Platform:

```
$ ./clairctl --config ./clair-config.yaml import-updaters updates.gz
```

## 9.3. ENABLING CLAIR CRDA

Java scanning depends on a public, Red Hat provided API service called Code Ready Dependency Analytics (CRDA). CRDA is only available with internet access and is not enabled by default.

Use the following procedure to integrate the CRDA service with a custom API key and enable CRDA for Java and Python scanning.

#### Prerequisites

- Red Hat Quay 3.7 or greater

#### Procedure

1. Submit [the API key request form](#) to obtain the Quay-specific CRDA remote matcher.
2. Set the CRDA configuration in your **clair-config.yaml** file:

```
matchers:
  config:
    crda:
```

```
url: https://gw.api.openshift.io/api/v2/
key: <CRDA_API_KEY> 1
source: <QUAY_SERVER_HOSTNAME> 2
```

- 1** Insert the Quay-specific CRDA remote matcher from [the API key request form](#) here.
- 2** The hostname of your Quay server.

## 9.4. MAPPING REPOSITORIES TO COMMON PRODUCT ENUMERATION INFORMATION

Clair's Red Hat Enterprise Linux (RHEL) scanner relies on a Common Product Enumeration (CPE) file to map RPM packages to the corresponding security data to produce matching results. These files are owned by product security and updated daily.

The CPE file must be present, or access to the file must be allowed, for the scanner to properly process RPM packages. If the file is not present, RPM packages installed in the container image will not be scanned.

Table 9.1. Clair CPE mapping files

| CPE                | Link to JSON mapping file                      |
|--------------------|--|
| <b>repos2cpe</b>   | <a href="#">Red Hat Repository-to-CPE JSON</a> |
| <b>names2repos</b> | <a href="#">Red Hat Name-to-Repos JSON</a>     |

In addition to uploading CVE information to the database for disconnected Clair installations, you must also make the mapping file available locally:

- For standalone Red Hat Quay and Clair deployments, the mapping file must be loaded into the Clair pod.
- For Red Hat Quay Operator deployments on OpenShift Container Platform and Clair deployments, you must set the Clair component to **unamanged**. Then, Clair must be deployed manually, setting the configuration to load a local copy of the mapping file.

### 9.4.1. Mapping repositories to Common Product Enumeration example configuration

Use the **repo2cpe\_mapping\_file** and **name2repos\_mapping\_file** fields in your Clair configuration to include the CPE JSON mapping files. For example:

```
indexer:
scanner:
  repo:
    rhel-repository-scanner:
      repo2cpe_mapping_file: /data/cpe-map.json
  package:
    rhel_containerscanner:
      name2repos_mapping_file: /data/repo-map.json
```

For more information, see [How to accurately match OVAL security data to installed RPMs](#) .



## CHAPTER 10. CLAIR CONFIGURATION OVERVIEW

Clair is configured by a structured YAML file. Each Clair node needs to specify what mode it will run in and a path to a configuration file through CLI flags or environment variables. For example:

```
$ clair -conf ./path/to/config.yaml -mode indexer
```

or

```
$ clair -conf ./path/to/config.yaml -mode matcher
```

The aforementioned commands each start two Clair nodes using the same configuration file. One runs the indexing facilities, while other runs the matching facilities.

Environment variables respected by the Go standard library can be specified if needed, for example:

- **HTTP\_PROXY**
- **HTTPS\_PROXY**
- **SSL\_CERT\_DIR**

If you are running Clair in **combo** mode, you must supply the indexer, matcher, and notifier configuration blocks in the configuration.

### 10.1. CLAIR CONFIGURATION REFERENCE

The following YAML shows an example Clair configuration:

```
http_listen_addr: ""
introspection_addr: ""
log_level: ""
tls: {}
indexer:
  connstring: ""
  scanlock_retry: 0
  layer_scan_concurrency: 0
  migrations: false
  scanner: {}
  airgap: false
matcher:
  connstring: ""
  indexer_addr: ""
  migrations: false
  period: ""
  disable_updaters: false
  update_retention: 2
matchers:
  names: nil
  config: nil
updaters:
  sets: nil
  config: nil
notifier:
```

```

connstring: ""
migrations: false
indexer_addr: ""
matcher_addr: ""
poll_interval: ""
delivery_interval: ""
disable_summary: false
webhook: null
amqp: null
stomp: null
auth:
  psk: nil
trace:
  name: ""
  probability: null
  jaeger:
    agent:
      endpoint: ""
    collector:
      endpoint: ""
      username: null
      password: null
      service_name: ""
    tags: nil
    buffer_max: 0
metrics:
  name: ""
  prometheus:
    endpoint: null
  dogstatsd:
    url: ""

```



## NOTE

The above YAML file lists every key for completeness. Using this configuration file as-is will result in some options not having their defaults set normally.

## 10.2. CLAIR GENERAL FIELDS

The following section describes the general configuration fields available for a Clair deployment:

| Field                           | Type   | Description  |
|---------------------------------|--------|--|
| <code>http_listen_addr</code>   | String | Configures where the HTTP API is exposed.<br><br>Default: <b>:6060</b> |
| <code>introspection_addr</code> | String | Configures where Clair's metrics and health endpoints are exposed.     |

| Field                  | Type   | Description   |
|------------------------|--------|---|
| <code>log_level</code> | String | Sets the logging level. Requires one of the following strings: <b>debug-color</b> , <b>debug</b> , <b>info</b> , <b>warn</b> , <b>error</b> , <b>fatal</b> , <b>panic</b> |
| <code>tls</code>       | String | A map containing the configuration for serving the HTTP API of TLS/SSL and HTTP/2.  |
| <code>.cert</code>     | String | The TLS certificate to be used. Must be a full-chain certificate.   |

### 10.3. CLAIR INDEXER CONFIGURATION FIELDS

The following indexer configuration fields are available for Clair.

| Field  | Type    | Description  |
|--|---------|--|
| <code>indexer</code>                           | Object  | Provides Clair indexer node configuration.   |
| <code>.airgap</code>                           | Boolean | Disables HTTP access to the internet for indexers and fetchers. Private IPv4 and IPv6 addresses are allowed. Database connections are unaffected.  |
| <code>.connstring</code>                       | String  | A Postgres connection string. Accepts format as a URL or libpq connection string.  |
| <code>.index_report_request_concurrency</code> | Integer | Rate limits the number of index report creation requests. Setting this to <b>0</b> attempts to auto-size this value. Setting a negative value means unlimited. The auto-sizing is a multiple of the number of available cores.<br><br>The API returns a <b>429</b> status code if concurrency is exceeded. |

| Field                                | Type    | Description   |
|--------------------------------------|---------|---|
| <code>.scanlock_retry</code>         | Integer | A positive integer representing seconds. Concurrent indexers lock on manifest scans to avoid clobbering. This value tunes how often a waiting indexer polls for the lock.                   |
| <code>.layer_scan_concurrency</code> | Integer | Positive integer limiting the number of concurrent layer scans. Indexers will match a manifest's layer concurrently. This value tunes the number of layers an indexer scans in parallel.    |
| <code>.migrations</code>             | Boolean | Whether indexer nodes handle migrations to their database.  |
| <code>.scanner</code>                | String  | Indexer configuration.<br><br>Scanner allows for passing configuration options to layer scanners. The scanner will have this configuration pass to it on construction if designed to do so. |
| <code>.scanner.dist</code>           | String  | A map with the name of a particular scanner and arbitrary YAML as a value.  |
| <code>.scanner.package</code>        | String  | A map with the name of a particular scanner and arbitrary YAML as a value.  |
| <code>.scanner.repo</code>           | String  | A map with the name of a particular scanner and arbitrary YAML as a value.  |

## 10.4. CLAIR MATCHER CONFIGURATION FIELDS

The following matcher configuration fields are available for Clair.



### NOTE

Differs from **matchers** configuration fields.

| Field                          | Type    | Description   |
|--------------------------------|---------|---|
| <code>matcher</code>           | Object  | Provides Clair matcher node configuration.  |
| <code>.cache_age</code>        | String  | Controls how long users should be hinted to cache responses for.  |
| <code>.connstring</code>       | String  | A Postgres connection string. Accepts format as a URL or libpq connection string.   |
| <code>.max_conn_pool</code>    | Integer | Limits the database connection pool size.<br><br>Clair allows for a custom connection pool size. This number directly sets how many active database connections are allowed concurrently.<br><br>This parameter will be ignored in a future version. Users should configure this through the connection string. |
| <code>.indexer_addr</code>     | String  | A matcher contacts an indexer to create a <b>VulnerabilityReport</b> . The location of this indexer is required.<br><br>Defaults to <b>30m</b> .  |
| <code>.migrations</code>       | Boolean | Whether matcher nodes handle migrations to their databases.   |
| <code>.period</code>           | String  | Determines how often updates for new security advisories take place.<br><br>Defaults to <b>30m</b> .  |
| <code>.disable_updaters</code> | Boolean | Whether to run background updates or not.   |

| Field                          | Type    | Description   |
|--------------------------------|---------|---|
| <code>.update_retention</code> | Integer | <p>Sets the number of update operations to retain between garbage collection cycles. This should be set to a safe MAX value based on database size constraints.</p> <p>Defaults to <b>10m</b>.</p> <p>If a value of less than <b>0</b> is provided, garbage collection is disabled. <b>2</b> is the minimum value to ensure updates can be compared to notifications.</p> |

## 10.5. CLAIR MATCHERS CONFIGURATION FIELDS

The following matchers configuration fields are available for Clair.



### NOTE

Differs from **matcher** configuration fields.

| Field                 | Type             | Description   |
|-----------------------|------------------|---|
| <code>matchers</code> | Array of strings | Provides configuration for the in-tree <b>matchers</b> and <b>remotematchers</b> .  |
| <code>.names</code>   | String           | A list of string values informing the matcher factory about enabled matchers. If value is set to <b>null</b> , the default list of matchers run: <b>alpine, aws, debian, oracle, photon, python, python, rhel, suse, ubuntu, crda</b> |

| Field                | Type   | Description  |
|----------------------|--------|--|
| <code>.config</code> | String | <p>Provides configuration to a specific matcher.</p> <p>A map keyed by the name of the matcher containing a sub-object which will be provided to the matchers factory constructor. For example:</p> <pre> config: python: ignore_vulns: - CVE-XYZ - CVE-ABC </pre> |

## 10.6. CLAIR UPDATERS CONFIGURATION FIELDS

The following updaters configuration fields are available for Clair.

| Field                 | Type   | Description  |
|-----------------------|--------|--|
| <code>updaters</code> | Object | Provides configuration for the matcher's update manager.   |
| <code>.sets</code>    | String | <p>A list of values informing the update manager which updaters to run.</p> <p>If value is set to <b>null</b>, the default set of updaters runs the following: <b>alpine, aws, debian, oracle, photon, pyupio, rhel, suse, ubuntu</b></p> <p>If left blank, zero updaters run.</p> |


| Field                | Type   | Description   |
|----------------------|--------|---|
| <code>.config</code> | String | <p>Provides configuration to specific updater sets.</p> <p>A map keyed by the name of the updater set containing a sub-object which will be provided to the updater set's constructor. For example:</p> <pre> config:   ubuntu:     security_tracker_url:       http://security.url     ignore_distributions:       - cosmic </pre> |

## 10.7. CLAIR NOTIFIER CONFIGURATION FIELDS


The following notifier configuration fields are available for Clair.

| Field                       | Type    | Description   |
|-----------------------------|---------|---|
| <code>notifier</code>       | Object  | Provides Clair notifier node configuration.   |
| <code>.connstring</code>    | String  | Postgres connection string. Accepts format as URL, or libpq connection string.  |
| <code>.migrations</code>    | Boolean | Whether notifier nodes handle migrations to their database.   |
| <code>.indexer_addr</code>  | String  | A notifier contacts an indexer to create or obtain manifests affected by vulnerabilities. The location of this indexer is required. |
| <code>.matcher_addr</code>  | String  | A notifier contacts a matcher to list update operations and acquire diffs. The location of this matcher is required.                |
| <code>.poll_interval</code> | String  | The frequency at which the notifier will query a matcher for update operations.   |



| Field                           | Type    | Description  |
|---------------------------------|---------|--|
| <code>.delivery_interval</code> | String  | The frequency at which the notifier attempts delivery of created, or previously failed, notifications.   |
| <code>.disable_summary</code>   | Boolean | Controls whether notifications should be summarized to one per manifest.   |
| <code>.webhook</code>           | Object  | Configures the notifier for webhook delivery.  |
| <code>.webhook.target</code>    | String  | URL where the webhook will be delivered.   |
| <code>.webhook.callback</code>  | String  | The callback URL where notifications can be retrieved. The notification ID will be appended to this URL.<br><br>This will typically be where the Clair notifier is hosted.   |
| <code>.webhook.headers</code>   | String  | A map associating a header name to a list of values.   |
| <code>.amqp</code>              | Object  | Configures the notifier for AMQP delivery.<br><br> <p><b>NOTE</b></p> <p>Clair does not declare any AMQP components on its own. All attempts to use an exchange or queue are passive only and will fail. Broker administrators should setup exchanges and queues ahead of time.</p> |

| Field                                   | Type    | Description  |
|---|---------|--|
| <code>.amqp.direct</code>               | Boolean | If <b>true</b> , the notifier will deliver individual notifications (not a callback) to the configured AMQP broker.  |
| <code>.amqp.rollup</code>               | Integer | When <b>amqp.direct</b> is set to <b>true</b> , this value informs the notifier of how many notifications to send in a direct delivery. For example, if <b>direct</b> is set to <b>true</b> , and <b>amqp.rollup</b> is set to <b>5</b> , the notifier delivers no more than 5 notifications in a single JSON payload to the broker. Setting the value to <b>0</b> effectively sets it to <b>1</b> . |
| <code>.amqp.exchange</code>             | Object  | The AMQP exchange to connect to.   |
| <code>.amqp.exchange.name</code>        | String  | The name of the exchange to connect to.  |
| <code>.amqp.exchange.type</code>        | String  | The type of the exchange. Typically one of the following: <b>direct</b> , <b>fanout</b> , <b>topic</b> , <b>headers</b> .  |
| <code>.amqp.exchange.durability</code>  | Boolean | Whether the configured queue is durable.   |
| <code>.amqp.exchange.auto_delete</code> | Boolean | Whether the configured queue uses an <b>auto_delete_policy</b> .   |
| <code>.amqp.routing_key</code>          | String  | The name of the routing key each notification is sent with.  |
| <code>.amqp.callback</code>             | String  | If <b>amqp.direct</b> is set to <b>false</b> , this URL is provided in the notification callback sent to the broker. This URL should point to Clair's notification API endpoint.   |
| <code>.amqp.uris</code>                 | String  | A list of one or more AMQP brokers to connect to, in priority order.   |
| <code>.amqp.tls</code>                  | Object  | Configures TLS/SSL connection to an AMQP broker.   |

| Field                          | Type    | Description  |
|--------------------------------|---------|--|
| <code>.amqp.tls.root_ca</code> | String  | The filesystem path where a root CA can be read.   |
| <code>.amqp.tls.cert</code>    | String  | <p>The filesystem path where a TLS/SSL certificate can be read.</p> <div style="display: flex; align-items: flex-start;">  <div> <p><b>NOTE</b></p> <p>Clair also allows <b>SSL_CERT_DIR</b>, as documented for the Go <b>crypto/x509</b> package.</p> </div> </div>    |
| <code>.amqp.tls.key</code>     | String  | The filesystem path where a TLS/SSL private key can be read.   |
| <code>.stomp</code>            | Object  | Configures the notifier for STOMP delivery.  |
| <code>.stomp.direct</code>     | Boolean | If <b>true</b> , the notifier delivers individual notifications (not a callback) to the configured STOMP broker.   |
| <code>.stomp.rollup</code>     | Integer | If <b>stomp.direct</b> is set to <b>true</b> , this value limits the number of notifications sent in a single direct delivery. For example, if <b>direct</b> is set to <b>true</b> , and <b>rollup</b> is set to <b>5</b> , the notifier delivers no more than 5 notifications in a single JSON payload to the broker. Setting the value to <b>0</b> effectively sets it to <b>1</b> . |
| <code>.stomp.callback</code>   | String  | If <b>stomp.callback</b> is set to <b>false</b> , the provided URL in the notification callback is sent to the broker. This URL should point to Clair's notification API endpoint.   |

| Field                             | Type   | Description   |
|-----------------------------------|--------|---|
| <code>.stomp.destination</code>   | String | The STOMP destination to deliver notifications to.  |
| <code>.stomp.uris</code>          | String | A list of one or more STOMP brokers to connect to in priority order.  |
| <code>.stomp.tls</code>           | Object | Configured TLS/SSL connection to STOMP broker.  |
| <code>.stomp.tls.root_ca</code>   | String | The filesystem path where a root CA can be read.<br><br> <p><b>NOTE</b><br/>Clair also respects <b>SSL_CERT_DIR</b>, as documented for the Go <b>crypto/x509</b> package.</p> |
| <code>.stomp.tls.cert</code>      | String | The filesystem path where a TLS/SSL certificate can be read.  |
| <code>.stomp.tls.key</code>       | String | The filesystem path where a TLS/SSL private key can be read.  |
| <code>.stomp.user</code>          | String | Configures login details for the STOMP broker.  |
| <code>.stomp.user.login</code>    | String | The STOMP login to connect with.  |
| <code>.stomp.user.passcode</code> | String | The STOMP passcode to connect with.   |

## 10.8. CLAIR AUTHORIZATION CONFIGURATION FIELDS

The following authorization configuration fields are available for Clair.

| Field | Type | Description |
|-------|------|-------------|
|-------|------|-------------|

| Field                 | Type   | Description   |
|-----------------------|--------|---|
| <code>auth</code>     | Object | Defines Clair's external and intra-service JWT based authentication. If multiple <b>auth</b> mechanisms are defined, Clair picks one. Currently, multiple mechanisms are unsupported. |
| <code>.psk</code>     | String | Defines pre-shared key authentication.  |
| <code>.psk.key</code> | String | A shared base64 encoded key distributed between all parties signing and verifying JWTs.   |
| <code>.psk.iss</code> | String | A list of JWT issuers to verify. An empty list accepts any issuer in a JWT claim.   |

## 10.9. CLAIR TRACE CONFIGURATION FIELDS

The following trace configuration fields are available for Clair.

| Field                               | Type    | Description   |
|-------------------------------------|---------|---|
| <code>trace</code>                  | Object  | Defines distributed tracing configuration based on OpenTelemetry.                         |
| <code>.name</code>                  | String  | The name of the application traces will belong to.  |
| <code>.probability</code>           | Integer | The probability a trace will occur.   |
| <code>.jaeger</code>                | Object  | Defines values for Jaeger tracing.  |
| <code>.jaeger.agent</code>          | Object  | Defines values for configuring delivery to a Jaeger agent.                                |
| <code>.jaeger.agent.endpoint</code> | String  | An address in the <b>&lt;host&gt;: &lt;port&gt;</b> syntax where traces can be submitted. |
| <code>.jaeger.collector</code>      | Object  | Defines values for configuring delivery to a Jaeger collector.                            |

| Field                                   | Type    | Description   |
|---|---------|---|
| <code>.jaeger.collector.endpoint</code> | String  | An address in the <b>&lt;host&gt;: &lt;port&gt;</b> syntax where traces can be submitted.                                       |
| <code>.jaeger.collector.username</code> | String  | A Jaeger username.  |
| <code>.jaeger.collector.password</code> | String  | A Jaeger password.  |
| <code>.jaeger.service_name</code>       | String  | The service name registered in Jaeger.  |
| <code>.jaeger.tags</code>               | String  | Key-value pairs to provide additional metadata.   |
| <code>.jaeger.buffer_max</code>         | Integer | The maximum number of spans that can be buffered in memory before they are sent to the Jaeger backend for storage and analysis. |

## 10.10. CLAIR METRICS CONFIGURATION FIELDS

The following metrics configuration fields are available for Clair.

| Field                             | Type   | Description   |
|-----------------------------------|--------|---|
| <code>metrics</code>              | Object | Defines distributed tracing configuration based on OpenTelemetry. |
| <code>.name</code>                | String | The name of the metrics in use.                                   |
| <code>.prometheus</code>          | String | Configuration for a Prometheus metrics exporter.                  |
| <code>.prometheus.endpoint</code> | String | Defines the path where metrics are served.                        |