Red Hat Quay 3.9

Vulnerability reporting with Clair on Red Hat Quay
Abstract

Get started with Red Hat Quay
# Table of Contents

**PREFACE** .................................................................................................................. 5

**PART I. VULNERABILITY REPORTING WITH CLAIR ON RED HAT QUAY OVERVIEW** ............................................. 6

**CHAPTER 1. CLAIR FOR RED HAT QUAY** ........................................................................ 7

1.1. ABOUT CLAIR ........................................................................................................... 7

1.1.1. Clair releases
Clair 4.7.1
Clair 4.7

1.1.2. Clair supported dependencies
1.1.3. Clair containers

1.2. CLAIR VULNERABILITY DATABASES ....................................................................... 8

1.2.1. Information about Open Source Vulnerability (OSV) database for Clair

**CHAPTER 2. CLAIR CONCEPTS** .................................................................................. 10

2.1. CLAIR IN PRACTICE ............................................................................................... 10

2.1.1. Indexing
2.1.2. Matching
2.1.3. Notifications

2.2. CLAIR AUTHENTICATION .......................................................................................... 11

2.3. CLAIR UPDATERS .................................................................................................... 11

2.4. INFORMATION ABOUT CLAIR UPDATERS .......................................................... 11

2.5. CONFIGURING UPDATERS ....................................................................................... 13

2.5.1. Selecting specific updater sets
Configuring Clair for multiple updaters
Configuring Clair for Alpine
Configuring Clair for AWS
Configuring Clair for Debian
Configuring Clair for Clair CVSS
Configuring Clair for Oracle
Configuring Clair for Photon
Configuring Clair for SUSE
Configuring Clair for Ubuntu
Configuring Clair for OSV

2.5.2. Selecting updater sets for full Red Hat Enterprise Linux (RHEL) coverage

2.5.3. Advanced updater configuration
Configuring the alpine updater
Configuring the debian updater
Configuring the clair.cvss updater
Configuring the oracle updater
Configuring the photon updater
Configuring the rhel updater
Configuring the rhcc updater
Configuring the suse updater
Configuring the ubuntu updater
Configuring the osv updater

2.5.4. Disabling the Clair Updater component

2.6. CVE RATINGS FROM THE NATIONAL VULNERABILITY DATABASE ..................... 19

2.7. FEDERAL INFORMATION PROCESSING STANDARD (FIPS) READINESS AND COMPLIANCE ......................................................................................................................... 20

2.7.1. Enabling FIPS compliance

**PART II. CLAIR ON RED HAT QUAY** ............................................................................ 21
CHAPTER 3. SETTING UP CLAIR ON STANDALONE RED HAT QUAY DEPLOYMENTS .......................... 22
CHAPTER 4. CLAIR ON OPENShift CONTAINER PLATFORM ......................................................... 25
CHAPTER 5. TESTING CLAIR ..................................................................................................... 26
PART III. ADVANCED CLAIR CONFIGURATION ................................................................... 28
CHAPTER 6. UNMANAGED CLAIR CONFIGURATION ............................................................... 29
  6.1. RUNNING A CUSTOM CLAIR CONFIGURATION WITH AN UNMANAGED CLAIR DATABASE 29
  6.2. CONFIGURING A CUSTOM CLAIR DATABASE WITH AN UNMANAGED CLAIR DATABASE 29
CHAPTER 7. RUNNING A CUSTOM CLAIR CONFIGURATION WITH A MANAGED CLAIR DATABASE ... 32
  7.1. SETTING A CLAIR DATABASE TO MANAGED ................................................................. 32
  7.2. CONFIGURING A CUSTOM CLAIR DATABASE WITH A MANAGED CLAIR CONFIGURATION 32
CHAPTER 8. CLAIR IN DISCONNECTED ENVIRONMENTS ....................................................... 35
  8.1. SETTING UP CLAIR IN A DISCONNECTED OPENShift CONTAINER PLATFORM CLUSTER 35
     8.1.1. Installing the clairctl command line utility tool for OpenShift Container Platform deployments 35
     8.1.2. Retrieving and decoding the Clair configuration secret for Clair deployments on OpenShift Container Platform 35
     8.1.3. Exporting the updators bundle from a connected Clair instance 36
     8.1.4. Configuring access to the Clair database in the disconnected OpenShift Container Platform cluster 36
     8.1.5. Importing the updators bundle into the disconnected OpenShift Container Platform cluster 38
  8.2. SETTING UP A SELF-MANAGED DEPLOYMENT OF CLAIR FOR A DISCONNECTED OPENShift CONTAINER PLATFORM CLUSTER 38
     8.2.1. Installing the clairctl command line utility tool for a self-managed Clair deployment on OpenShift Container Platform 38
     8.2.2. Deploying a self-managed Clair container for disconnected OpenShift Container Platform clusters 38
     8.2.3. Exporting the updators bundle from a connected Clair instance 39
     8.2.4. Configuring access to the Clair database in the disconnected OpenShift Container Platform cluster 40
     8.2.5. Importing the updators bundle into the disconnected OpenShift Container Platform cluster 41
  8.3. MAPPING REPOSITORIES TO COMMON PRODUCT ENUMERATION INFORMATION 41
     8.3.1. Mapping repositories to Common Product Enumeration example configuration 42
CHAPTER 9. CLAIR CONFIGURATION OVERVIEW ................................................................. 43
  9.1. INFORMATION ABOUT USING CLAIR IN A PROXY ENVIRONMENT .............................. 43
  9.2. CLAIR CONFIGURATION REFERENCE ............................................................................ 44
  9.3. CLAIR GENERAL FIELDS .................................................................................................. 45
     Example configuration for general Clair fields 45
  9.4. CLAIR INDEXER CONFIGURATION FIELDS ..................................................................... 46
     Example indexer configuration 47
  9.5. CLAIR MATCHER CONFIGURATION FIELDS ................................................................... 47
     Example matcher configuration 49
  9.6. CLAIR MATCHERS CONFIGURATION FIELDS ................................................................. 49
     Example matchers configuration 50
  9.7. CLAIR UPDATERS CONFIGURATION FIELDS ................................................................. 50
     Example updators configuration 51
  9.8. CLAIR NOTIFIER CONFIGURATION FIELDS ................................................................... 51
     Example notifier configuration 52
     9.8.1. Clair webhook configuration fields ............................................................................. 53
         Example webhook configuration 53
     9.8.2. Clair amqp configuration fields ................................................................................. 53
         Example AMQP configuration 55
     9.8.3. Clair STOMP configuration fields .............................................................................. 55
Example STOMP configuration

9.9. CLAIR AUTHORIZATION CONFIGURATION FIELDS
Example authorization configuration

9.10. CLAIR TRACE CONFIGURATION FIELDS
Example trace configuration

9.11. CLAIR METRICS CONFIGURATION FIELDS
Example metrics configuration
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. ABOUT CLAIR

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

1.1.1. 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.

Clair 4.7.1

Clair 4.7.1 was released as part of Red Hat Quay 3.9.1. The following changes have been made:

- With this release, you can view unpatched vulnerabilities from Red Hat Enterprise Linux (RHEL) sources. If you want to view unpatched vulnerabilities, you can set the ignore_unpatched parameter to false. For example:

  ```yaml
  updaters:
  config:
    rhel:
      ignore_unpatched: false
  ```

  To disable this feature, you can set ignore_unpatched to true.

Clair 4.7

Clair 4.7 was released as part of Red Hat Quay 3.9, and includes support for the following features:

- Native support for indexing Golang modules and RubeGems in container images.
- Change to OSV.dev as the vulnerability database source for any programming language package managers.
  - This includes popular sources like GitHub Security Advisories or PyPA.
  - This allows offline capability.
- Use of pyup.io for Python and CRDA for Java is suspended.
- Clair now supports Java, Golang, Python, and Ruby dependencies.

1.1.2. Clair supported dependencies

Clair supports identifying and managing the following dependencies:

- Java
This means that it can analyze and report on the third-party libraries and packages that a project in these languages relies on to work correctly.

1.1.3. 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.

1.2. CLAIR VULNERABILITY DATABASES

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

- Ubuntu Oval database
- Debian Security Tracker
- 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
- Open Source Vulnerability (OSV) Database

For information about how Clair does security mapping with the different databases, see Claircore Severity Mapping.

1.2.1. Information about Open Source Vulnerability (OSV) database for Clair

Open Source Vulnerability (OSV) is a vulnerability database and monitoring service that focuses on tracking and managing security vulnerabilities in open source software.

OSV provides a comprehensive and up-to-date database of known security vulnerabilities in open source projects. It covers a wide range of open source software, including libraries, frameworks, and other components that are used in software development. For a full list of included ecosystems, see defined ecosystems.

Clair also reports vulnerability and security information for golang, java, and ruby ecosystems through the Open Source Vulnerability (OSV) database.
By leveraging OSV, developers and organizations can proactively monitor and address security vulnerabilities in open source components that they use, which helps to reduce the risk of security breaches and data compromises in projects.

For more information about OSV, see the OSV website.
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 plays a crucial role in understanding the makeup of a container image. In Clair, container image representations called "manifests." Manifests are used to comprehend the contents of the image’s layers. To streamline this process, Clair takes advantage of the fact that Open Container Initiative (OCI) manifests and layers are designed for content addressing, reducing repetitive tasks.

During indexing, a manifest that represents a container image is taken and broken down into its essential components. The indexer’s job is to uncover the image’s contained packages, its origin distribution, and the package repositories it relies on. This valuable information is then recorded and stored within Clair’s database. The insights gathered during indexing serve as the basis for generating a comprehensive vulnerability report. This report can be seamlessly transferred to a matcher node for further analysis and action, helping users make informed decisions about their container images' security.

2.1.2. Matching

With Clair, a matcher node is responsible for matching vulnerabilities to a provided index report.

Matchers are responsible for keeping the database of vulnerabilities up to date. Matchers 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 vulnerability report when queried. The vulnerability report summarizes both a manifest’s content and any vulnerabilities affecting the content.

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.

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.

## 2.2. 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:

   ```yaml
   auth:
   psk:
     key: >-
       MDQ4ODBINDAtNDc0ZC00MWUxLTlhMzAtOTk0MzEwMGQwYTMxGg==
     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.

### 2.4. INFORMATION ABOUT CLAIR UPDATERS

The following table provides details about each Clair updater, including the configuration parameter, a brief description, relevant URLs, and the associated components that they interact with. This list is not exhaustive, and some servers might issue redirects, while certain request URLs are dynamically constructed to ensure accurate vulnerability data retrieval.

For Clair, each updater is responsible for fetching and parsing vulnerability data related to a specific package type or distribution. For example, the Debian updater focuses on Debian-based Linux distributions, while the AWS updater focuses on vulnerabilities specific to Amazon Web Services’ Linux distributions. Understanding the package type is important for vulnerability management because different package types might have unique security concerns and require specific updates and patches.
NOTE

If you are using a proxy server in your environment with Clair’s updater URLs, you must identify which URL needs to be added to the proxy allowlist to ensure that Clair can access them unimpeded. Use the following table to add updater URLs to your proxy allowlist.

Table 2.1. Clair updater information

<table>
<thead>
<tr>
<th>Updater</th>
<th>Description</th>
<th>URLs</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpine</td>
<td>The Alpine updater is responsible for fetching and parsing vulnerability data related to packages in Alpine Linux distributions.</td>
<td>● <a href="https://secdb.alpinelinux.org/">https://secdb.alpinelinux.org/</a></td>
<td>Alpine Linux SecDB database</td>
</tr>
</tbody>
</table>
| aws     | The AWS updater is focused on AWS Linux-based packages, ensuring that vulnerability information specific to Amazon Web Services’ custom Linux distributions is kept up-to-date. | ● http://repo.us-west-2.amazonaws.com/2018.03/updates/x86_64/mirror.list  
● https://cdn.amazonaws.com/2/core/latest/x86_64/mirror.list  
● https://cdn.amazonaws.com/al2023/core/mirrors/latest/x86_64/mirror.list | Amazon Web Services (AWS) UpdateInfo |
| debian  | The Debian updater is essential for tracking vulnerabilities in packages associated with Debian-based Linux distributions. | ● https://deb.debian.org/  
● https://security-tracker.debian.org/tracker/data/json | Debian Security Tracker |
<p>| clair.cvss | The Clair Common Vulnerability Scoring System (CVSS) updater focuses on maintaining data about vulnerabilities and their associated CVSS scores. This is not tied to a specific package type but rather to the severity and risk assessment of vulnerabilities in general. | ● <a href="https://nvd.nist.gov/feeds/json/cve/1.1/">https://nvd.nist.gov/feeds/json/cve/1.1/</a> | National Vulnerability Database (NVD) feed for Common Vulnerabilities and Exposures (CVE) data in JSON format |
| oracle  | The Oracle updater is dedicated to Oracle Linux packages, maintaining data on vulnerabilities that affect Oracle Linux systems. | ● <a href="https://linux.oracle.com/security/oval/com.oracle.elsa-*e.xml.bz2">https://linux.oracle.com/security/oval/com.oracle.elsa-*e.xml.bz2</a> | Oracle Oval database |</p>
<table>
<thead>
<tr>
<th>Updater</th>
<th>Description</th>
<th>URLs</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>photon</td>
<td>The Photon updater deals with packages in VMware Photon OS.</td>
<td><a href="https://packages.vmware.com/photon/photon_oval_definitions/">https://packages.vmware.com/photon/photon_oval_definitions/</a></td>
<td>VMware Photon OS oval definitions</td>
</tr>
</tbody>
</table>
| rhel | The Red Hat Enterprise Linux (RHEL) updater is responsible for maintaining vulnerability data for packages in Red Hat’s Enterprise Linux distribution. | [https://access.redhat.com/security/cve/](https://access.redhat.com/security/cve/)  
| rhcc | The Red Hat Container Catalog (RHCC) updater is connected to Red Hat’s container images. This updater ensures that vulnerability information related to Red Hat’s containerized software is kept current. | [https://access.redhat.com/security/data/metrics/cvemap.xml](https://access.redhat.com/security/data/metrics/cvemap.xml) | Resource Handler Configuration Controller (RHCC) database |
| suse | The SUSE updater manages vulnerability information for packages in the SUSE Linux distribution family, including openSUSE, SUSE Enterprise Linux, and others. | [https://support.novell.com/security/oval/](https://support.novell.com/security/oval/) | SUSE Oval database |
| ubuntu | The Ubuntu updater is dedicated to tracking vulnerabilities in packages associated with Ubuntu-based Linux distributions. Ubuntu is a popular distribution in the Linux ecosystem. | [https://security-metadata.canonical.com/oval/com.ubuntu.*.cve.oval.xml](https://security-metadata.canonical.com/oval/com.ubuntu.*.cve.oval.xml)  
[https://api.launchpad.net/1.0/](https://api.launchpad.net/1.0/) | Ubuntu Oval Database |
| osv | The Open Source Vulnerability (OSV) updater specializes in tracking vulnerabilities within open source software components. OSV is a critical resource that provides detailed information about security issues found in various open source projects. | [https://osv-vulnerabilities.storage.googleapis.com/](https://osv-vulnerabilities.storage.googleapis.com/) | Open Source Vulnerabilities database |

### 2.5. CONFIGURING UPDATERs
Updaters can be configured by the `updaters.sets` key in your `clair-config.yaml` file.

**IMPORTANT**

- If the `sets` field is not populated, it defaults to using all sets. In using all sets, Clair tries to reach the URL or URLs of each updater. If you are using a proxy environment, you must add these URLs to your proxy allowlist.

- 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.

2.5.1. Selecting specific updater sets

Use the following references to select one, or multiple, updaters for your Red Hat Quay deployment.

Configuring Clair for multiple updaters

Multiple specific updaters

```yaml
#...
updaters:
  sets:
    - alpine
    - aws
    - osv
#...
```

Configuring Clair for Alpine

Alpine config.yaml example

```yaml
#...
updaters:
  sets:
    - alpine
#...
```

Configuring Clair for AWS

AWS config.yaml example

```yaml
#...
updaters:
  sets:
    - aws
#...
```

Configuring Clair for Debian

Debian config.yaml example

```yaml
#...
updaters:
```
sets:
  - debian
#

Configuring Clair for Clair CVSS

Clair CVSS config.yaml example

#...
updaters:
  sets:
    - clair.cvss
#

Configuring Clair for Oracle

Oracle config.yaml example

#...
updaters:
  sets:
    - oracle
#

Configuring Clair for Photon

Photon config.yaml example

#...
updaters:
  sets:
    - photon
#

Configuring Clair for SUSE

SUSE config.yaml example

#...
updaters:
  sets:
    - suse
#

Configuring Clair for Ubuntu

Ubuntu config.yaml example

#...
updaters:
  sets:
    - ubuntu
#
Configuring Clair for OSV

**OSV config.yaml example**

```yaml
#...
updaters:
  sets:
    - osv
#...
```

### 2.5.2. Selecting updater sets for full Red Hat Enterprise Linux (RHEL) coverage

For full coverage of vulnerabilities in Red Hat Enterprise Linux (RHEL), you must use the following updater sets:

- **rhel.** This updater ensures that you have the latest information on the vulnerabilities that affect RHEL.

- **rhcc.** This updater keeps track of vulnerabilities related to Red hat’s container images.

- **clair.cvss.** This updater offers a comprehensive view of the severity and risk assessment of vulnerabilities by providing Common Vulnerabilities and Exposures (CVE) scores.

- **osv.** This updater focuses on tracking vulnerabilities in open-source software components. This updater is recommended due to how common the use of Java and Go are in RHEL products.

**RHEL updaters example**

```yaml
#...
updaters:
  sets:
    - rhel
    - rhcc
    - clair.cvss
    - osv
#...
```

### 2.5.3. Advanced updater configuration

In some cases, users might want to configure updaters for specific behavior, for example, if you want to allowlist specific ecosystems for the Open Source Vulnerabilities (OSV) updaters.

Advanced updater configuration might be useful for proxy deployments or air gapped deployments. Configuration for specific updaters in these scenarios can be passed by putting a key underneath the `config` environment variable of the `updaters` object. Users should examine their Clair logs to double-check names.

The following YAML snippets detail the various settings available to some Clair updater

**IMPORTANT**

For more users, advanced updater configuration is unnecessary.

**Configuring the alpine updater**
Configuring the debian updater

```yaml
updaters:
  sets:
    - alpine
  config:
    alpine:
      url: https://secdb.alpinelinux.org/
```

Configuring the clair.cvss updater

```yaml
updaters:
  sets:
    - debian
  config:
    debian:
      mirror_url: https://deb.debian.org/
      json_url: https://security-tracker.debian.org/tracker/data/json
```

Configuring the oracle updater

```yaml
updaters:
  sets:
    - oracle
  config:
    oracle-2023-updater:
      url:
    oracle-2022-updater:
      url:
```

Configuring the photon updater

```yaml
updaters:
  sets:
    - photon
  config:
    photon:
      url: https://packages.vmware.com/photon/photon_oval_definitions/
```
Configuring the rhel updater

```yaml
updaters:
  sets:
    - rhel
  config:
    rhel:
      url: https://access.redhat.com/security/data/oval/v2/PULP_MANIFEST
      ignore_unpatched: true
```

1 Boolean. Whether to include information about vulnerabilities that do not have corresponding patches or updates available.

Configuring the rhcc updater

```yaml
updaters:
  sets:
    - rhcc
  config:
    rhcc:
      url: https://access.redhat.com/security/data/metrics/cvemap.xml
```

Configuring the suse updater

```yaml
updaters:
  sets:
    - suse
  config:
    suse:
      url: https://support.novell.com/security/oval/
```

Configuring the ubuntu updater

```yaml
updaters:
  config:
    ubuntu:
      url: https://api.launchpad.net/1.0/
      name: ubuntu
      force: 1
      - name: focal
        version: 20.04
```

1 Used to force the inclusion of specific distribution and version details in the resulting UpdaterSet, regardless of their status in the API response. Useful when you want to ensure that particular distributions and versions are consistently included in your updater configuration.
Specifies the distribution name that you want to force to be included in the UpdaterSet.

Specifies the version of the distribution you want to force into the UpdaterSet.

Configuring the osv updater

```
#...
updaters:
  sets:
    - osv
config:
osv:
  url: https://osv-vulnerabilities.storage.googleapis.com/
  allowlist: 1
    - npm
    - pypi
#...
```

The list of ecosystems to allow. When left unset, all ecosystems are allowed. Must be lowercase. For a list of supported ecosystems, see the documentation for defined ecosystems.

2.5.4. 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
#...
```

2.6. 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 presents the distribution’s score by default. For example:
This differs from the previous interface, which would only display the following information:

2.7. 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 FIPS by providing a FIPS mode, in which the system only allows usage of specific FIPS-validated cryptographic modules like openssl. This ensures FIPS compliance.

2.7.1. Enabling FIPS compliance

Use the following procedure to enable FIPS compliance on your Red Hat Quay deployment.

Prerequisite

- If you are running a standalone deployment of Red Hat Quay, your Red Hat Enterprise Linux (RHEL) deployment is version 8 or later and FIPS-enabled.

- If you are using the Red Hat Quay Operator, OpenShift Container Platform is version 4.10 or later.

- Your Red Hat Quay version is 3.5.0 or later.

- You have administrative privileges for your Red Hat Quay deployment.

Procedure

- In your Red Hat Quay config.yaml file, set the FEATURE_FIPS configuration field to true. For example:

  ```yaml
  FEATURE_FIPS = true
  ```

  With FEATURE_FIPS set to true, Red Hat Quay runs using FIPS-compliant hash functions.
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 3. 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:


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

   CREATE EXTENSION

   NOTE

   Clair requires the uuid-ossp 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-ossp" extension. (SQLSTATE 42501).

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

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

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 --name quay_config \\   -p 80:8080 -p 443:8443 \

22
6. Log in to the configuration tool and click Enable Security Scanning in the Security Scanner section of the UI.

7. Set the HTTP endpoint for Clair using a port that is not already in use on the quay-server system, for example, 8081.

8. 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 any notification support.
  - Enable Security Scanning

- **Security Scanner Endpoint**: http://quay-server:8081
  - The HTTP URL at which the security scanner is running.

- **Security Scanner PSK**: MJTUvzA4YZD82zJoX2Z
  - Clair Pre-Shared Key. Make sure to include this value in your Clair config.

9. Validate and download the config.yaml file for Red Hat Quay, and then stop the Quay container that is running the configuration editor.

10. 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/
    ```

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

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

12. Change into the Clair configuration folder:

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

13. 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
        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:v3.9.5
```

**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 4. CLAIR ON OPENSFILE 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 5. 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:

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

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

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

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

   ```sh
   $ 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

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

Report details
NOTE

In some cases, Clair shows duplicate reports on images, for example, `ubi8/nodejs-12` or `ubi8/nodejs-16`. This occurs because vulnerabilities with same name are for different packages. This behavior is expected with Clair vulnerability reporting and will not be addressed as a bug.
PART III. ADVANCED CLAIRE CONFIGURATION

Use this section to configure advanced Clair features.
CHAPTER 6. 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.

6.1. 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`:

```yaml
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
```

6.2. 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:

   ```bash
   $ 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**

   ```yaml
   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](https://github.com/openshift/clair-operator).  

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

   ```yaml
   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:

```bash
$ 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 7. 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 an 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 an disconnected environment, the airgap parameter of your clair-config.yaml must be set to true.
- If you are running Red Hat Quay in an disconnected environment, you should disable all updater components.

7.1. 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:

```yaml
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
```

7.2. 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:

```
```

**Example Clair config.yaml file**

```yaml
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:

```yaml
apiVersion: v1
class: 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:

```bash
$ 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. 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 important 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”.

8.1. SETTING UP CLAIR IN A DISCONNECTED OPENSШIFT CONTAINER PLATFORM CLUSTER

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

8.1.1. 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

8.1.2. 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:

   ```bash
   $ 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:

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

**8.1.3. 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:

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

**8.1.4. 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
   
   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
   ...                                   
   ```

   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  
   scanlock_retry: 10
   layer_scan_concurrency: 5
   migrations: true
   scanner:
      repo:
         rhel-repository-scanner:
            repo2cpe_mapping_file: /data/cpe-map.json
         rhel_containerscanner:
            name2repos_mapping_file: /data/repo-map.json
   ```

   1. 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".
8.15. 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
  ```

8.2. SETTING UP A SELF-MANAGED DEPLOYMENT OF CLAIR FOR A DISCONNECTED OPENShift CONTAINER PLATFORM CLUSTER

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

8.2.1. 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
   ```

8.2.2. 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:
   ```bash
   $ mkdir /etc/clairv4/config/
   ```

2. Create a Clair configuration file with the `disable_updaters` parameter set to `true`, for example:
   ```yaml
---
indexer:
  airgap: true
---
matcher:
  disable_updaters: true
---
```

3. Start Clair by using the container image, mounting in the configuration from the file you created:
   ```bash
   $ 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:v3.9.5
   ```

8.2.3. 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:
  ```bash
  $ ./clairctl --config ./config.yaml export-updaters updates.gz
  ```
8.2.4. 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:

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

   **Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>example-registry-clair-app</td>
<td>ClusterIP</td>
<td>172.30.224.93</td>
<td>&lt;none&gt;</td>
<td>80/TCP,8089/TCP 4d21h</td>
</tr>
<tr>
<td>example-registry-clair-postgres</td>
<td>ClusterIP</td>
<td>172.30.246.88</td>
<td>&lt;none&gt;</td>
<td>5432/TCP      4d21h</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

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

   **Replace the value of the host in the multiple connstring fields with localhost.**
8.2.5. 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
  ```

8.3. 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 8.1. Clair CPE mapping files

<table>
<thead>
<tr>
<th>CPE</th>
<th>Link to JSON mapping file</th>
</tr>
</thead>
<tbody>
<tr>
<td>repos2cpe</td>
<td>Red Hat Repository-to-CPE JSON</td>
</tr>
<tr>
<td>names2repos</td>
<td>Red Hat Name-to-Repos JSON</td>
</tr>
</tbody>
</table>
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 **unmanaged**. Then, Clair must be deployed manually, setting the configuration to load a local copy of the mapping file.

### 8.3.1. 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:

```yaml
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 9. 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.

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

9.1. INFORMATION ABOUT USING CLAIR IN A PROXY ENVIRONMENT

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

- **HTTP_PROXY**
  
  $ export http://<user_name>:<password>@<proxy_host>:<proxy_port>

- **HTTPS_PROXY**
  
  $ export https://<user_name>:<password>@<proxy_host>:<proxy_port>

- **SSL_CERT_DIR**
  
  $ export SSL_CERT_DIR=/<path>/to/<ssl>/certificates

If you are using a proxy server in your environment with Clair’s updater URLs, you must identify which URL needs to be added to the proxy allowlist to ensure that Clair can access them unimpeded. For example, the osv updater requires access to https://osv-vulnerabilities.storage.googleapis.com to fetch ecosystem data dumps. In this scenario, the URL must be added to the proxy allowlist. For a full list of updater URLs, see “Clair updater URLs”.

You must also ensure that the standard Clair URLs are added to the proxy allowlist:

- **https://search.maven.org/solrsearch/select**

- **https://catalog.redhat.com/api/containers/**

- **https://access.redhat.com/security/data/metrics/repository-to-cpe.json**

- **https://access.redhat.com/security/data/metrics/container-name-repos-map.json**

When configuring the proxy server, take into account any authentication requirements or specific proxy settings needed to enable seamless communication between Clair and these URLs. By thoroughly documenting and addressing these considerations, you can ensure that Clair functions effectively while routing its updater traffic through the proxy.
9.2. CLAIR CONFIGURATION REFERENCE

The following YAML shows an example Clair configuration:

```yaml
http_listen_addr: ""
introspection_addr: ""
log_level: ""
tls: {}
indexer:
  connstring: ""
  scanlock_retry: 0
  layer_scan_concurrency: 5
  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: ""
```
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.

9.3. CLAIR GENERAL FIELDS

The following table describes the general configuration fields available for a Clair deployment.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>http_listen_addr</td>
<td>String</td>
<td>Configures where the HTTP API is exposed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: :6060</td>
</tr>
<tr>
<td>introspection_addr</td>
<td>String</td>
<td>Configures where Clair’s metrics and health endpoints are exposed.</td>
</tr>
<tr>
<td>log_level</td>
<td>String</td>
<td>Sets the logging level. Requires one of the following strings: debug-color, debug, info, warn, error, fatal, panic</td>
</tr>
<tr>
<td>tls</td>
<td>String</td>
<td>A map containing the configuration for serving the HTTP API of TLS/SSL and HTTP/2.</td>
</tr>
<tr>
<td>.cert</td>
<td>String</td>
<td>The TLS certificate to be used. Must be a full-chain certificate.</td>
</tr>
</tbody>
</table>

Example configuration for general Clair fields
The following example shows a Clair configuration.

Example configuration for general Clair fields

```yaml
# ...
http_listen_addr: 0.0.0.0:6060
introspection_addr: 0.0.0.0:8089
log_level: info
# ...
```
## 9.4. CLAIR INDEXER CONFIGURATION FIELDS

The following table describes the configuration fields for Clair’s *indexer* component.

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

The following example shows a hypothetical indexer configuration for Clair.

#### Example indexer configuration

```yaml
# ...
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
# ...
```

### 9.5. CLAIR MATCHER CONFIGURATION FIELDS

The following table describes the configuration fields for Clair’s `matcher` component.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>matcher</td>
<td>Object</td>
<td>Provides Clair matcher node configuration.</td>
</tr>
</tbody>
</table>

**NOTE**

Differs from `matchers` configuration fields.
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.cache_age</td>
<td>String</td>
<td>Controls how long users should be hinted to cache responses for.</td>
</tr>
<tr>
<td>.connstring</td>
<td>String</td>
<td>A Postgres connection string. Accepts format as a URL or libpq connection string.</td>
</tr>
<tr>
<td>.max_conn_pool</td>
<td>Integer</td>
<td>Limits the database connection pool size.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clair allows for a custom connection pool size. This number directly sets how many active database connections are allowed concurrently. This parameter will be ignored in a future version. Users should configure this through the connection string.</td>
</tr>
<tr>
<td>.indexer_addr</td>
<td>String</td>
<td>A matcher contacts an indexer to create a vulnerability report. The location of this indexer is required. Defaults to 30m.</td>
</tr>
<tr>
<td>.migrations</td>
<td>Boolean</td>
<td>Whether matcher nodes handle migrations to their databases.</td>
</tr>
<tr>
<td>.period</td>
<td>String</td>
<td>Determines how often updates for new security advisories take place.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 30m.</td>
</tr>
<tr>
<td>.disable_updaters</td>
<td>Boolean</td>
<td>Whether to run background updates or not.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: False</td>
</tr>
</tbody>
</table>
### 9.6. CLAIR MATCHERS CONFIGURATION FIELDS

The following table describes the configuration fields for Clair’s **matchers** component.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| .update_retention   | Integer       | 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. Defaults to **10m**.
|                     |               | If a value of less than **0** is provided, garbage collection is disabled. **2** is the minimum value to ensure updates can be compared to notifications. |

#### Example matcher configuration

```
# ...
matcher:
  connstring: >-
    host=<DB_HOST> port=5432 dbname=<matcher> user=<DB_USER> password=D<B_PASS>
    sslmode=verify-ca sslcert=/etc/clair/ssl/cert.pem sslkey=/etc/clair/ssl/key.pem
    sslrootcert=/etc/clair/ssl/ca.pem
  indexer_addr: http://clair-v4/
  disable_updaters: false
  migrations: true
  period: 6h
  update_retention: 2
# ...
```
### .names

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.names</td>
<td>String</td>
<td>A list of string values informing the matcher factory about enabled matchers. If value is set to <code>null</code>, the default list of matchers run. The following strings are accepted: <code>alpine-matcher</code>, <code>aws-matcher</code>, <code>debian-matcher</code>, <code>gobin</code>, <code>java-maven</code>, <code>oracle</code>, <code>photon</code>, <code>python</code>, <code>rhel</code>, <code>rhel-container-matcher</code>, <code>ruby</code>, <code>suse</code>, <code>ubuntu-matcher</code></td>
</tr>
</tbody>
</table>

### .config

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.config</td>
<td>String</td>
<td>Provides configuration to a specific matcher. A map keyed by the name of the matcher containing a sub-object which will be provided to the matchers factory constructor. For example:</td>
</tr>
</tbody>
</table>

#### Example matchers configuration

The following example shows a hypothetical Clair deployment that only requires only the `alpine`, `aws`, `debian`, `oracle` matchers.

```
# ...
matchers:
  names:
  - "alpine-matcher"
  - "aws"
  - "debian"
  - "oracle"
# ...
```

### 9.7. CLAIR UPDATER CONFIGURATION FIELDS

The following table describes the configuration fields for Clair’s `updaters` component.

#### Table 9.2. Updaters configuration fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>updaters</td>
<td>Object</td>
<td>Provides configuration for the matcher’s update manager.</td>
</tr>
</tbody>
</table>
A list of values informing the update manager which updaters to run.

If value is set to `null`, the default set of updaters runs the following: `alpine`, `aws`, `clair.cvss`, `debian`, `oracle`, `photon`, `osv`, `rhel`, `rhcc`, `suse`, `ubuntu`.

If left blank, zero updaters run.

---

### Example updaters configuration

In the following configuration, only the `rhel` set is configured. The `ignore_unpatched` variable, which is specific to the `rhel` updater, is also defined.

#### Example updaters configuration

```yaml
# ...
updaters:
  sets:
    - rhel
  config:
    rhel:
      ignore_unpatched: false
# ...
```

---

### 9.8. CLAIR NOTIFIER CONFIGURATION FIELDS

The general notifier configuration fields for Clair are listed below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifier</td>
<td>Object</td>
<td>Provides Clair notifier node configuration.</td>
</tr>
<tr>
<td>.connstring</td>
<td>String</td>
<td>Postgres connection string. Accepts format as URL, or libpq connection string.</td>
</tr>
</tbody>
</table>
### Field | Type | Description
--- | --- | ---
.migrations | Boolean | Whether notifier nodes handle migrations to their database.
.indexer_addr | String | A notifier contacts an indexer to create or obtain manifests affected by vulnerabilities. The location of this indexer is required.
.matcher_addr | String | A notifier contacts a matcher to list update operations and acquire diffs. The location of this matcher is required.
.poll_interval | String | The frequency at which the notifier will query a matcher for update operations.
.delivery_interval | String | The frequency at which the notifier attempts delivery of created, or previously failed, notifications.
.disable_summary | Boolean | Controls whether notifications should be summarized to one per manifest.

**Example notifier configuration**
The following **notifier** snippet is for a minimal configuration.

```
# ...
notifier:
  connstring: >-
    host=DB_HOST port=5432 dbname=notifier user=DB_USER password=DB_PASS
    sslmode=verify-ca sslcert=/etc/clair/ssl/cert.pem sslkey=/etc/clair/ssl/key.pem
    sslrootcert=/etc/clair/ssl/ca.pem
  indexer_addr: http://clair-v4/
  matcher_addr: http://clair-v4/
  delivery_interval: 5s
  migrations: true
  poll_interval: 15s
  webhook:
    target: "http://webhook/"
    callback: "http://clair-notifier/notifier/api/v1/notifications"
```
9.8.1. Clair webhook configuration fields

The following webhook fields are available for the Clair notifier environment.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.webhook</td>
<td>Object</td>
<td>Configures the notifier for webhook delivery.</td>
</tr>
<tr>
<td>.webhook.target</td>
<td>String</td>
<td>URL where the webhook will be delivered.</td>
</tr>
<tr>
<td>.webhook.callback</td>
<td>String</td>
<td>The callback URL where notifications can be retrieved. The notification ID will be appended to this URL. This will typically be where the Clair notifier is hosted.</td>
</tr>
<tr>
<td>.webhook.headers</td>
<td>String</td>
<td>A map associating a header name to a list of values.</td>
</tr>
</tbody>
</table>

Example webhook configuration

```yaml
headers: ""
amqp: null
stomp: null
# ...
```

9.8.2. Clair amqp configuration fields

The following Advanced Message Queuing Protocol (AMQP) fields are available for the Clair notifier environment.
### .amqp

**Object**

Configures the notifier for AMQP delivery.

*NOTE* 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.

### .amqp.direct

**Boolean**

If `true`, the notifier will deliver individual notifications (not a callback) to the configured AMQP broker.

### .amqp.rollup

**Integer**

When `.amqp.direct` is set to `true`, this value informs the notifier of how many notifications to send in a direct delivery. For example, if `direct` is set to `true`, and `.amqp.rollup` is set to `5`, the notifier delivers no more than 5 notifications in a single JSON payload to the broker. Setting the value to `0` effectively sets it to `1`.

### .amqp.exchange

**Object**

The AMQP exchange to connect to.

### .amqp.exchange.name

**String**

The name of the exchange to connect to.

### .amqp.exchange.type

**String**

The type of the exchange. Typically one of the following: `direct`, `fanout`, `topic`, `headers`.

### .amqp.exchange.durability

**Boolean**

Whether the configured queue is durable.

### .amqp.exchange.auto_delete

**Boolean**

Whether the configured queue uses an `auto_delete_policy`.

### .amqp.routing_key

**String**

The name of the routing key each notification is sent with.
<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.amqp.callback</td>
<td>String</td>
<td>If <code>amqp.direct</code> is set to <code>false</code>, this URL is provided in the notification callback sent to the broker. This URL should point to Clair’s notification API endpoint.</td>
</tr>
<tr>
<td>.amqp.uris</td>
<td>String</td>
<td>A list of one or more AMQP brokers to connect to, in priority order.</td>
</tr>
<tr>
<td>.amqp.tls</td>
<td>Object</td>
<td>Configures TLS/SSL connection to an AMQP broker.</td>
</tr>
<tr>
<td>.amqp.tls.root_ca</td>
<td>String</td>
<td>The filesystem path where a root CA can be read.</td>
</tr>
<tr>
<td>.amqp.tls.cert</td>
<td>String</td>
<td>The filesystem path where a TLS/SSL certificate can be read.</td>
</tr>
<tr>
<td>[NOTE]</td>
<td></td>
<td>Clair also allows <code>SSL_CERT_DIR</code>, as documented for the Go <code>crypto/x509</code> package.</td>
</tr>
<tr>
<td>.amqp.tls.key</td>
<td>String</td>
<td>The filesystem path where a TLS/SSL private key can be read.</td>
</tr>
</tbody>
</table>

Example AMQP configuration
The following example shows a hypothetical AMQP configuration for Clair.

```
# ...
notifier:
# ...
amqp:
  exchange:
    name: ""
    type: "direct"
    durable: true
    auto_delete: false
  uris: ["amqp://user:pass@host:10000/vhost"]
  direct: false
  routing_key: "notifications"
  callback: "http://clair-notifier/notifier/api/v1/notifications"
  tls:
    root_ca: "optional/path/to/rootca"
    cert: "madatory/path/to/cert"
    key: "madatory/path/to/key"
# ...
```

9.8.3. Clair STOMP configuration fields
The following Simple Text Oriented Message Protocol (STOMP) fields are available for the Clair notifier environment.

<table>
<thead>
<tr>
<th><strong>.stomp</strong></th>
<th><strong>Object</strong></th>
<th><strong>Configures the notifier for STOMP delivery.</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>.stomp.direct</strong></td>
<td>Boolean</td>
<td>If <strong>true</strong>, the notifier delivers individual notifications (not a callback) to the configured STOMP broker.</td>
</tr>
<tr>
<td><strong>.stomp.rollup</strong></td>
<td>Integer</td>
<td>If <strong>stomp.direct</strong> is set to <strong>true</strong>, this value limits the number of notifications sent in a single direct delivery. For example, if <strong>direct</strong> is set to <strong>true</strong>, and <strong>rollup</strong> is set to <strong>5</strong>, the notifier delivers no more than <strong>5</strong> notifications in a single JSON payload to the broker. Setting the value to <strong>0</strong> effectively sets it to <strong>1</strong>.</td>
</tr>
<tr>
<td><strong>.stomp.callback</strong></td>
<td>String</td>
<td>If <strong>stomp.callback</strong> is set to <strong>false</strong>, the provided URL in the notification callback is sent to the broker. This URL should point to Clair’s notification API endpoint.</td>
</tr>
<tr>
<td><strong>.stomp.destination</strong></td>
<td>String</td>
<td>The STOMP destination to deliver notifications to.</td>
</tr>
<tr>
<td><strong>.stomp.uris</strong></td>
<td>String</td>
<td>A list of one or more STOMP brokers to connect to in priority order.</td>
</tr>
<tr>
<td><strong>.stomp.tls</strong></td>
<td>Object</td>
<td>Configured TLS/SSL connection to STOMP broker.</td>
</tr>
<tr>
<td><strong>.stomp.tls.root_ca</strong></td>
<td>String</td>
<td>The filesystem path where a root CA can be read.</td>
</tr>
<tr>
<td><strong>.stomp.tls.cert</strong></td>
<td>String</td>
<td>The filesystem path where a TLS/SSL certificate can be read.</td>
</tr>
<tr>
<td><strong>.stomp.tls.key</strong></td>
<td>String</td>
<td>The filesystem path where a TLS/SSL private key can be read.</td>
</tr>
</tbody>
</table>

[NOTE] ==== Clair also respects **SSL_CERT_DIR**, as documented for the Go **crypto/x509** package. ==== 
## 9.9. CLAIR AUTHORIZATION CONFIGURATION FIELDS

The following authorization configuration fields are available for Clair.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth</td>
<td>Object</td>
<td>Defines Clair’s external and intra-service JWT based authentication. If multiple <strong>auth</strong> mechanisms are defined, Clair picks one. Currently, multiple mechanisms are unsupported.</td>
</tr>
<tr>
<td>.psk</td>
<td>String</td>
<td>Defines pre-shared key authentication.</td>
</tr>
</tbody>
</table>
A shared base64 encoded key distributed between all parties signing and verifying JWTs.

A list of JWT issuers to verify. An empty list accepts any issuer in a JWT claim.

Example authorization configuration

The following `authorization` snippet is for a minimal configuration.

```
# ...
auth:
psk:
  key: MTU5YzA4Y2ZkNzJoMQ==
  iss: ["quay"]
# ...
```

9.10. CLAIR TRACE CONFIGURATION FIELDS

The following trace configuration fields are available for Clair.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.trace</td>
<td>Object</td>
<td>Defines distributed tracing configuration based on OpenTelemetry.</td>
</tr>
<tr>
<td>.name</td>
<td>String</td>
<td>The name of the application traces will belong to.</td>
</tr>
<tr>
<td>.probability</td>
<td>Integer</td>
<td>The probability a trace will occur.</td>
</tr>
<tr>
<td>.jaeger</td>
<td>Object</td>
<td>Defines values for Jaeger tracing.</td>
</tr>
<tr>
<td>.jaeger.agent</td>
<td>Object</td>
<td>Defines values for configuring delivery to a Jaeger agent.</td>
</tr>
<tr>
<td>.jaeger.agent.endpoint</td>
<td>String</td>
<td>An address in the <code>&lt;host&gt;</code>: <code>&lt;post&gt;</code> syntax where traces can be submitted.</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>.jaeger.collector</td>
<td>Object</td>
<td>Defines values for configuring delivery to a Jaeger collector.</td>
</tr>
<tr>
<td>.jaeger.collector.endpoint</td>
<td>String</td>
<td>An address in the \texttt{&lt;host&gt;:&lt;post&gt;} syntax where traces can be submitted.</td>
</tr>
<tr>
<td>.jaeger.collector.username</td>
<td>String</td>
<td>A Jaeger username.</td>
</tr>
<tr>
<td>.jaeger.collector.password</td>
<td>String</td>
<td>A Jaeger password.</td>
</tr>
<tr>
<td>.jaeger.service_name</td>
<td>String</td>
<td>The service name registered in Jaeger.</td>
</tr>
<tr>
<td>.jaeger.tags</td>
<td>String</td>
<td>Key-value pairs to provide additional metadata.</td>
</tr>
<tr>
<td>.jaeger.buffer_max</td>
<td>Integer</td>
<td>The maximum number of spans that can be buffered in memory before they are sent to the Jaeger backend for storage and analysis.</td>
</tr>
</tbody>
</table>

**Example trace configuration**
The following example shows a hypothetical trace configuration for Clair.

```yaml
# ...
trace:
  name: "jaeger"
  probability: 1
  jaeger:
    agent:
      endpoint: "localhost:6831"
    service_name: "clair"
# ...
```

### 9.11. CLAIR METRICS CONFIGURATION FIELDS

The following metrics configuration fields are available for Clair.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
### Example metrics configuration
The following example shows a hypothetical metrics configuration for Clair.

```
# ...
metrics:
  name: "prometheus"
  prometheus:
    endpoint: "/metricsz"
# ...
```