Red Hat Quay 3

Red Hat Quay の Clair による脆弱性レポート

Red Hat Quay の Clair による脆弱性レポート

Last Updated: 2024-04-05

Red Hat Quay 3 Red Hat Quay の Clair による脆弱性レポート

Red Hat Quay の Clair による脆弱性レポート

法律上の通知

概要

Clair のスタートガイド

目次

はじめに

このガイドでは、Red Hat Quay の Clair の概要、スタンドアロンの Red Hat Quay および Operator デプロイメントでの Clair の実行、および高度な Clair 設定を説明します。

パート I. RED HAT QUAY の概要に関する CLAIR による脆弱性レポート

このガイドでは、Red Hat Quay での Clair の主な目的と概念を説明します。また、Clair のリリースと公式の Clair コンテナーの場所に関する情報も含まれます。

第1章 CLAIR セキュリティースキャナー

Clair v4 (Clair) は、静的コード分析を活用してイメージコンテンツを解析し、コンテンツに影響を与える脆弱性を報告するオープンソースアプリケーションです。Clair は Red Hat Quay にパッケージ化されており、スタンドアロンと Operator デプロイメントの両方で使用できます。エンタープライズ環境に合わせてコンポーネントを個別にスケーリングできる、非常にスケーラブルな設定で実行できます。

1.1. CLAIR について

Clair は、National Vulnerability Database (NVD) の Common Vulnerability Scoring System (CVSS) データを使用して脆弱性データを補完します。NVD は、さまざまなソフトウェアコンポーネントやシステムの既知の脆弱性やセキュリティー問題など、セキュリティー関連情報を提供する米国政府のリポジトリーです。NVD のスコアを使用することは、Clair にとって次の利点があります。

- **データの同期**。Clair は、脆弱性データベースを NVD と定期的に同期できます。これにより、最新の脆弱性データが確実に保持されます。
- **照合と補完**。Clair は、コンテナーイメージ内で発見した脆弱性のメタデータと識別子を NVD のデータと比較します。このプロセスでは、Common Vulnerabilities and Exposures (CVE) ID などの一意の識別子と NVD 内のエントリーの照合を実行します。一致するものが見つかった場合、Clair は重大度スコア、説明、リファレンスなど、NVD から詳細情報を追加して、脆弱性情報を補完することができます。
- **重大度スコア**。NVD は、Common Vulnerability Scoring System (CVSS) スコアなどの重大度スコアを脆弱性に割り当て、各脆弱性に関連する潜在的な影響とリスクを示します。NVD の重大度スコアを組み込むことにより、Clair は検出した脆弱性の重大さに関するコンテキストをより多く提供できます。

Clair が NVD に基づいて脆弱性を発見した場合、コンテナーイメージ内で検出された脆弱性の重大度と潜在的な影響について、標準化された詳細な評価が、UI を通じてユーザーに報告されます。CVSS の補完データは、Clair に次の利点を提供します。

- **脆弱性の優先順位付け**。CVSS スコアを利用することで、ユーザーは重大度に基づいて脆弱性に優先順位を付け、最も重要な問題に最初に対処できるようになります。
- **リスクの評価**。CVSS スコアは、コンテナー化されたアプリケーションに対する脆弱性の潜在的なリスクを Clair ユーザーが理解するのに役立ちます。
- **重大度の伝達**。CVSS スコアを使用すると、Clair ユーザーはチームおよび組織全体に脆弱性の重大度を標準化された方法で伝達できます。
- **修復戦略の通知**。CVSS の補完データは、Quay.io のユーザーが適切な修復戦略を策定する際に役立ちます。
- **コンプライアンスとレポート**。Clair によって生成されるレポートに CVSS データを統合すると、セキュリティーの脆弱性に対処し、業界の標準と規制に準拠するという組織的な取り組みを示すのに役立ちます。

1.1.1. Clair のリリース

Clair の新しいバージョンは定期的にリリースされます。Clair のビルドに必要なソースコードは、アーカイブとしてパッケージ化され、各リリースに添付されています。Clair のリリースは、Clair のリリース にあります。

リリースアーティファクトには、オープンホストを使用してインターネットからアップデーターデータを取得する **clairctl** コマンドラインインターフェイスツールも含まれています。

Clair 4.7.1
Clair 4.7.1 は Red Hat Quay 3.9.1 の一部としてリリースされました。次の変更が加えられました。

- このリリースでは、Red Hat Enterprise Linux (RHEL) ソースからパッチ未適用の脆弱性を表示できます。パッチ未適用の脆弱性を表示する場合は、**ignore_unpatched** パラメーターを **false** に設定します。以下に例を示します。

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

  この機能を無効にするには、**ignore_unpatched** を **true** に設定します。

Clair 4.7
Clair 4.7 は Red Hat Quay 3.9 の一部としてリリースされ、以下の機能のサポートが含まれています。

- コンテナーイメージ内の Golang モジュールと RubeGem のインデックス作成のネイティブサポート。
- プログラミング言語パッケージマネージャーの脆弱性データベースソースとして OSV.dev に変更します。
  - これには、GitHub Security Advisories や PyPA などの一般的なソースが含まれます。
  - これにより、オフライン機能が利用できます。
- Python 用の pyup.io および Java 用の CRDA の使用は一時停止されています。
- Clair は、Java、Golang、Python、および Ruby の依存関係をサポートするようになりました。

1.1.2. Clair 脆弱性データベース

Clair は、次の脆弱性データベースを使用して、イメージの問題を報告します。

- Ubuntu Oval データベース
- Debian Security Tracker
- Red Hat Enterprise Linux (RHEL) Oval データベース
- SUSE Oval データベース
- Oracle Oval データベース
- アルパイン SecDB データベース
- VMware Photon OS データベース
- Amazon Web Services (AWS) UpdateInfo
- Open Source Vulnerability (OSV) Database

1.1.3. Clair がサポートする依存関係

Clair は、次の依存関係の特定と管理をサポートしています。

- Java
- golang
- Python
- Ruby

そのため、Clair はこれらの言語のプロジェクトが正しく動作するために依存しているサードパーティーのライブラリーとパッケージを分析して報告できます。

Clair でサポートされていない言語のパッケージを含むイメージがリポジトリーにプッシュされると、そのパッケージに対して脆弱性スキャンを実行することができません。ユーザーには、サポートされていない依存関係やパッケージに関する分析レポートやセキュリティーレポートは表示されません。そのため、次のような影響を考慮する必要があります。

- **セキュリティーリスク**。脆弱性がスキャンされていない依存関係またはパッケージがあると、組織にセキュリティーリスクが発生する可能性があります。
- **コンプライアンスの問題**。組織に特定のセキュリティー要件またはコンプライアンス要件がある場合、スキャンされていない、または一部しかスキャンされていないコンテナーイメージにより、特定の規制への違反が発生する可能性があります。

注記

スキャンされたイメージにはインデックスが付けられ、脆弱性レポートが作成されますが、サポートされていない特定の言語のデータがレポートから省略されている場合があります。たとえば、コンテナーイメージに Lua アプリケーションが含まれている場合、Clair はそれを検出しないため、Clair からフィードバックは提供されません。Clair はコンテナーイメージで使用されている他の言語を検出し、それらの言語に対して検出された CVE を表示します。そのため、Clair イメージは、Clair がサポートする言語に基づいて **完全にスキャン** されます。

1.1.4. Clair コンテナー

Red Hat Quay にバンドルされている公式のダウンストリーム Clair コンテナーは、Red Hat Ecosystem Catalog にあります。

公式のアップストリームコンテナーはパッケージ化され、Quay.io/projectquay/clair でコンテナーとしてリリースされます。最新のタグは、Git 開発ブランチを追跡します。バージョンタグは、対応するリリースから構築されます。

1.2. CLAIR の重大度のマッピング

Clair は包括的なアプローチにより脆弱性を評価および管理します。その重要な機能の 1 つは、セキュリティーデータベースの重大度文字列の正規化です。これは、脆弱性の重大度を事前定義された値のセットにマッピングすることで、重大度の評価を効率化するプロセスです。このマッピングにより、クライアントは、各セキュリティーデータベースの固有の複雑な重大度文字列を解読することなく、脆弱性の重大度に効率よく対応できます。これらのマッピングされた重大度文字列は、それぞれのセキュリティーデータベース内にあるものと整合しているため、脆弱性評価の一貫性と正確性が確保されます。

1.2.1. Clair の重大度文字列

Clair は、次の重大度文字列をユーザーに通知します。

- Unknown
- Negligible
- Low
- Medium
- High
- Critical

これらの重大度文字列は、関連するセキュリティーデータベース内にある文字列と似ています。

Alpine のマッピング
Alpine SecDB データベースは重大度情報を提供しません。すべての脆弱性の重大度が Unknown になります。

Alpine の重大度	Clair の重大度
*	Unknown

AWS のマッピング
AWS UpdateInfo データベースが重大度情報を提供します。

AWS の重大度	Clair の重大度
low	Low
medium	Medium
important	High
critical	Critical

Debian のマッピング
Debian Oval データベースが重大度情報を提供します。

Debian の重大度	Clair の重大度
*	Unknown
Unimportant	Low
Low	Medium
Medium	High

Debian の重大度	Clair の重大度
High	Critical

Oracle のマッピング
Oracle Oval データベースが重大度情報を提供します。

Oracle の重大度	Clair の重大度
該当なし	Unknown
LOW	Low
MODERATE	Medium
IMPORTANT	High
CRITICAL	Critical

RHEL のマッピング
RHEL Oval データベースが重大度情報を提供します。

RHEL の重大度	Clair の重大度
なし	Unknown
Low	Low
Moderate	Medium
Important	High
Critical	Critical

SUSE のマッピング
SUSE Oval データベースが重大度情報を提供します。

重大度	Clair の重大度
なし	Unknown
Low	Low
Moderate	Medium

重大度	Clair の重大度
Important	High
Critical	Critical

Ubuntu のマッピング
Ubuntu Oval データベースが重大度情報を提供します。

重大度	Clair の重大度
Untriaged	Unknown
Negligible	Negligible
Low	Low
Medium	Medium
High	High
Critical	Critical

OSV のマッピング

表1.1 CVSSv3

ベーススコア	Clair の重大度
0.0	Negligible
0.1-3.9	Low
4.0-6.9	Medium
7.0-8.9	High
9.0-10.0	Critical

表1.2 CVSSv2

ベーススコア	Clair の重大度
0.0-3.9	Low
4.0-6.9	Medium

ベーススコア	Clair の重大度
7.0-10	High

第2章 CLAIR の概念

次のセクションでは、Clair がどのように機能するかの概念的な概要を示します。

2.1. CLAIR の実践

Clair の分析は、インデックス作成、マッチング、通知の 3 つの部分に分類されます。

2.1.1. インデックス作成

Clair のインデクサーサービスは、コンテナーイメージの構成を理解する上で重要です。Clair では、コンテナーイメージの表現を "マニフェスト" と呼びます。 マニフェストは、イメージのレイヤーのコンテンツを理解するために使用されます。このプロセスを効率化するために、Clair は Open Container Initiative (OCI) のマニフェストとレイヤーがコンテンツのアドレス指定を考慮して設計されていることを利用して、反復的なタスクを削減します。

インデックス作成時には、コンテナーイメージを表現するマニフェストが取得され、基本的な要素に分割されます。インデクサーの役割は、イメージに含まれるパッケージ、イメージの元のディストリビューション、およびイメージが依存するパッケージリポジトリーを明らかにすることです。この貴重な情報は、Clair のデータベース内に記録、保存されます。インデックス作成時に収集された詳細情報は、包括的な脆弱性レポートを生成する際の土台となります。このレポートは、さらなる分析とアクションのために matcher ノードにシームレスに転送でき、ユーザーがコンテナーイメージのセキュリティーについて情報に基づいた意思決定を行うのに役立ちます。

IndexReport は Clair のデータベースに保存されます。これを **matcher** ノードにフィードすることで、脆弱性レポートを計算できます。

2.1.2. マッチング

Clair では、マッチャーノードが、提供されたインデックスレポートと脆弱性をマッチングします。

matcher は脆弱性のデータベースを最新の状態に維持します。matcher は一連のアップデーターを実行し、定期的にデータソースをプローブして新しいコンテンツを探します。新しい脆弱性は、発見されると、データベースに保存されます。

matcher API は、クエリー時に常に最新の脆弱性レポートを提供するように設計されています。脆弱性レポートは、マニフェストのコンテンツと、そのコンテンツに影響を与える脆弱性の両方をまとめたものです。

新しい脆弱性は、発見されると、データベースに保存されます。

マッチャー API は頻繁な使用を想定して設計されています。マッチャー API は、クエリー時に常に最新の **VulnerabilityReport** を提供するように設計されています。**VulnerabilityReport** は、マニフェストのコンテンツと、コンテンツに影響を与える脆弱性の両方をまとめたものです。

2.1.3. ノーティファイアーサービス

Clair は、新しいセキュリティーデータベースの更新を追跡し、新しい脆弱性または削除された脆弱性がインデックス付きマニフェストに影響を与えるかどうかをユーザーに通知するノーティファイアーサービスを使用します。

ノーティファイアーは、以前にインデックス付けされたマニフェストに影響を与える新しい脆弱性を認識すると、**config.yaml** ファイルで設定されたメソッドを使用して、新しい変更に関する通知を発行します。返された通知は、変更により発見された最も深刻な脆弱性を表しています。これにより、同じセキュリティーデータベースの更新に対して過剰な通知が作成されるのを回避できます。

ユーザーが通知を受け取ると、ノーティファイアーは最新の脆弱性レポートを受信するために matcher に対して新しいリクエストを発行します。

次の方法で通知に登録できます。

- Webhook 配信
- AMQP 配信
- STOMP 配信

ノーティファイアーの設定は、Clair YAML 設定ファイルを介して行われます。

2.2. CLAIR 認証

現在のイテレーションでは、Clair v4 (Clair) は認証を内部で処理します。

注記

Clair の以前のバージョンでは、JWT Proxy を使用して認証を制御していました。

認証は、設定の **auth** キーの下に設定オブジェクトを指定することによって設定されます。複数の認証設定が存在する場合がありますが、次の順序で優先的に使用されます。

1. PSK。この認証設定により、Clair は事前共有キーを使用して JWT ベースの認証を実装します。
2. 設定以下に例を示します。

```
auth:
  psk:
    key: >-
      MDQ4ODBlNDAtNDc0ZC00MWUxLThhMzAtOTk0MzEwMGQwYTMxCg==
    iss: 'issuer'
```

この設定では、**auth** フィールドに 2 つのパラメーターが必要です。**iss** は、すべての受信リクエストを検証する発行者であり、**key** は、リクエストを検証するための base64 コード化された対称鍵です。

2.3. CLAIR アップデーター

Clair は、さまざまな脆弱性データベースを取得して解析するロジックを含む、**アップデーター** と呼ばれる **Go** パッケージを使用しています。

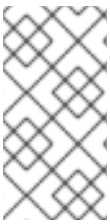
通常、アップデーターはマッチャーとペアになって、脆弱性がパッケージに関連しているかどうか、およびどのように関連しているかを解釈します。管理者は、脆弱性データベースの更新頻度を減らす場合も、使用されないことがわかっているデータベースから脆弱性をインポートしないようにする場合もあります。

2.4. CLAIR アップデーターに関する情報

次の表に、各 Clair アップデーターの詳細 (設定パラメーター、簡単な説明、関連 URL、対話する関連コンポーネントなど) を示します。このリストはすべてを網羅したものではありません。サーバーによってはリダイレクトを発行する場合があります。一部のリクエスト URL は、正確な脆弱性データを

確実に取得するために動的に構築されます。

Clair では、各アップデーターが、特定のパッケージタイプまたはディストリビューションに関連する脆弱性データの取得と解析を担当します。たとえば、Debian アップデーターは Debian ベースの Linux ディストリビューションを扱い、AWS アップデーターは Amazon Web Services の Linux ディストリビューションに固有の脆弱性を扱います。パッケージタイプを理解することは、脆弱性を管理する上で重要です。各パッケージタイプに固有のセキュリティー上の懸念があると、特定の更新やパッチが必要になる場合があるためです。



注記

Clair のアップデーター URL を使用して環境内でプロキシーサーバーを使用している場合は、Clair がスムーズにその URL にアクセスできるように、どの URL をプロキシー許可リストに追加する必要があるかを特定する必要があります。次の表を使用して、アップデーター URL をプロキシー許可リストに追加してください。

表2.1 Clair のアップデーターの情報

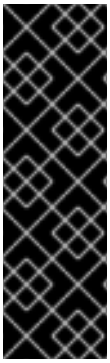
アップデーター	説明	URL	コンポーネント
alpine	Alpine アップデーターは、Alpine Linux ディストリビューションのパッケージに関連する脆弱性データの取得と解析を担当します。	• **https://secdb.alpinelinux.org/**	Alpine Linux SecDB データベース
aws	AWS アップデーターは AWS Linux ベースのパッケージを扱い、Amazon Web Services のカスタム Linux ディストリビューションに固有の脆弱性情報を最新の状態に保ちます。	• **http://repo.us-west-2.amazonaws.com/2018.03/updates/x86_64/mirror.list** • **https://cdn.amazonlinux.com/2/core/latest/x86_64/mirror.list** • **https://cdn.amazonlinux.com/al2023/core/mirrors/latest/x86_64/mirror.list**	Amazon Web Services (AWS) UpdateInfo
debian	Debian アップデーターは、Debian ベースの Linux ディストリビューションに関連するパッケージの脆弱性を追跡するために不可欠です。	• **https://deb.debian.org/** • **https://security-tracker.debian.org/tracker/data/json**	Debian Security Tracker

アップデーター	説明	URL	コンポーネント
clair.cvss	Clair Common Vulnerability Scoring System (CVSS) アップデーターは、脆弱性とそれに関連する CVSS スコアに関するデータの維持に特化しています。このスコアは特定のパッケージタイプではなく、一般的な脆弱性の重大度およびリスク評価に関連付けられます。	• **https://nvd.nist.gov/feeds/json/cve/1.1/**	JSON 形式の Common Vulnerabilities and Exposures (CVE) データの National Vulnerability Database (NVD) フィード
oracle	Oracle アップデーターは Oracle Linux パッケージに特化しており、Oracle Linux システムに影響を与える脆弱性に関するデータを維持します。	• **https://linux.oracle.com/security/oval/com.oracle.elsa-*.xml.bz2**	Oracle Oval データベース
photon	Photon アップデーターは、VMware Photon OS のパッケージを扱います。	• **https://packages.vmware.com/photon/photon_oval_definitions/**	VMware Photon OS oval の定義
rhel	Red Hat Enterprise Linux (RHEL) アップデーターは、Red Hat Enterprise Linux ディストリビューションのパッケージの脆弱性データを維持する役割を担っています。	• **https://access.redhat.com/security/cve/** • **https://access.redhat.com/security/data/oval/v2/PULP_MANIFEST**	Red Hat Enterprise Linux (RHEL) Oval データベース
rhcc	Red Hat Container Catalog (RHCC) アップデーターは、Red Hat のコンテナーイメージに関係しています。このアップデーターは、Red Hat のコンテナー化されたソフトウェアに関連する脆弱性情報を最新の状態に保ちます。	• **https://access.redhat.com/security/data/metrics/cvemap.xml**	Resource Handler Configuration Controller (RHCC) データベース
suse	SUSE アップデーターは、openSUSE、SUSE Enterprise Linux などを含む SUSE Linux ディストリビューションファミリーのパッケージの脆弱性情報を管理します。	• **https://support.novell.com/security/oval/**	SUSE Oval データベース

アップデーター	説明	URL	コンポーネント
ubuntu	Ubuntu アップデーターは、Ubuntu ベースの Linux ディストリビューションに関連するパッケージの脆弱性の追跡に特化しています。Ubuntu は、Linux エコシステムで広く使用されているディストリビューションです。	• **https://security-metadata.canonical.com/oval/com.ubuntu.*.cve.oval.xml** • **https://api.launchpad.net/1.0/**	Ubuntu Oval データベース
osv	Open Source Vulnerability (OSV) アップデーターは、オープンソースソフトウェアコンポーネント内の脆弱性の追跡に特化しています。OSV は、さまざまなオープンソースプロジェクトで見つかったセキュリティー問題に関する詳細情報を提供する重要なリソースです。	• **https://osv-vulnerabilities.storage.googleapis.com/**	Open Source Vulnerabilities データベース

2.5. アップデーターの設定

アップデーターは **clair-config.yaml** ファイルの **updaters.sets** キーによって設定できます。



重要

- **sets** フィールドが入力されていない場合、デフォルトですべてのセットが使用されます。すべてのセットを使用する場合、Clair は各アップデーターの URL にアクセスしようとします。プロキシー環境を使用している場合は、これらの URL をプロキシー許可リストに追加する必要があります。
- アップデーターがマッチャープロセス内で自動的に実行されている場合 (デフォルトの設定)、アップデーターを実行する期間はマッチャーの設定フィールドで設定されます。

2.5.1. 特定のアップデーターセットの選択

以下を参考に、Red Hat Quay デプロイメント用の 1 つまたは複数のアップデーターを選択してください。

複数のアップデーター用に Clair を設定する

複数の特定のアップデーター

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

```
    - aws
    - osv
#...
```

Alpine 用の Clair の設定

Alpine の config.yaml の例

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

AWS 用の Clair の設定

AWS の config.yaml の例

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

Debian 用の Clair の設定

Debian の config.yaml の例

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

Clair CVSS 用の Clair の設定

Clair CVSS の config.yaml の例

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

Oracle 用の Clair の設定

Oracle の config.yaml の例

```
#...
updaters:
  sets:
    - oracle
#...
```

Photon 用の Clair の設定

Photon の config.yaml の例

```
#...
updaters:
  sets:
    - photon
#...
```

SUSE 用の Clair の設定

SUSE の config.yaml の例

```
#...
updaters:
  sets:
    - suse
#...
```

Ubuntu 用の Clair の設定

Ubuntu の config.yaml の例

```
#...
updaters:
  sets:
    - ubuntu
#...
```

OSV 用の Clair の設定

OSV の config.yaml の例

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

2.5.2. Red Hat Enterprise Linux (RHEL) のすべての脆弱性を対象とするようにアップデーターセットを選択する

Red Hat Enterprise Linux (RHEL) のすべての脆弱性を対象とするには、次のアップデーターセットを使用する必要があります。

- **rhel**。このアップデーターは、RHEL に影響を与える脆弱性に関する最新情報を提供します。
- **rhcc**。このアップデーターは、Red Hat のコンテナーイメージに関連する脆弱性を追跡します。
- **clair.cvss**。このアップデーターは、Common Vulnerabilities and Exposures (CVE) スコアを提供し、脆弱性の重大度とリスク評価の全体像を提供します。

- **osv**。このアップデーターは、オープンソースソフトウェアコンポーネントの脆弱性の追跡に特化しています。RHEL 製品では Java と Go がよく使用されているため、このアップデーターが推奨されます。

RHEL アップデーターの例

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

2.5.3. 高度なアップデーター設定

場合によっては、特定の動作に合わせてアップデーターを設定する必要があります。たとえば、Open Source Vulnerabilities (OSV) アップデーター用に特定のエコシステムを許可リストに登録する場合などです。

高度なアップデータ設定は、プロキシーデプロイメントまたはエアギャップデプロイメントで役立つ場合があります。このような環境用の特定のアップデーター設定は、**updaters** オブジェクトの **config** 環境変数の下にキーを配置することで渡すことができます。Clair のログを調べて名前を再確認してください。

次の YAML スニペットで、一部の Clair アップデーターで利用できるさまざまな設定を詳しく説明します。

重要

ユーザーの増加に合わせて高度なアップデーター設定を行う必要はありません。

alpine アップデーターの設定

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

debian アップデーターの設定

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

```
      mirror_url: https://deb.debian.org/
      json_url: https://security-tracker.debian.org/tracker/data/json
#...
```

clair.cvss アップデーターの設定

```
#...
updaters:
  config:
    clair.cvss:
      url: https://nvd.nist.gov/feeds/json/cve/1.1/
#...
```

oracle アップデーターの設定

```
#...
updaters:
  sets:
    - oracle
  config:
    oracle-2023-updater:
      url:
        - https://linux.oracle.com/security/oval/com.oracle.elsa-2023.xml.bz2
    oracle-2022-updater:
      url:
        - https://linux.oracle.com/security/oval/com.oracle.elsa-2022.xml.bz2
#...
```

photon アップデーターの設定

```
#...
updaters:
  sets:
    - photon
  config:
    photon:
      url: https://packages.vmware.com/photon/photon_oval_definitions/
#...
```

rhel アップデーターの設定

```
#...
updaters:
  sets:
    - rhel
  config:
    rhel:
      url: https://access.redhat.com/security/data/oval/v2/PULP_MANIFEST
      ignore_unpatched: true (1)
#...
```

1. ブール値。対応するパッチや更新が利用できない脆弱性に関する情報を含めるかどうか。

rhcc アップデーターの設定

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

suse アップデーターの設定

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

ubuntu アップデーターの設定

```
#...
updaters:
  config:
    ubuntu:
      url: https://api.launchpad.net/1.0/
      name: ubuntu
      force: 1
        - name: focal 2
          version: 20.04 3
#...
```

1. API レスポンスのステータスに関係なく、生成される UpdaterSet に特定のディストリビューションとバージョンの詳細を強制的に含めるために使用します。特定のディストリビューションとバージョンがアップデーター設定に一貫して含まれていることを確認する場合に便利です。
2. UpdaterSet に強制的に含めるディストリビューション名を指定します。
3. UpdaterSet に強制的に含めるディストリビューションのバージョンを指定します。

OSV アップデーターの設定

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

1 許可するエコシステムのリスト。未設定のままにすると、すべてのエコシステムが許可されます。小文字を使用する必要があります。サポートされているエコシステムのリストについては、定義されているエコシステム のドキュメントを参照してください。

2.5.4. Clair アップデーターコンポーネントの無効化

一部のシナリオでは、Clair アップデーターコンポーネントを無効にする場合があります。切断された環境で Red Hat Quay を実行する場合は、アップデーターを無効にする必要があります。

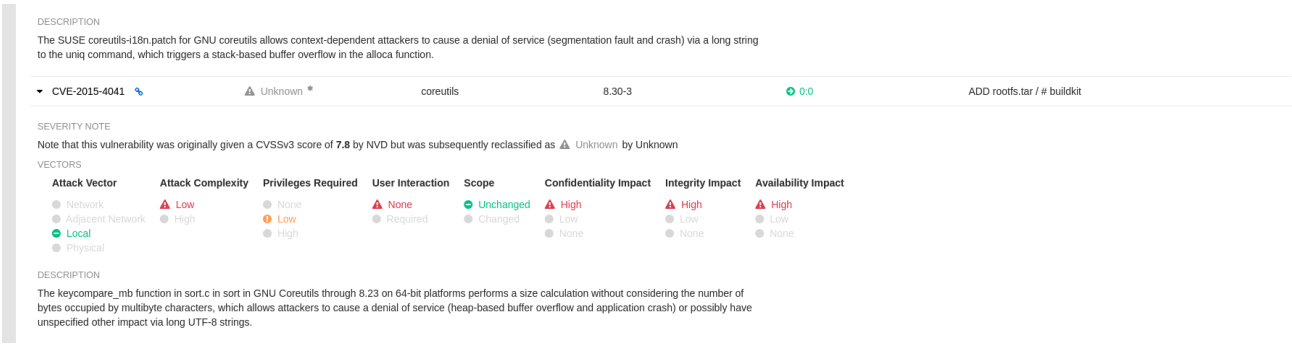
次の例では、Clair アップデーターが無効になっています。

```
#...
matcher:
  disable_updaters: true
#...
```

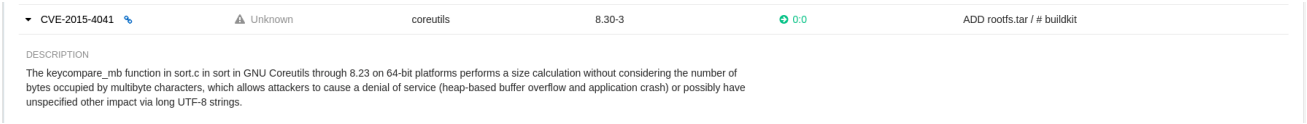
2.6. 脆弱性情報データベース (NVD: NATIONAL VULNERABILITY DATABASE) の CVE 評価

Clair v4.2 の時点で、Common Vulnerability Scoring System (CVSS) 強化データが Red Hat Quay UI で表示できるようになりました。さらに、Clair v4.2 は、検出された脆弱性について National Vulnerability Database から CVSS スコアを追加します。

今回の変更により、脆弱性の CVSS スコアがディストリビューションスコアの 2 レベル以内である場合、Red Hat Quay UI はデフォルトでディストリビューションのスコアを提示します。以下に例を示します。



これは以前のインターフェイスとは異なり、以下の情報のみを表示します。



2.7. 連邦情報処理標準 (FIPS) の準備と準拠

米国国立標準技術研究所 (NIST) によって開発された連邦情報処理標準 (FIPS) は、特に銀行、医療、公共部門などの高度に規制された分野で、機密データを保護および暗号化するために高く評価されていると見なされています。Red Hat Enterprise Linux (RHEL) および OpenShift Container Platform は **FIPS モード** を提供することで FIPS をサポートします。このモードでは、システムは **openssl** などの特定の FIPS 検証済み暗号モジュールの使用のみを許可します。これにより、FIPS への準拠が保証されます。

2.7.1. FIPS コンプライアンスの有効化

以下の手順を使用して、Red Hat Quay デプロイメントで FIPS コンプライアンスを有効にします。

前提条件

- Red Hat Quay のスタンドアロンデプロイメントを実行している場合、Red Hat Enterprise Linux (RHEL) デプロイメントがバージョン 8 以降であり、FIPS が有効である。
- Red Hat Quay を OpenShift Container Platform にデプロイしている場合、OpenShift Container Platform がバージョン 4.10 以降である。
- Red Hat Quay のバージョンが 3.5.0 以降である。
- IBM Power または IBM Z クラスター上の OpenShift Container Platform で Red Hat Quay を使用している場合:
  - OpenShift Container Platform バージョン 4.14 以降
  - Red Hat Quay バージョン 3.10 以降
- Red Hat Quay デプロイメントの管理者権限がある。

手順

- Red Hat Quay の **config.yaml** ファイルで、**FEATURE_FIPS** 設定フィールドを **true** に設定します。以下に例を示します。

```
---
FEATURE_FIPS = true
---
```

FEATURE_FIPS を **true** に設定すると、Red Hat Quay は FIPS 準拠のハッシュ関数を使用して実行されます。

パート II. RED HAT QUAY の CLAIR

このガイドには、スタンドアロンおよび OpenShift Container Platform Operator デプロイメントの両方で Red Hat Quay で Clair を実行するための手順が含まれています。

第3章 スタンドアロンの RED HAT QUAY デプロイメントでの CLAIR のセットアップ

スタンドアロンの Red Hat Quay デプロイメントの場合、Clair を手動でセットアップできます。

手順

1. Red Hat Quay インストールディレクトリーに、Clair データベースデータ用の新しいディレクトリーを作成します。

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

2. 次のコマンドを入力して、**postgres-clairv4** ファイルに適切な権限を設定します。

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

3. 次のコマンドを入力して、Clair Postgres データベースをデプロイします。

```
$ 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 \
  registry.redhat.io/rhel8/postgresql-13:1-109
```

4. Clair デプロイメント用に Postgres **uuid-ossp** モジュールをインストールします。

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

出力例

```
CREATE EXTENSION
```

注記

Clair では、**uuid-ossp** 拡張機能を Postgres データベースに追加する必要があります。適切な権限を持つユーザーの場合は、拡張機能を作成すると、Clair によって自動的に追加されます。ユーザーが適切な権限を持っていない場合は、Clair を開始する前に拡張機能を追加する必要があります。

拡張機能が存在しない場合は、Clair が起動しようとすると、**ERROR: Please load the "uuid-ossp" extension.(SQLSTATE 42501)** エラーが発生します。

5. 実行中の場合は、**Quay** コンテナーを停止し、設定モードで再始動して、既存の設定をボリュームとしてロードします。

```
$ sudo podman run --rm -it --name quay_config \
  -p 80:8080 -p 443:8443 \
  -v $QUAY/config:/conf/stack:Z \
```

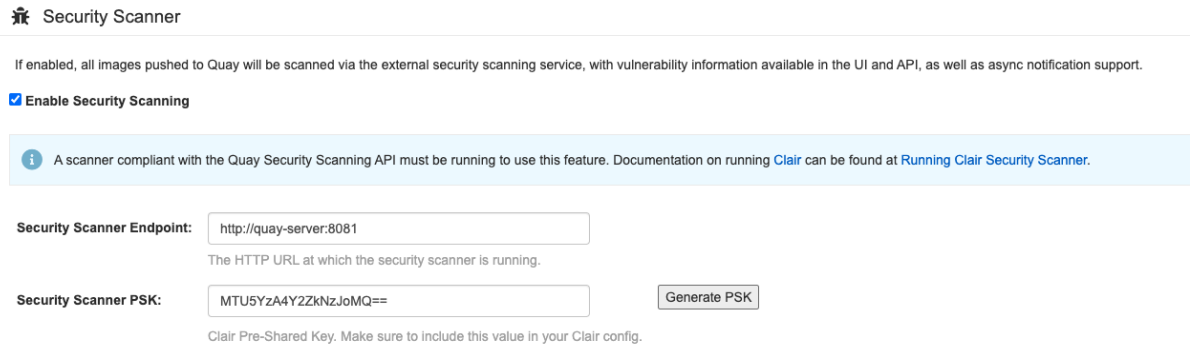
```
{productrepo}/{quayimage}:{productminv} config secret
```

6. 設定ツールにログインし、UI の **Security Scanner** セクションで **Enable Security Scanning** をクリックします。

7. **quay-server** システムでまだ使用されていないポート (**8081** など) を使用して、Clair の HTTP エンドポイントを設定します。

8. **Generate PSK** ボタンを使用して、事前共有キー (PSK) を作成します。

セキュリティースキャナー UI



9. Red Hat Quay の **config.yaml** ファイルを検証してダウンロードし、設定エディターを実行している **Quay** コンテナーを停止します。

10. 新しい設定バンドルを Red Hat Quay インストールディレクトリーに展開します。次に例を示します。

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

11. Clair 設定ファイル用のフォルダーを作成します。次に例を示します。

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

12. Clair 設定フォルダーに移動します。

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

13. 以下のように、Clair 設定ファイルを作成します。

```
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"
```

Clair の設定形式の詳細は、Clair 設定リファレンス を参照してください。

14. コンテナーイメージを使用して Clair を起動し、作成したファイルから設定にマウントします。

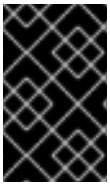
```
$ sudo podman run -d --name clairv4 \
-p 8081:8081 -p 8088:8088 \
-e CLAIR_CONF=/clair/config.yaml \
-e CLAIR_MODE=combo \
-v /etc/opt/clairv4/config:/clair:Z \
registry.redhat.io/quay/clair-rhel8:v3.10.3
```

注記

複数の Clair コンテナーを実行することもできますが、単一のコンテナーを超えるデプロイシナリオでは、Kubernetes や OpenShift Container Platform などのコンテナーオーケストレーターを使用することが強く推奨されます。

3.1. RED HAT QUAY のアップストリームイメージで CLAIR を使用する

ほとんどのユーザーにとって、Clair を現在のバージョン (4.7.2) から個別にアップグレードする必要はありません。ただし、特定のバグ修正やダウンストリームでまだリリースされていない新機能を試したいなど、さまざまな理由で、顧客が アップストリームリポジトリー から Clair のイメージをプルしたい場合があります。以下の手順を使用して、Clair のアップストリームバージョンを Red Hat Quay で実行できます。



重要

Clair のアップストリームバージョンは、Red Hat Quay との互換性について完全にテストされていません。結果として、この組み合わせにより、展開に問題が発生する可能性があります。

手順

1. Clair が実行中の場合は、次のコマンドを入力して停止します。

```
$ podman stop <clairv4_container_name>
```

2. 上流のリポジトリー に移動し、使用する Clair のバージョンを見つけて、それをローカルマシンにプルします。以下に例を示します。

```
$ podman pull quay.io/projectquay/clair:nightly-2024-02-03
```

3. コンテナーイメージを使用して Clair を起動し、作成したファイルから設定にマウントします。

```
$ 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 \
quay.io/projectquay/clair:nightly-2024-02-03
```

第4章 OPENSHIFT CONTAINER PLATFORM の CLAIR

OpenShift Container Platform 上の Red Hat Quay デプロイメントで Clair v4 (Clair) をセットアップするには、Red Hat Quay Operator を使用することが推奨されます。デフォルトでは、Red Hat Quay Operator は、Clair デプロイメントを Red Hat Quay デプロイメントとともにインストールまたはアップグレードし、Clair を自動的に設定します。

第5章 CLAIR のテスト

以下の手順を使用して、スタンドアロンの Red Hat Quay デプロイメントまたは OpenShift Container Platform Operator ベースのデプロイメントで Clair をテストします。

前提条件

- Clair コンテナーイメージをデプロイしている。

手順

1. 次のコマンドを入力して、サンプルイメージをプルします。

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

2. 次のコマンドを入力して、レジストリーにイメージをタグ付けします。

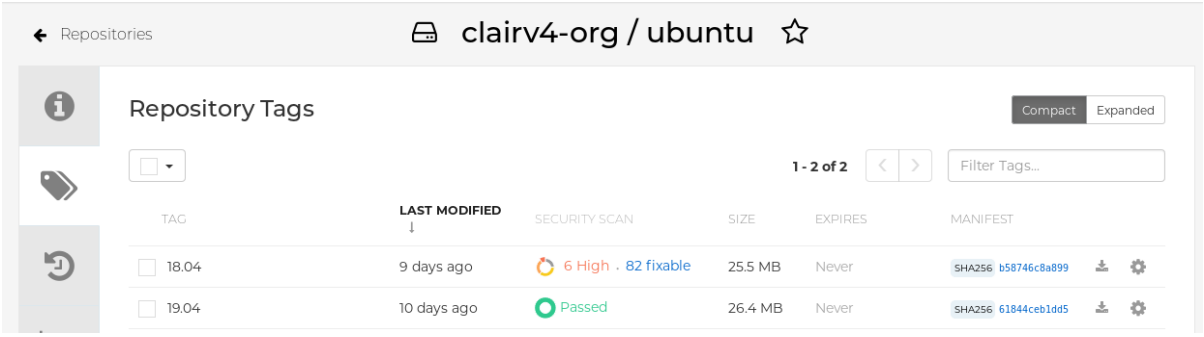
```
$ sudo podman tag docker.io/library/ubuntu:20.04 <quay-server.example.com>/<user-name>/ubuntu:20.04
```

3. 以下のコマンドを入力して、イメージを Red Hat Quay レジストリーにプッシュします。

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

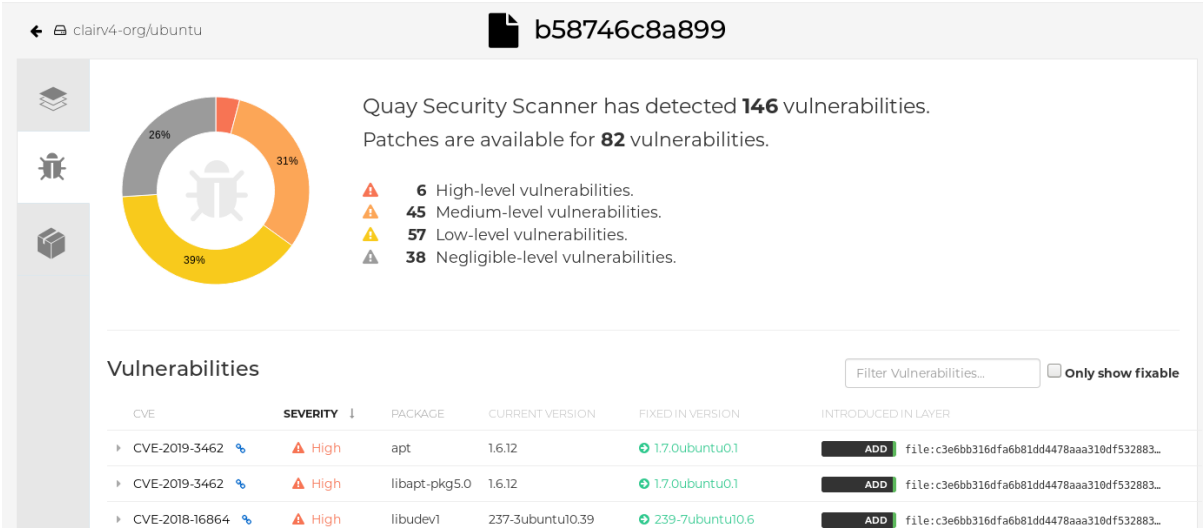
4. UI から Red Hat Quay デプロイメントにログインします。
5. リポジトリー名 (**quayadmin/ubuntu** など) をクリックします。
6. ナビゲーションウィンドウで、**Tags** をクリックします。

レポートの概要



7. イメージレポート (例: **45 medium**) をクリックして、より詳細なレポートを表示します。

レポートの詳細



注記

場合によっては、Clair はイメージに関する重複レポートを表示します (例: **ubi8/nodejs-12** または **ubi8/nodejs-16**)。これは、同じ名前の脆弱性が異なるパッケージに存在するために発生します。この動作は Clair 脆弱性レポートで予期されており、バグとしては扱われません。

パート III. 高度な CLAIR 設定

このセクションを使用して、高度な Clair 機能を設定します。

第6章 アンマネージド CLAIR 設定

Red Hat Quay ユーザーは、Red Hat Quay OpenShift Container Platform Operator を使用してアンマネージド Clair 設定を実行できます。この機能により、ユーザーはアンマネージド Clair データベースを作成したり、アンマネージドデータベースなしでカスタム Clair 設定を実行したりできます。

アンマネージド Clair データベースにより、Red Hat Quay オペレーターは、Operator の複数のインスタンスが同じデータベースと通信する必要がある地理的に複製された環境で作業できます。アンマネージド Clair データベースは、ユーザーがクラスターの外部に存在する高可用性 (HA) Clair データベースを必要とする場合にも使用できます。

6.1. アンマネージド CLAIR データベースを使用したカスタム CLAIR 設定の実行

次の手順を使用して、Clair データベースをアンマネージドに設定します。

手順

- Quay Operator で、**QuayRegistry** カスタムリソースの **clairpostgres** コンポーネントを **managed: false** に設定します。

```
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. アンマネージド CLAIR データベースを使用したカスタム CLAIR データベースの設定

OpenShift Container Platform 上の Red Hat Quay では、ユーザーが独自の Clair データベースを指定できます。

次の手順を使用して、カスタム Clair データベースを作成します。

注記

次の手順では、SSL/TLS 証明書を使用して Clair をセットアップします。SSL/TSL 証明書を使用して Clair をセットアップしない同様の手順を表示するには、マネージド Clair 設定を使用したカスタム Clair データベースの設定を参照してください。

手順

1. 次のコマンドを入力して、**clair-config.yaml** を含む Quay 設定バンドルシークレットを作成します。

```
$ 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
```

Clair config.yaml ファイルの例

```
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
```

注記

- データベース証明書は、**clair-config.yaml** の Clair アプリケーション Pod の **/run/certs/rds-ca-2019-root.pem** の下にマウントされます。**clair-config.yaml** を設定するときに指定する必要があります。
- **clair-config.yaml** の例は、OpenShift 設定の Clair にあります。

2. **clair-config.yaml** ファイルをバンドルシークレットに追加します。次に例を示します。

```
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>
```

注記

更新すると、提供された **clair-config.yaml** ファイルが Clair Pod にマウントされます。提供されていないフィールドには、Clair 設定モジュールを使用してデフォルトが自動的に入力されます。

3. **Build History** ページでコミットをクリックするか、**oc get pods -n <namespace>** を実行して、Clair Pod のステータスを確認できます。以下に例を示します。

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

出力例

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

第7章 マネージド CLAIR データベースを使用したカスタム CLAIR 設定の実行

場合によっては、マネージド Clair データベースを使用してカスタム Clair 設定を実行します。これは、以下のシナリオで役に立ちます。

- ユーザーが特定のアップデーターリソースを無効にする場合。
- ユーザーが非接続環境で Red Hat Quay を実行している場合。非接続環境での Clair の実行の詳細は、非接続環境での Clair を参照してください。

注記

- 非接続環境で Red Hat Quay を実行している場合は、**clair-config.yaml** の **airgap** パラメーターを **true** に設定する必要があります。
- 非接続環境で Red Hat Quay を実行している場合は、すべてのアップデーターコンポーネントを無効にする必要があります。

7.1. CLAIR データベースをマネージドに設定する

次の手順を使用して、Clair データベースをマネージドに設定します。

手順

- Quay Operator で、**QuayRegistry** カスタムリソースの **clairpostgres** コンポーネントを **managed: true** に設定します。

```
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. マネージド CLAIR 設定を使用したカスタム CLAIR データベースの設定

OpenShift Container Platform 上の Red Hat Quay では、ユーザーが独自の Clair データベースを指定できます。

次の手順を使用して、カスタム Clair データベースを作成します。

手順

1. 次のコマンドを入力して、**clair-config.yaml** を含む Quay 設定バンドルシークレットを作成します。

```
$ 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
```

Clair config.yaml ファイルの例

```
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
```

注記

- データベース証明書は、**clair-config.yaml** の Clair アプリケーション Pod の **/run/certs/rds-ca-2019-root.pem** の下にマウントされます。**clair-config.yaml** を設定するときに指定する必要があります。
- **clair-config.yaml** の例は、OpenShift 設定の Clair にあります。

2. **clair-config.yaml** ファイルをバンドルシークレットに追加します。次に例を示します。

```
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>
```

注記

- 更新すると、提供された **clair-config.yaml** ファイルが Clair Pod にマウントされます。提供されていないフィールドには、Clair 設定モジュールを使用してデフォルトが自動的に入力されます。

3. **Build History** ページでコミットをクリックするか、**oc get pods -n <namespace>** を実行して、Clair Pod のステータスを確認できます。以下に例を示します。

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

出力例

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

第8章 非接続環境での CLAIR

注記

現在、非接続環境での Clair のデプロイは、IBM Power および IBM Z ではサポートされていません。

Clair は、**updater** と呼ばれる一連のコンポーネントを使用して、さまざまな脆弱性データベースからのデータのフェッチと解析を処理します。updater はデフォルトで、脆弱性データをインターネットから直接プルし、すぐに使用できるように設定されています。ただし、ユーザーによっては、Red Hat Quay を非接続環境、またはインターネットに直接アクセスできない環境で実行する必要がある場合があります。Clair は、ネットワーク分離を考慮したさまざまな種類の更新ワークフローを使用することで、非接続環境をサポートします。これは **clairctl** コマンドラインインターフェイスツールを使用して機能します。このツールは、オープンホストを使用してインターネットから updater データを取得し、そのデータを隔離されたホストにセキュアに転送してから、隔離されたホスト上の updater データを Clair にインポートします。

非接続環境で Clair をデプロイするには、このガイドを使用してください。

重要

既知の問題 PROJQUAY-6577 により、Red Hat Quay Operator はカスタマイズされた Clair **config.yaml** ファイルを適切に処理しません。そのため、現在、次の手順は機能しません。

Operator を利用してフィールドに入力するのではなく、ユーザー自身が Clair の設定全体を最初から作成する必要があります。これを行うには、Procedure to enable Clair scanning of images in disconnected environments の手順に従ってください。

注記

現在、Clair エンリッチメントデータは CVSS データです。エンリッチメントデータは現在、オフライン環境ではサポートされていません。

Clair アップデーターの詳細は、Clair アップデーターを参照してください。

8.1. 非接続の OPENSHIFT CONTAINER PLATFORM クラスターで CLAIR をセットアップする

以下の手順を使用して、非接続の OpenShift Container Platform クラスターに OpenShift Container Platform でプロビジョニングされた Clair Pod をセットアップします。

重要

既知の問題 PROJQUAY-6577 により、Red Hat Quay Operator はカスタマイズされた Clair **config.yaml** ファイルを適切に処理しません。そのため、現在、次の手順は機能しません。

Operator を利用してフィールドに入力するのではなく、ユーザー自身が Clair の設定全体を最初から作成する必要があります。これを行うには、Procedure to enable Clair scanning of images in disconnected environments の手順に従ってください。

8.1.1. OpenShift Container Platform デプロイメント用の clairctl コマンドラインユーティリティーツールのインストール

以下の手順を使用して、OpenShift Container Platform デプロイメント用の **clairctl** CLI ツールをインストールします。

手順

1. 以下のコマンドを入力して、Clair デプロイメント用の **clairctl** プログラムを OpenShift Container Platform クラスターにインストールします。

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

注記

非公式ですが、**clairctl** ツールをダウンロードできます。

2. **clairctl** ファイルの権限を設定して、ユーザーが実行できるようにします。次に例を示します。

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

8.1.2. OpenShift Container Platform での Clair デプロイメントの Clair 設定シークレットの取得とデコード

以下の手順を使用して、OpenShift Container Platform 上の OpenShift Container Platform でプロビジョニングされた Clair インスタンスの設定シークレットを取得してデコードします。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。

手順

1. 次のコマンドを入力して、設定シークレットを取得してデコードし、それを Clair 設定 YAML に保存します。

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

2. **disable_updaters** および **airgap** パラメーターが **true** に設定されるように、**clair-config.yaml** ファイルを更新します。次に例を示します。

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

8.1.3. 接続された Clair インスタンスからアップデータバンドルをエクスポートする

次の手順を使用して、インターネットにアクセスできる Clair インスタンスから更新プログラムバンドルをエクスポートします。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。
- Clair 設定シークレットを取得してデコードし、Clair の **config.yaml** ファイルに保存している。
- Clair の **config.yaml** ファイルで、**disable_updaters** および **airgap** パラメーターが **true** に設定されている。

手順

- インターネットにアクセスできる Clair インスタンスから、設定ファイルで **clairctl** CLI ツールを使用して、アップデーターバンドルをエクスポートします。以下に例を示します。

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

8.1.4. 非接続の OpenShift Container Platform クラスター内の Clair データベースへのアクセスの設定

以下の手順を使用して、非接続の OpenShift Container Platform クラスター内の Clair データベースへのアクセスを設定します。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。
- Clair 設定シークレットを取得してデコードし、Clair の **config.yaml** ファイルに保存している。
- Clair の **config.yaml** ファイルで、**disable_updaters** および **airgap** パラメーターが **true** に設定されている。
- インターネットにアクセスできる Clair インスタンスからアップデーターバンドルをエクスポートしている。

手順

1. CLI ツール **oc** を使用して、Clair データベースサービスを特定します。次に例を示します。

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

出力例

```
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. Clair データベースポートを転送して、ローカルマシンからアクセスできるようにします。以下に例を示します。

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

3. Clair の **config.yaml** ファイルを更新します。次に例を示します。

```
indexer:
  connstring: host=localhost port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable (1)
  scanlock_retry: 10
  layer_scan_concurrency: 5
  migrations: true
  scanner:
    repo:
      rhel-repository-scanner: (2)
        repo2cpe_mapping_file: /data/cpe-map.json
    package:
      rhel_containerscanner: (3)
        name2repos_mapping_file: /data/repo-map.json
```

(1) 複数の **connstring** フィールドの **host** の値を **localhost** に置き換えます。

(2) **rhel-repository-scanner** パラメーターの詳細は、「Common Product Enumeration 情報へのリポジトリーのマッピング」を参照してください。

(3) **rhel_containersscanner** パラメーターの詳細は、「Common Product Enumeration へのリポジトリーのマッピング」を参照してください。

8.1.5. 非接続の OpenShift Container Platform クラスターへのアップデーターバンドルのインポート

以下の手順を使用して、アップデーターバンドルを非接続の OpenShift Container Platform クラスターにインポートします。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。
- Clair 設定シークレットを取得してデコードし、Clair の **config.yaml** ファイルに保存している。
- Clair の **config.yaml** ファイルで、**disable_updaters** および **airgap** パラメーターが **true** に設定されている。
- インターネットにアクセスできる Clair インスタンスからアップデーターバンドルをエクスポートしている。
- アップデーターバンドルを非接続環境に転送している。

手順

- CLI ツール **clairctl** を使用して、アップデーターバンドルを OpenShift Container Platform によってデプロイされた Clair データベースにインポートします。以下に例を示します。

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

8.2. 非接続の OPENSHIFT CONTAINER PLATFORM クラスター用の CLAIR の自己管理デプロイメントをセットアップする

以下の手順を使用して、非接続の OpenShift Container Platform クラスター用の Clair の自己管理デプロイメントをセットアップします。

重要

既知の問題 PROJQUAY-6577 により、Red Hat Quay Operator はカスタマイズされた Clair **config.yaml** ファイルを適切に処理しません。そのため、現在、次の手順は機能しません。

Operator を利用してフィールドに入力するのではなく、ユーザー自身が Clair の設定全体を最初から作成する必要があります。これを行うには、Procedure to enable Clair scanning of images in disconnected environments の手順に従ってください。

8.2.1. OpenShift Container Platform で自己管理 Clair デプロイメント用の clairctl コマンドラインユーティリティーツールをインストールする

以下の手順を使用して、OpenShift Container Platform に自己管理 Clair デプロイメント用の **clairctl** CLI ツールをインストールします。

手順

1. **podman cp** コマンドを使用して、自己管理の Clair デプロイメント用の **clairctl** プログラムをインストールします。以下に例を示します。

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

2. **clairctl** ファイルの権限を設定して、ユーザーが実行できるようにします。次に例を示します。

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

8.2.2. 非接続の OpenShift Container Platform クラスター用の自己管理 Clair コンテナーをデプロイする

以下の手順を使用して、非接続の OpenShift Container Platform クラスター用の自己管理 Clair コンテナーをデプロイします。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。

手順

1. Clair 設定ファイル用のフォルダーを作成します。次に例を示します。

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

2. **disable_updaters** パラメーターを **true** に設定して Clair 設定ファイルを作成します。次に例を示します。

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

3. コンテナーイメージを使用して Clair を起動し、作成したファイルから設定にマウントします。

```
$ 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:v3.10.3
```

8.2.3. 接続された Clair インスタンスからアップデータバンドルをエクスポートする

次の手順を使用して、インターネットにアクセスできる Clair インスタンスから更新プログラムバンドルをエクスポートします。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。
- Clair をデプロイしている。
- Clair の **config.yaml** ファイルで、**disable_updaters** および **airgap** パラメーターが **true** に設定されている。

手順

- インターネットにアクセスできる Clair インスタンスから、設定ファイルで **clairctl** CLI ツールを使用して、アップデーターバンドルをエクスポートします。以下に例を示します。

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

8.2.4. 非接続の OpenShift Container Platform クラスター内の Clair データベースへのアクセスの設定

以下の手順を使用して、非接続の OpenShift Container Platform クラスター内の Clair データベースへのアクセスを設定します。

前提条件

- **clairctl** コマンドラインユーティリティーツールをインストールしている。
- Clair をデプロイしている。

- Clair の **config.yaml** ファイルで、**disable_updaters** および **airgap** パラメーターが **true** に設定されている。
- インターネットにアクセスできる Clair インスタンスからアップデーターバンドルをエクスポートしている。

手順

1. CLI ツール **oc** を使用して、Clair データベースサービスを特定します。次に例を示します。

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

出力例

```
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. Clair データベースポートを転送して、ローカルマシンからアクセスできるようにします。以下に例を示します。

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

3. Clair の **config.yaml** ファイルを更新します。次に例を示します。

```
indexer:
  connstring: host=localhost port=5432 dbname=postgres user=postgres
password=postgres sslmode=disable (1)
  scanlock_retry: 10
  layer_scan_concurrency: 5
  migrations: true
  scanner:
    repo:
      rhel-repository-scanner: (2)
        repo2cpe_mapping_file: /data/cpe-map.json
    package:
      rhel_containerscanner: (3)
        name2repos_mapping_file: /data/repo-map.json
```

(1) 複数の **connstring** フィールドの **host** の値を **localhost** に置き換えます。

(2) **rhel-repository-scanner** パラメーターの詳細は、「Common Product Enumeration 情報へのリポジトリーのマッピング」を参照してください。

(3) **rhel_containersscanner** パラメーターの詳細は、「Common Product Enumeration へのリポジトリーのマッピング」を参照してください。

8.2.5. 非接続の OpenShift Container Platform クラスターへのアップデーターバンドルのインポート

以下の手順を使用して、アップデーターバンドルを非接続の OpenShift Container Platform クラスターにインポートします。

前提条件

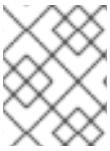
- **clairctl** コマンドラインユーティリティーツールをインストールしている。
- Clair をデプロイしている。
- Clair の **config.yaml** ファイルで、**disable_updaters** および **airgap** パラメーターが **true** に設定されている。
- インターネットにアクセスできる Clair インスタンスからアップデーターバンドルをエクスポートしている。
- アップデーターバンドルを非接続環境に転送している。

手順

- CLI ツール **clairctl** を使用して、アップデーターバンドルを OpenShift Container Platform によってデプロイされた Clair データベースにインポートします。

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

8.3. COMMON PRODUCT ENUMERATION へのリポジトリーのマッピング



注記

現在、Common Product Enumeration へのリポジトリーのマッピングは、IBM Power および IBM Z ではサポートされていません。

Clair の Red Hat Enterprise Linux (RHEL) スキャナーは、Common Product Enumeration (CPE) ファイルに依存して、RPM パッケージを対応するセキュリティーデータにマッピングし、マッチングする結果を生成します。これらのファイルは製品セキュリティーによって所有され、毎日更新されます。

スキャナーが RPM を適切に処理するには、CPE ファイルが存在するか、ファイルへのアクセスが許可されている必要があります。ファイルが存在しないと、コンテナーイメージにインストールされている RPM パッケージはスキャンされません。

表8.1 Clair CPE マッピングファイル

CPE	JSON マッピングファイルへのリンク
repos2cpe	Red Hat Repository-to-CPE JSON
names2repos	Red Hat Name-to-Repos JSON

非接続の Clair インストール用のデータベースに CVE 情報をアップロードするだけでなく、マッピングファイルをローカルで使用できるようにする必要もあります。

- スタンドアロン Red Hat Quay および Clair デプロイメントの場合は、マッピングファイルを Clair Pod に読み込む必要があります。
- OpenShift Container Platform デプロイメント上の Red Hat Quay の場合、Clair コンポーネントを **unmanaged** に設定する必要があります。次に、Clair を手動でデプロイメントし、マッピングファイルのローカルコピーを読み込むように設定する必要があります。

8.3.1. Common Product Enumeration サンプル設定へのリポジトリーのマッピング

Clair 設定の **repo2cpe_mapping_file** フィールドと **name2repos_mapping_file** フィールドを使用して、CPE JSON マッピングファイルを含めます。以下に例を示します。

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

詳細は、OVAL セキュリティーデータをインストール済みの RPM と正確にマッチングする方法 を参照してください。

第9章 CLAIR 設定の概要

Clair は、構造化された YAML ファイルによって設定されます。各 Clair ノードは、CLI フラグまたは環境変数を使用して、実行するモードと設定ファイルへのパスを指定する必要があります。以下に例を示します。

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

または、以下を実行します。

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

前述のコマンドはそれぞれ、同じ設定ファイルを使用して 2 つの Clair ノードを開始します。1 つはインデックス作成機能を実行し、もう 1 つはマッチング機能を実行します。

Clair を **combo** モードで実行している場合は、設定でインデクサー、matcher、およびノーティファイアー設定ブロックを指定する必要があります。

9.1. プロキシー環境での CLAIR の使用に関する情報

Go 標準ライブラリーが尊重する環境変数は、必要に応じて指定できます。次に例を示します。

- **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>
  ```

Clair のアップデーター URL を使用して環境内でプロキシーサーバーを使用している場合は、Clair がスムーズにその URL にアクセスできるように、どの URL をプロキシー許可リストに追加する必要があるかを特定する必要があります。たとえば、**osv** アップデーターは、エコシステムデータダンプを取得するために **https://osv-vulnerabilities.storage.googleapis.com** にアクセスする必要があります。このような場合、URL をプロキシー許可リストに追加する必要があります。アップデーター URL の完全なリストについては、「Clair のアップデーター URL」を参照してください。

また、標準の Clair URL がプロキシー許可リストに追加されていることを確認する必要があります。

- **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**

プロキシーサーバーを設定するときは、Clair とこれらの URL 間のシームレスな通信を可能にするために必要な認証要件や特定のプロキシー設定を考慮してください。これらの考慮事項をすべて文書化して対処することで、アップデーターのトラフィックをプロキシー経由でルーティングしながら、Clair を

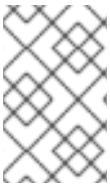
効果的に機能させることができます。

9.2. CLAIR 設定リファレンス

次の YAML は、Clair 設定の例を示しています。

```
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: ""
  prometheus:
    endpoint: null
  dogstatsd:
    url: ""
```



注記

上記の YAML ファイルには、万全を期すためにすべてのキーがリストされています。この設定ファイルをそのまま使用すると、一部のオプションがデフォルトで正常に設定されない場合があります。

9.3. CLAIR の一般的なフィールド

次の表では、Clair デプロイメントで使用できる一般的な設定フィールドについて説明します。

フィールド	Typhttp_listen_ae	説明
http_listen_addr	String	HTTP API が公開される場所を設定します。 デフォルト: **:6060**
introspection_addr	String	Clair のメトリクスと正常性エンドポイントが公開される場所を設定します。
log_level	String	ログレベルを設定します。文字列 **debug-color**、**debug**、**info**、**warn**、**error**、**fatal**、**panic** のいずれかが必要です。
tls	String	TLS/SSL および HTTP/2 の HTTP API を提供するための設定を含むマップ。
.cert	String	使用する TLS 証明書。フルチェーン証明書である必要があります。

一般的な Clair フィールドの設定例

次の例は、Clair 設定を示しています。

一般的な Clair フィールドの設定例

```
# ...
http_listen_addr: 0.0.0.0:6060
```

```
introspection_addr: 0.0.0.0:8089
log_level: info
# ...
```

9.4. CLAIR インデクサー設定フィールド

次の表では、Clair の **indexer** コンポーネントの設定フィールドについて説明します。

フィールド	型	説明
indexer	Object	Clair インデクサーノード設定を提供します。
.airgap	Boolean	インデクサーとフェッチャーのインターネットへの HTTP アクセスを無効にします。プライベート IPv4 および IPv6 アドレスが許可されます。データベース接続は影響を受けません。
.connstring	String	Postgres 接続文字列。URL または libpq 接続文字列として形式を受け入れます。
.index_report_request_concurrency	Integer	レートは、インデックスレポート作成リクエストの数を制限します。これを **0** に設定すると、この値の自動サイズ調整が試みられます。負の値を設定すると、無制限になります。自動サイジングは、使用可能なコア数の倍数です。 同時実行数を超えた場合、API はステータスコード **429** を返します。
.scanlock_retry	Integer	秒を表す正の整数。並行インデクサーは、マニフェストスキャンをロックして、上書きを回避します。この値は、待機中のインデクサーがロックをポーリングする頻度をチューニングします。
.layer_scan_concurrency	Integer	レイヤーの同時スキャン数を制限する正の整数。インデクサーは、マニフェストのレイヤーを同時にマッチングします。この値は、インデクサーが並行してスキャンするレイヤーの数をチューニングします。

フィールド	型	説明
.migrations	Boolean	インデクサーノードがデータベースへの移行を処理するかどうか。
.scanner	String	インデクサー設定。 Scanner を使用すると、設定オプションをレイヤースキャナーに渡すことができます。スキャナーは、そのように設計されていると、構築時にこの設定を渡します。
.scanner.dist	String	特定のスキャナーの名前と値として任意の YAML を持つマップ。
.scanner.package	String	特定のスキャナーの名前と値として任意の YAML を持つマップ。
.scanner.repo	String	特定のスキャナーの名前と値として任意の YAML を持つマップ。

インデクサー設定の例
次の例は、Clair の仮のインデクサー設定を示しています。

インデクサー設定の例

```
# ...
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 設定フィールド

次の表では、Clair の **matcher** コンポーネントの設定フィールドについて説明します。

注記

matchers 設定フィールドとは異なります。

フィールド	型	説明
matcher	Object	Clair matcher ノード設定を提供します。

フィールド	型	説明
.cache_age	String	レスポンスをキャッシュするように、ユーザーに通知する期間を制御します。
.connstring	String	Postgres 接続文字列。URL または libpq 接続文字列として形式を受け入れます。
.max_conn_pool	Integer	データベース接続プールのサイズを制限します。 Clair では、カスタムの接続プールサイズを使用できます。この数は、同時に許可されるアクティブなデータベース接続の数を直接設定します。 このパラメーターは、将来のバージョンでは無視されます。ユーザーは、接続文字列を使用して、これを設定する必要があります。
.indexer_addr	String	matcher は indexer に接続して脆弱性レポートを作成します。このインデクサーの場所は必須です。 デフォルトは **30m** です。
.migrations	Boolean	matcher ノードがデータベースへの移行を処理するかどうか。
.period	String	新しいセキュリティーアドバイザリーの更新頻度を決定します。 デフォルトは **30m** です。
.disable_updaters	Boolean	バックグラウンド更新を実行するかどうか。 デフォルト: **False**

フィールド	型	説明
.update_retention	Integer	ガベージコレクションサイクル間で保持する更新操作の数を設定します。これは、データベースサイズの制約に基づいて安全な MAX 値に設定する必要があります。 デフォルトは **10m** です。 **0** 未満の値を指定すると、ガベージコレクションが無効になります。**2** は、更新を通知と比較できるようにするための最小値です。

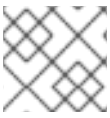
matcher 設定の例

matcher 設定の例

```
# ...
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
# ...
```

9.6. CLAIR MATCHERS 設定フィールド

次の表では、Clair の **matchers** コンポーネントの設定フィールドについて説明します。



注記

matcher 設定フィールドとは異なります。

表9.1 matchers 設定フィールド

フィールド	型	説明
matchers	文字列の配列。	ツリー内 **matchers** の設定を提供します。

フィールド	型	説明
.names	String	有効な matchers について matcher ファクトリーに通知する文字列値のリスト。値を **null** に設定すると、matchers のデフォルトのリストが実行されます。次の文字列が受け入れられます。**alpine-matcher**、**aws-matcher**、**debian-matcher**、**gobin**、**java-maven**、**oracle**、**photon**、**python**、**rhel**、**rhel-container-matcher**、**Ruby**、**suse**、**ubuntu-matcher**
.config	String	特定の matcher に設定を提供します。 matchers ファクトリーコンストラクターに提供されるサブオブジェクトを含む matcher の名前をキーとするマップ。以下に例を示します。

matchers 設定の例

次の例は、**alpine**、**aws**、**debian**、**oracle** matchers のみを必要とする仮の Clair デプロイメントを示しています。

matchers 設定の例

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

9.7. CLAIR アップデーター設定フィールド

次の表では、Clair の **updaters** コンポーネントの設定フィールドについて説明します。

表9.2 アップデーター設定フィールド

フィールド	型	説明
updaters	Object	matcher の更新マネージャーの設定を提供します。

フィールド	型	説明
.sets	String	どのアップデーターを実行するかを更新マネージャーに通知する値のリスト。 値を **null** に設定すると、アップデーターのデフォルトのセットが、**alpine**、**aws**、**clair.cvss**、**debian**、**oracle**、**photon**、**osv**、**rhel**、**rhcc suse**、**ubuntu** を実行します。 空白のままにすると、アップデーターは実行されません。
.config	String	特定のアップデーターセットに設定を提供します。 アップデーターセットのコンストラクターに提供されるサブオブジェクトを含むアップデーターセットの名前をキーとするマップ。各アップデーターのサブオブジェクトのリストについては、「詳細なアップデーター設定」を参照してください。

アップデーター設定の例
次の設定では、**rhel** セットのみが設定されます。**rhel** アップデーターに固有の **ignore_unpatched** 変数も定義されています。

アップデーター設定の例

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

9.8. CLAIR ノーティファイアー設定フィールド

Clair の一般的なノーティファイアー設定フィールドを以下に示します。

フィールド	型	説明

フィールド	型	説明
notifier	Object	Clair ノーティファイアーノード設定を提供します。
.connstring	String	Postgres 接続文字列。形式を URL または libpq 接続文字列として受け入れます。
.migrations	Boolean	ノーティファイアーノードがデータベースへの移行を処理するかどうか。
.indexer_addr	String	ノーティファイアーはインデクサーに接続して、脆弱性の影響を受けるマニフェストを作成または取得します。このインデクサーの場所は必須です。
.matcher_addr	String	ノーティファイアーは matcher に接続して、更新操作をリストし、差分を取得します。この matcher の場所は必須です。
.poll_interval	String	ノーティファイアーが matcher に更新操作をクエリーする頻度。
.delivery_interval	String	ノーティファイアーが、作成された通知または以前に失敗した通知の配信を試行する頻度。
.disable_summary	Boolean	通知をマニフェストごとに 1 つに要約するかどうかを制御します。

ノーティファイアー設定の例

次の **notifier** スニペットは、最小設定用です。

ノーティファイアー設定の例

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

9.8.1. Clair Webhook 設定フィールド

次の Webhook フィールドを Clair ノーティファイアー環境で使用できます。

表9.3 Clair Webhook フィールド

.webhook	Object	Webhook 配信のノーティファイアーを設定します。
.webhook.target	String	Webhook が配信される URL。
.webhook.callback	String	通知を取得できるコールバック URL。この URL に通知 ID が追加されます。 これは通常、Clair ノーティファイアーがホスティングされている場所です。
.webhook.headers	String	ヘッダー名を値のリストに関連付けるマップ。

Webhook 設定の例

Webhook 設定の例

```
# ...
notifier:
# ...
  webhook:
    target: "http://webhook/"
    callback: "http://clair-notifier/notifier/api/v1/notifications"
# ...
```

9.8.2. Clair amqp 設定フィールド

次の Advanced Message Queuing Protocol (AMQP) フィールドを Clair ノーティファイアー環境で使用できます。

.amqp	Object	AMQP 配信のノーティファイアーを設定します。 [注記] ==== Clair は独自に AMQP コンポーネントを宣言しません。エクスチェンジまたはキューを使用しようとするすべての試みは、パッシブのみであり、失敗します。ブローカー管理者は、事前にエクスチェンジとキューをセットアップする必要があります。====
.amqp.direct	Boolean	**true** の場合、ノーティファイアーは設定された AMQP ブローカーに個別の通知 (コールバックではない) を配信します。
.amqp.rollup	Integer	**amqp.direct** が **true** に設定されている場合、この値は直接配信で送信する通知の数をノーティファイアーに通知します。たとえば、**direct** が **true** に設定され、**amqp.rollup** が **5** に設定されている場合、ノーティファイアーは単一の JSON ペイロードで 5 つ以下の通知をブローカーに配信します。値を **0** に設定すると、実質的に **1** に設定されます。
.amqp.exchange	Object	接続先の AMQP エクスチェンジ。
.amqp.exchange.name	String	接続先のエクスチェンジの名前。
.amqp.exchange.type	String	エクスチェンジのタイプ。通常は、**direct**、**fanout**、**topic**、**headers** のいずれかです。
.amqp.exchange.durability	Boolean	設定されたキューが永続的かどうか。
.amqp.exchange.auto_delete	Boolean	設定されたキューが **auto_delete_policy** を使用するかどうか。
.amqp.routing_key	String	各通知が送信されるルーティングキーの名前。

.amqp.callback	String	**amqp.direct** が **false** に設定されている場合、この URL はブローカーに送信される通知コールバックで提供されます。この URL は、Clair の通知 API エンドポイントを指している必要があります。
.amqp.uris	String	接続先の 1 つ以上の AMQP ブローカーのリスト (優先順位順)。
.amqp.tls	Object	AMQP ブローカーへの TLS/SSL 接続を設定します。
.amqp.tls.root_ca	String	ルート CA を読み取ることができるファイルシステムパス。
.amqp.tls.cert	String	TLS/SSL 証明書を読み取ることができるファイルシステムパス。 [注意] ==== Go **crypto/x509** パッケージに記載されているように、Clair は **SSL_CERT_DIR** も許可します。====
.amqp.tls.key	String	TLS/SSL 秘密鍵を読み取ることができるファイルシステムパス。

AMQP 設定の例

次の例は、Clair の仮の AMQP 設定を示しています。

AMQP 設定の例

```
# ...
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 設定フィールド

次の Simple Text Oriented Message Protocol (STOMP) フィールドを Clair ノーティファイアー環境で使用できます。

.stomp	Object	STOMP 配信のノーティファイアーを設定します。
.stomp.direct	Boolean	**true** の場合、ノーティファイアーは個別の通知 (コールバックではない) を設定済みの STOMP ブローカーに配信します。
.stomp.rollup	Integer	**stomp.direct** が **true** に設定されている場合、この値は、1 回の直接配信で送信される通知の数を制限します。たとえば、**direct** が **true** に設定され、**rollup** が **5** に設定されている場合、ノーティファイアーは単一の JSON ペイロードで 5 つ以下の通知をブローカーに配信します。値を **0** に設定すると、実質的に **1** に設定されます。
.stomp.callback	String	**stomp.callback** が **false** に設定されている場合は、通知コールバックで指定された URL がブローカーに送信されます。この URL は、Clair の通知 API エンドポイントを指している必要があります。
.stomp.destination	String	通知を配信する STOMP の宛先。
.stomp.uris	String	接続先の 1 つ以上の STOMP ブローカーのリスト (優先順位順)。
.stomp.tls	Object	STOMP ブローカーへの TLS/SSL 接続を設定しました。
.stomp.tls.root_ca	String	ルート CA を読み取ることができるファイルシステムパス。 [注意] ==== Go **crypto/x509** パッケージに記載されているように、Clair は **SSL_CERT_DIR** も受け入れます。====
.stomp.tls.cert	String	TLS/SSL 証明書を読み取ることができるファイルシステムパス。

.stomp	Object	STOMP 配信のノーティファイアーを設定します。
.stomp.tls.key	String	TLS/SSL 秘密鍵を読み取ることができるファイルシステムパス。
.stomp.user	String	STOMP ブローカーのログインの詳細を設定します。
.stomp.user.login	String	接続に使用する STOMP ログイン。
.stomp.user.passcode	String	接続に使用する STOMP パスコード。

STOMP 設定の例

次の例は、Clair の仮の STOMP 設定を示しています。

STOMP 設定の例

```
# ...
notifier:
# ...
  stomp:
    desitnation: "notifications"
    direct: false
    callback: "http://clair-notifier/notifier/api/v1/notifications"
    login:
      login: "username"
      passcode: "passcode"
    tls:
     root_ca: "optional/path/to/rootca"
     cert: "madatory/path/to/cert"
     key: "madatory/path/to/key"
# ...
```

9.9. CLAIR 認可設定フィールド

Clair では、次の認可設定フィールドを使用できます。

フィールド	型	説明
auth	Object	Clair の外部およびサービス内 JWT ベースの認証を定義します。複数の **auth** 機能が定義されている場合、Clair は 1 つを選択します。現在、複数の機能はサポートされていません。
.psk	String	事前共有キー認証を定義します。

フィールド	型	説明
.psk.key	String	JWT の署名と検証を行うすべての当事者間で配布される、base64 でエンコードされた共有キー。
.psk.iss	String	確認する JWT 発行者のリスト。空のリストは、JWT クレームで任意の発行者を受け入れます。

認可設定の例
次の **authorization** スニペットは、最小設定用です。

認可設定の例

```
# ...
auth:
  psk:
    key: MTU5YzA4Y2ZkNzJoMQ== 1
    iss: ["quay"]
# ...
```

9.10. CLAIR トレース設定フィールド

Clair では、次のトレース設定フィールドを使用できます。

フィールド	型	説明
trace	Object	OpenTelemetry に基づいて分散トレース設定を定義します。
.name	String	トレースが属するアプリケーションの名前。
.probability	Integer	トレースが発生する確率。
.jaeger	Object	Jaeger トレースの値を定義します。
.jaeger.agent	Object	Jaeger エージェントへの配信を設定するための値を定義します。
.jaeger.agent.endpoint	String	トレースを送信できる **<host>:<post>** 構文のアドレス。
.jaeger.collector	Object	Jaeger コレクターへの配信を設定するための値を定義します。

フィールド	型	説明
.jaeger.collector.endpoint	String	トレースを送信できる **<host>:<post>** 構文のアドレス。
.jaeger.collector.username	String	Jaeger ユーザー名。
.jaeger.collector.password	String	Jaeger パスワード。
.jaeger.service_name	String	Jaeger に登録されているサービス名。
.jaeger.tags	String	追加のメタデータを提供するキーと値のペア。
.jaeger.buffer_max	Integer	保管および分析のために Jaeger バックエンドに送信される前にメモリーにバッファーできるスパンの最大数。

トレース設定の例
次の例は、Clair の仮のトレース設定を示しています。

トレース設定の例

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

9.11. CLAIR メトリクス設定フィールド

Clair では、次のメトリクス設定フィールドを使用できます。

フィールド	型	説明
metrics	Object	OpenTelemetry に基づいて分散トレース設定を定義します。
.name	String	使用中のメトリクスの名前。
.prometheus	String	Prometheus メトリクスエクスポーターの設定。

フィールド	型	説明
.prometheus.endpoint	String	メトリクスが提供されるパスを定義します。

メトリクス設定の例
次の例は、Clair の仮のメトリクス設定を示しています。

メトリクス設定の例

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