Red Hat Quay 3.11

Quay IO について

Quay IO について

Last Updated: 2024-04-12

Red Hat Quay 3.11 Quay IO について

Quay IO について

法律上の通知

概要

Quay IO について

目次

はじめに

この包括的なガイドは、堅牢で機能豊富なコンテナーレジストリーサービス Quay.io を最大限に活用するために必要な知識とツールをユーザーに提供します。

第1章 QUAY.IO の概要

Quay.io は、コンテナーイメージやその他の OCI アーティファクトを保存、構築、配布するためのレジストリーです。この堅牢で機能豊富なコンテナーレジストリーサービスは、開発者、組織、企業の間で広く人気を博し、コンテナー化エコシステムにおける先駆的なプラットフォームの 1 つとしての地位を確立しています。さまざまなユーザーのニーズに応えるために、無料層と有料層の両方を提供します。

Quay.io は、その中核として、コンテナーイメージを保存、管理、配布するための集中リポジトリーとして機能します。Quay.io の主な利点の 1 つは、その柔軟性と使いやすさです。ユーザーがコンテナーイメージを迅速にアップロードして管理できる直感的な Web インターフェイスを提供します。開発者はプライベートリポジトリーを作成して、機密コードや独自コードを組織内で安全に保つことができます。さらに、ユーザーはアクセス制御を設定し、チームのコラボレーションを管理できるため、指定されたチームメンバー間でコンテナーイメージをシームレスに共有できます。

Quay.io は、統合イメージスキャナー Clair を介してコンテナーのセキュリティー問題を対処します。このサービスは、既知の脆弱性やセキュリティーの問題についてコンテナーイメージを自動的にスキャンし、潜在的なリスクに関する貴重な洞察を開発者に提供し、修復手順を提案します。

Quay.io は自動化に優れており、一般的な継続的インテグレーション/継続的デプロイメント (CI/CD) ツールおよびプラットフォームとの統合をサポートしており、コンテナーのビルドおよびデプロイメントプロセスのシームレスな自動化を可能にします。その結果、開発者はワークフローを合理化し、手動介入を大幅に削減し、全体的な開発効率を向上させることができます。

Quay.io は、大規模と小規模の両方のデプロイメントのニーズに対応します。その堅牢なアーキテクチャーと高可用性のサポートにより、組織はミッションクリティカルなアプリケーションに信頼できることが保証されます。このプラットフォームは、大量のコンテナーイメージトラフィックを処理でき、コンテナーイメージをさまざまな地理的場所に配信するための効率的なレプリケーションおよび配布メカニズムを提供します。

Quay.io は、コンテナー愛好家のためのアクティブなハブとしての地位を確立しています。開発者は、他のユーザーが共有している構築済みのパブリックコンテナーイメージの膨大なコレクションを発見できるため、プロジェクトに役立つツール、アプリケーション、サービスを簡単に見つけることができます。このオープン共有エコシステムはコラボレーションを促進し、コンテナーコミュニティー内でのソフトウェア開発を加速します。

ソフトウェア開発環境においてコンテナー化が勢いを増し続ける中、Quay.io は常に最前線に立ち、サービスの改善と拡張を継続的に行っています。このプラットフォームのセキュリティー、使いやすさ、自動化、コミュニティーへの関与への取り組みにより、個人の開発者と大規模組織の両方にとって推奨されるコンテナーレジストリーサービスとしての地位が確固たるものになりました。

テクノロジーが進化するにつれて、Quay.io プラットフォームの最新の機能や更新を、公式 Web サイトやその他の信頼できるソースを通じて確認することが重要になります。あなたが個人の開発者であっても、チームの一員であっても、あるいは企業の代表であっても、Quay.io はコンテナー化エクスペリエンスを強化し、最新のアプリケーションを簡単に構築およびデプロイするための作業を合理化できます。

第2章 QUAY.IO のサポート

テクニカルサポートは、Quay.io コンテナーレジストリーサービスの重要な側面であり、コンテナーイメージの管理だけでなく、ホストされるプラットフォームの機能と可用性の確保も支援します。

機能関連の問題を抱えるユーザーを支援するために、Red Hat は Quay.io の顧客にいくつかのリソースへのアクセスを提供しています。Red Hat ナレッジベース には、Red Hat の製品とテクノロジーの可能性を最大限に引き出す貴重なコンテンツが含まれています。ユーザーは、Red Hat 製品のインストール、設定、利用に関するベストプラクティスを概説する記事、製品ドキュメント、およびビデオをご利用になれます。また、既知の問題に対する解決策のハブとしても機能し、根本原因の簡潔な説明と修復手順を提供します。

さらに、Quay.io の顧客は、質問に対処し、問題のトラブルシューティングを行い、プラットフォームのエクスペリエンスを最適化するためのソリューションを提供するテクニカルサポートチームを頼ることができます。特定の機能の理解、設定のカスタマイズ、コンテナーイメージのビルドの問題の解決など、サポートチームは明確さと専門知識を持って各ステップをユーザーにガイドすることに専念しています。

有料サービスをお使いのお客様は、Quay.io ステータスページ に記載されていないサービスの中断やパフォーマンスの問題に関連するインシデント (可用性や機能に関する懸念など) について、Red Hat カスタマーポータル からテクニカルサポートチケットを発行できます。サービスインシデントは、プラットフォームの複数のユーザーに影響を及ぼす、計画外のサービスの中断またはサービス品質低下として定義されます。

この包括的な技術サポートシステムを導入することで、Quay.io はユーザーが自信を持ってコンテナーイメージを管理し、プラットフォームエクスペリエンスを最適化し、発生する可能性のある課題を克服できるようにします。

関連情報

現在の Red Hat Quay および Quay.io ユーザーは、トラブルシューティングとサポートの詳細ついて、Red Hat Quay トラブルシューティングガイド を参照してください。

第3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 Quay.io にパッケージ化されており、自動的に有効になり、Red Hat Quay 開発チームによって管理されます。

Quay.io ユーザーの場合、イメージはリポジトリーにプッシュされた後、自動的にインデックスが作成されます。次に、Clair からレポートが取得され、Clair はイメージを CVE のデータベースと照合してセキュリティー情報を報告します。このプロセスは Quay.io で自動的に行われるため、手動での再スキャンは必要ありません。

3.1. 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。

3.1.1. 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

3.1.2. 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。

3.2. 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。

3.2.1. 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
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
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3.1 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3.2 CVSSv2

ベーススコア	Clair の重大度
0.0-3.9	Low
4.0-6.9	Medium
7.0-10	High

第4章 QUAY.IO ユーザーインターフェイスの概要

Quay.io のユーザーインターフェイス (UI) は、プラットフォームのエコシステム内でコンテナーイメージを管理および操作するためのユーザーのゲートウェイとして機能する基本コンポーネントです。Quay.io の UI は、直感的でユーザーフレンドリーなインターフェイスを提供するように設計されており、あらゆるスキルレベルのユーザーが Quay.io の機能を簡単に操作して利用できるようになります。

このドキュメントセクションは、Quay.io の UI の主要な要素と機能をユーザーに紹介することを目的としています。UI のレイアウト、ナビゲーション、主要な機能などの重要な側面をカバーし、ユーザーが Quay.io のコンテナーレジストリーサービスを探索して最大限に活用するための強固な基盤を提供します。

このドキュメント全体を通じて、次のトピックに関する段階的な手順、視覚的補助、および実践的な例が提供されます。

- アプリケーションとリポジトリーの探索
- Quay.io チュートリアルの使用
- 価格と Quay.io プラン
- サインインして Quay.io 機能を使用する

このドキュメントをまとめると、ユーザーが UI の微妙な違いをすぐに把握し、Quay.io を使用してコンテナー化の手順をうまく進めることができるようになります。

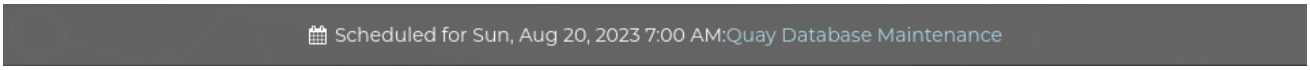
4.1. QUAY.IO ランディングページ

Quay.io ランディングページは、ユーザーが提供されるコンテナーレジストリーサービスにアクセスするための中央ハブとして機能します。このページには、ユーザーがコンテナーイメージを簡単に安全に保存、構築、デプロイできるようガイドするための重要な情報とリンクが提供されます。

Quay.io のランディングページには、次のリソースへのリンクが含まれています。

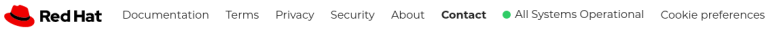
- Explore このページでは、Quay.io データベースでさまざまなアプリケーションやリポジトリーを検索できます。
- Tutorial このページでは、Quay.io の使用方法を段階的に説明します。
- Pricing このページでは、Quay.io に提供されるさまざまな価格帯について学ぶことができます。このページにはさまざまな FAQ も掲載されています。
- Sign in このリンクをクリックすると、Quay.io リポジトリーにサインインするようにリダイレクトされます。

ランディングページには、定期メンテナンスに関する情報も含まれています。定期メンテナンス中、Quay.io は読み取り専用モードで動作し、プル機能は通常どおり動作します。スケジュールされたメンテナンス中は、プッシュとビルドは動作しなくなります。Quay.io の **Status** ページ に移動し、**Subscribe To Updates** をクリックすると、Quay.io のメンテナンスに関する更新を購読できます。



ランディングページには、次のリソースへのリンクも含まれています。

- Documentationこのページでは、Quay.io の使用に関するドキュメントを提供します。
- Terms。このページでは、Red Hat Online Services に関する法的情報を提供します。
- Privacyこのページには、Red Hat のプライバシーに関する声明に関する情報が記載されています。
- セキュリティー。このページでは、SSL/TLS、暗号化、パスワード、アクセス制御、ファイアウォール、データ復元力など、Quay.io のセキュリティーに関する情報を提供します。
- Aboutこのページには、使用されているパッケージとプロジェクトに関する情報、および製品の簡単な歴史が含まれています。
- Contactこのページには、サポートと Red Hat サポートチームへの連絡先に関する情報が含まれています。
- All Systems Operational。このページには、Quay.io のステータスとメンテナンスの簡単な履歴に関する情報が含まれています。
- クッキー。このリンクをクリックすると、ポップアップボックスが表示され、Cookie の設定を行うことができます。



また、オンプレミスで Red Hat Quay を試す または クラウドで Red Hat Quay を試す に関する情報も表示され、**Pricing** ページにリダイレクトされます。各オプションには無料トライアルが用意されています。

4.1.1. Quay.io アカウントの作成

Quay.io の新規ユーザーは、Red Hat アカウントに登録 し、Quay.io ユーザー名を作成する必要があります。これらのアカウントは相関していますが、次の 2 つの明確な違いがあります。

- Quay.io アカウントを使用して、コンテナーイメージまたは Open Container Initiative イメージを Quay.io にプッシュおよびプルしてイメージを保存できます。
- Red Hat アカウントは、ユーザーに Quay.io ユーザーインターフェイスへのアクセスを提供します。有料サービスをご利用の場合は、このアカウントを使用して Red Hat Ecosystem Catalog のイメージにアクセスすることもでき、Quay.io リポジトリーにプッシュできます。

ユーザーはまず Red Hat アカウントに登録し、次に Quay.io アカウントを作成する必要があります。ユーザーが Quay.io のすべての機能を適切に使用するには、両方のアカウントが必要です。

4.1.1.1. Red Hat アカウントの登録

Quay.io の Red Hat アカウントを登録するには、次の手順を使用します。

手順

1. Red Hat カスタマーポータル に移動します。
2. ナビゲーションペインで、**Log In** をクリックします。
3. ログインページに移動したら、**Register for a Red Hat Account**をクリックします。

4. Red Hat のログイン ID を入力します。
5. パスワードを入力します。
6. 次の個人情報を入力します。
    - **名**
    - **姓**
    - **メールアドレス**
    - **電話番号**
7. お住まいの国または地域に応じた次の連絡先情報を入力してください。以下に例を示します。
    - **Country/region**
    - **アドレス**
    - **Postal code**
    - **City**
    - **County**
8. Red Hat の利用規約を選択して同意します。
9. **Create my account** をクリックします。
10. Quay.io に移動してログインします。

4.1.1.2. Quay.io ユーザーアカウントの作成

Quay.io ユーザーアカウントを作成するには、次の手順を使用します。

前提条件

- Red Hat アカウントが作成されている。

手順

1. 必要に応じて、**I am not a robot** をクリックして確認し、キャプチャーを解決します。**Confirm Username** ページにリダイレクトされます。
2. **Confirm Username** ページで、ユーザー名を入力します。デフォルトでは、ユーザー名が生成されます。同じユーザー名がすでに存在する場合は、一意になるように最後に番号が追加されます。このユーザー名は、Quay Container Registry の namespace として使用されます。
3. ユーザー名を決定したら、**Confirm Username** をクリックします。Quay.io **Repositories** ページにリダイレクトされます。このページは、ユーザーがリポジトリーに簡単にアクセスして管理できる専用ハブとして機能します。このページから、ユーザーはコンテナーイメージと関連リソースを効率的に整理、移動、操作できます。

4.1.1.3. Quay.io シングルサインオンのサポート

Red Hat シングルサインオン (SSO) は Quay.io で使用できます。Quay.io で Red Hat SSO をセット

アップするには、次の手順を使用します。ほとんどのユーザーにとって、これらのアカウントはすでにリンクされています。ただし、一部の従来の Quay.io ユーザーの場合は、この手順が必要になる場合があります。

前提条件

- Quay.io アカウントが作成されている。

手順

1. Quay.io の **Recovery** ページ に移動します。
2. ユーザー名とパスワードを入力し、**Sign in to Quay Container Registry**をクリックします。
3. ナビゲーションペインで、ユーザー名 → **Account Settings** をクリックします。
4. ナビゲーションウィンドウで、**External Logins and Applications** をクリックします。
5. **Attach to Red Hat** をクリックします。
6. すでに Red Hat SSO にサインインしている場合、アカウントは自動的にリンクされます。それ以外の場合は、Red Hat ログイン名または電子メールとパスワードを入力して、Red Hat SSO にサインインするように求められます。または、最初に新しいアカウントを作成しないといけない場合があります。
   Red Hat SSO にサインインした後、ログインページから Red Hat アカウントを使用して Quay.io に対して認証することを選択できます。

関連情報

- 詳細は、Quay.io Now Supports Red Hat Single Sign On を参照してください。

4.1.2. Quay.io を探索する

Quay.io の **Explore** ページは、ユーザーが Quay.io コミュニティーによって共有されているコンテナーイメージ、アプリケーション、およびリポジトリーの膨大なコレクションを詳しく調べることができる貴重なハブです。直感的でユーザーフレンドリーなデザインの **Explore** ページには強力な検索機能があり、ユーザーはコンテナー化されたアプリケーションやリソースを簡単に見つけることができます。

4.1.3. Quay.io を試す (非推奨)

注記

Red Hat Quay チュートリアルは現在非推奨となっており、v2 UI が一般公開 (GA) されると削除される予定です。

Quay.io の **Tutorial** ページでは、ユーザーと Quay.io コンテナーレジストリーサービスの概要が提供されます。**Continue Tutorial** をクリックすると、Quay.io で次の機能を実行する方法を学ぶことができます。

- Docker CLI から Quay Container Registry にログインする
- コンテナーの起動
- コンテナーからイメージを作成する

- リポジトリーを Quay Container Registry にプッシュする
- リポジトリーの表示
- ビルドトリガーのセットアップ
- リポジトリーの権限の変更

4.1.4. Quay.io の価格に関する情報

無料利用枠に加えて、Quay.io は特典を強化したいくつかの有料プランも提供しています。

Quay.io の **Pricing** ページには、Quay.io プランと各プランの関連価格に関する情報が表示されます。各段階のコストは、Pricing ページで確認できます。すべての Quay.io プランには次の利点が含まれます。

- 継続的インテグレーション
- パブリックリポジトリー
- ロボットアカウント
- チーム
- SSL/TLS 暗号化
- ロギングおよび監査
- 請求書履歴

Quay.io サブスクリプションは、Stripe 支払い処理プラットフォームによって処理されます。Quay.io にサインアップするには、有効なクレジットカードが必要です。

Quay.io にサインアップするには、次の手順を実行します。

手順

1. Quay.io の **Pricing** ページ に移動します。
2. プラン (例: **Small**) を決定し、**Buy Now** をクリックします。**Create New Organization** ページにリダイレクトされます。以下の情報を入力します。
   - **Organization Name**
   - **組織の電子メール**
   - オプション: たとえば、**Small** より大きいプランが必要な場合は、別のプランを選択できます。
3. そのキャプチャーを解決し、**Create Organization** を選択します。
4. Stripe にリダイレクトされます。以下の情報を入力します。
   - **MM/YY** および **CVC** を含む **Card information**
   - **Name on card**

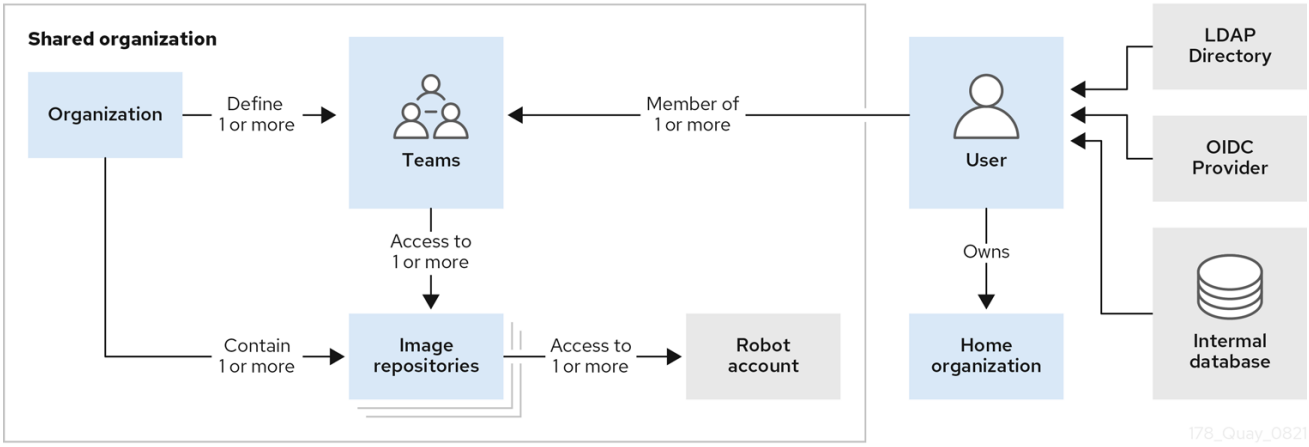
- **国または地域**
- **ZIP** (該当する場合)
- 情報を保存したい場合は、チェックボックスをオンにします。
- **電話番号**

5. すべてのボックスに入力したら、**Subscribe** をクリックします。

第5章 ユーザーと組織

Quay.io でコンテナーイメージを追加するリポジトリーを作成する前に、そのリポジトリーをどのように構成するかを検討する必要があります。Quay.io では、各リポジトリーに **Organization** または **User** のいずれかとの接続が必要です。この関係により、リポジトリーの所有権とアクセス制御が定義されます。

5.1. テナンシーモデル



- **組織** は、単一のユーザーに属さない共通の名前空間のもとでリポジトリーを共有する方法を提供します。このようなリポジトリーは、会社などの共有設定内の複数のユーザーに属します。
- **チーム** は、組織から権限を委任する方法を提供します。権限は、グローバルレベル (たとえば、すべてのリポジトリー全体) で設定することも、特定のリポジトリーに対して設定することもできます。特定のユーザーのセットまたはグループに対して設定することもできます。
- **ユーザー** は、Web UI を介してレジストリーにログインすることも、Podman や Docker などのクライアントを使用して、それぞれのログインコマンド (**$ podman login** など) を使用してレジストリーにログインすることもできます。各ユーザーは、ユーザー名前空間 **<quay-server.example.com>/<user>/<username>**、**quay.io/<username>** など) を自動的に取得します。
- **ロボットアカウント** は、パイプラインツールなどの人間以外のユーザーにリポジトリーへの自動アクセスを提供します。ロボットアカウントは OpenShift Container Platform の **サービスアカウント** に似ています。リポジトリー内のロボットアカウントに権限を付与するには、そのアカウントを他のユーザーやチームと同様に追加します。

5.2. キーへのログイン

Quay.io のユーザーアカウントは、プラットフォームの機能への認証アクセス権を持つ個人を表します。このアカウントを通じて、リポジトリーの作成と管理、コンテナーイメージのアップロードと取得、およびこれらのリソースへのアクセス権の制御を行うことができます。このアカウントは、Quay.io 内でのコンテナーイメージの管理を組織化および監視するうえで極めて重要です。

注記

Quay.io のすべての機能でユーザーのログインが必要なわけではありません。たとえば、プルしているイメージがパブリックリポジトリーから取得されている限り、ログインせずに Quay.io から匿名でイメージをプルできます。

ユーザーには Quay.io にログインするための 2 つのオプションがあります。

- Quay.io を通じてログインします。
  このオプションは、ユーザーにレガシー UI を提供するだけでなく、PatternFly UI の原則に準拠したベータ UI 環境をユーザーに提供するオプションも提供します。

- Red Hat Hybrid Cloud Console からログインする。
  このオプションは認証に Red Hat SSO を使用し、Red Hat が提供するパブリックマネージドサービスです。このオプションでは、ユーザーは **常に** ログインする必要があります。他のマネージドサービスと同様に、Red Hat Hybrid Cloud Console の Quay は、PatternFly UI 原則に準拠することでユーザーエクスペリエンスを強化します。

Quay.io を直接使用する場合と、Red Hat Hybrid Cloud Console で Quay を使用する場合の違いは、無料枠のユーザーを含め、ごくわずかです。Quay.io を直接使用している場合でも、Hybrid Cloud Console 上で使用している場合でも、リポジトリーへのプッシュなど、ログインが必要な機能には Quay.io ユーザー名の仕様が使用されます。

5.2.1. Quay.io へのログイン

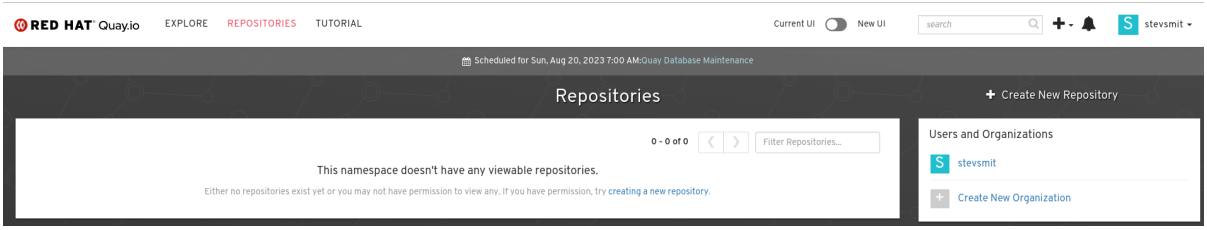
Quay.io にログインするには、次の手順を使用します。

前提条件

- Red Hat アカウントと Quay.io アカウントを作成している。詳細は、「Quay.io アカウントの作成」を参照してください。

手順

1. Quay.io に移動します。
2. ナビゲーションペインで、**Sign In** を選択し、Red Hat 認証情報を使用してログインします。
3. 初めてログインする場合は、自動生成されたユーザー名を確認する必要があります。**ユーザー名の確認** をクリックしてログインします。
   Quay.io リポジトリーのランディングページにリダイレクトされます。



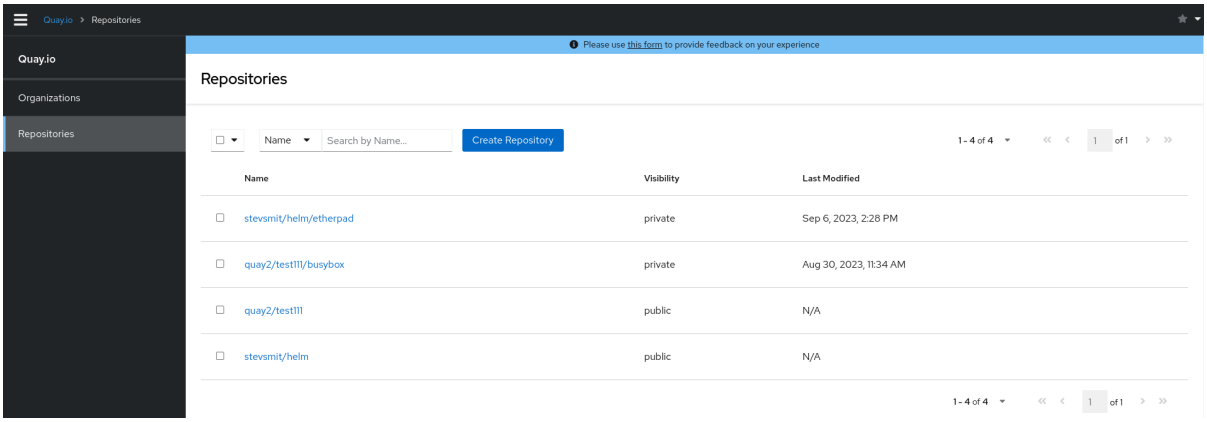
5.2.2. Hybrid Cloud Console を介した Quay へのログイン

前提条件

- Red Hat アカウントと Quay.io アカウントを作成している。詳細は、「Quay.io アカウントの作成」を参照してください。

手順

1. Red Hat Hybrid Cloud Console で Quay に移動し、Red Hat アカウントを使用してログインします。Quay リポジトリーのランディングページにリダイレクトされます。



5.3. リポジトリーの作成

リポジトリーは、関連するコンテナーイメージのセットを一元的に保存するための場所を提供します。これらのイメージを使用して、アプリケーションとその依存関係を標準化された形式で構築できます。

リポジトリーは名前空間を使用して整理します。名前空間には、それぞれ複数のリポジトリーを含めることができます。たとえば、個人プロジェクト用の名前空間、会社用の名前空間、または組織内の特定のチーム用の名前空間を設定できます。

有料プランでは、Quay.io はユーザーにリポジトリーへのアクセス制御を提供します。リポジトリーをパブリックにすると、誰でもリポジトリーからイメージをプルまたはダウンロードできるようになります。リポジトリーをプライベートにすると、許可されたユーザーまたはチームのみにアクセスが制限されます。

注記

Quay.io の無料利用枠では、プライベートリポジトリーは許可されません。プライベートリポジトリーを作成するには、Quay.io の有料層にアップグレードする必要があります。詳細は、「Quay.io の価格に関する情報」を参照してください。

Quay.io でリポジトリーを作成するには 2 つの方法があります。関連する **docker** コマンドまたは **podman** コマンドを使用してイメージをプッシュする方法と、Quay.io UI を使用する方法です。

最初に UI でリポジトリーを作成せずにコマンドラインインターフェイス (CLI) を介してイメージをプッシュすると、プランに関係なく、作成されたリポジトリーは **Private** に設定されます。

注記

イメージをプッシュする前に、Quay.io UI でリポジトリーを作成することを推奨します。Quay.io はプランのステータスをチェックし、プランがアクティブでない場合はプライベートリポジトリーの作成を許可しません。

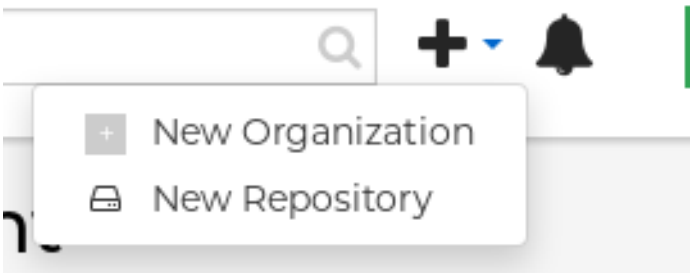
5.3.1. UI を使用したイメージリポジトリーの作成

Quay.io UI を使用してリポジトリーを作成するには、次の手順を実行します。

手順

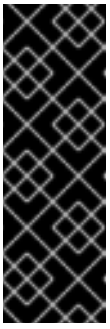
1. Web UI からユーザーアカウントにログインします。

2. Quay.io ランディングページで、**Create New Repository**をクリックします。または、+ アイコン → **New Repository** をクリックすることもできます。以下に例を示します。



3. **Create New Repository**ページで、以下を行います。

   a. 使用するユーザー名または組織に **Repository Name** を追加します。



重要

リポジトリー名には次の単語を使用しないでください。* **build** * **trigger** * **tag**

これらの単語をリポジトリー名に使用すると、ユーザーがリポジトリーにアクセスできなくなり、リポジトリーを完全に削除できなくなります。このようなリポジトリーを削除しようとすると、**Failed to delete repository <repository_name>, HTTP404 - Not Found.** というエラーが返されます。

   b. オプション: **Click to set repository description**をクリックして、リポジトリーの説明を追加します。

   c. ニーズに合わせて、**Public** または **Private** をクリックします。

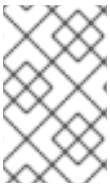
   d. オプション: 必要なリポジトリーの初期化を選択します。

4. **Create Private Repository**をクリックして、新しい空のリポジトリーを作成します。

5.3.2. CLI を使用したイメージリポジトリーの作成

適切な認証情報がある場合は、Docker または Podman を使用して、Quay.io インスタンスにまだ存在しないリポジトリーにイメージを **プッシュ** できます。イメージのプッシュとは、コンテナーイメージをローカルシステムまたは開発環境から Quay.io などのコンテナーレジストリーにアップロードするプロセスを指します。イメージを Quay.io にプッシュすると、リポジトリーが作成されます。

最初に UI でリポジトリーを作成せずにコマンドラインインターフェイス (CLI) を介してイメージをプッシュすると、プランに関係なく、作成されたリポジトリーは **Private** に設定されます。



注記

イメージをプッシュする前に、Quay.io UI でリポジトリーを作成することを推奨します。Quay.io はプランのステータスをチェックし、プランがアクティブでない場合はプライベートリポジトリーの作成を許可しません。

イメージをプッシュしてイメージリポジトリーを作成するには、次の手順を実行します。

前提条件

- **podman** CLI をダウンロードしてインストールしている。

- Quay.io にログインしている。
- イメージ (busybox など) をプルしている。

手順

1. サンプルレジストリーからサンプルページを取得します。以下に例を示します。

```
$ podman pull busybox
```

出力例

```
Trying to pull docker.io/library/busybox...
Getting image source signatures
Copying blob 4c892f00285e done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
22667f53682a2920948d19c7133ab1c9c3f745805c14125859d20cede07f11f9
```

2. ローカルシステム上のイメージに、新しいリポジトリーとイメージ名をタグ付けします。以下に例を示します。

```
$ podman tag docker.io/library/busybox quay.io/quayadmin/busybox:test
```

3. イメージをレジストリーにプッシュします。この手順の後に、ブラウザーを使用して、リポジトリーでタグ付けされたイメージを確認できます。

```
$ podman push --tls-verify=false quay.io/quayadmin/busybox:test
```

出力例

```
Getting image source signatures
Copying blob 6b245f040973 done
Copying config 22667f5368 done
Writing manifest to image destination
Storing signatures
```

5.4. リポジトリーへのアクセス管理

Quay.io ユーザーは、独自のリポジトリーを作成し、インスタンスに含まれる他のユーザーにそのリポジトリーへのアクセスを許可できます。または、特定の組織を作成して、定義されたチームに基づいてリポジトリーへのアクセスを許可することもできます。

ユーザーリポジトリーと組織リポジトリーのどちらでも、ロボットアカウントに関連する認証情報を作成すると、そのリポジトリーへのアクセスを許可できます。ロボットアカウントを使用すると、Quay.io ユーザーアカウントを持っていないさまざまなコンテナークライアント (Docker や Podman など) がリポジトリーに簡単にアクセスできるようになります。

5.4.1. ユーザーリポジトリーへのアクセスの許可

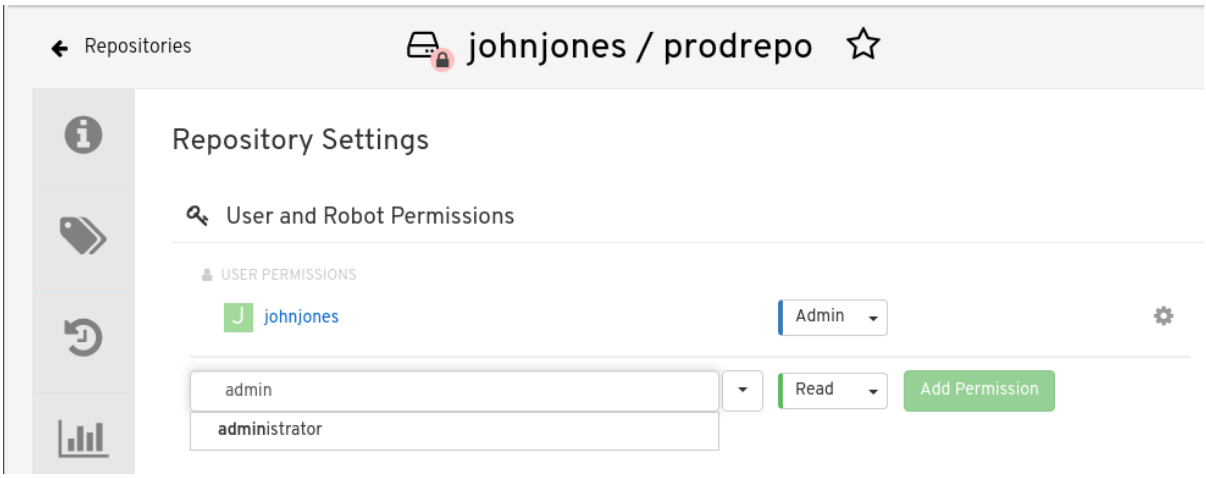
ユーザー名前空間にリポジトリーを作成すると、そのリポジトリーへのアクセスを、ユーザーアカウントに、またはロボットアカウントを通じて追加できます。

5.4.1.1. ユーザーリポジトリーへのユーザーアクセスの許可

ユーザーアカウントに関連付けられたリポジトリーへのアクセスを許可するには、次の手順を実行します。

手順

1. ユーザーアカウントを使用して Quay.io にログインします。
2. 複数のユーザー間で共有されるユーザー名前空間配下のリポジトリーを選択します。
3. ナビゲーションペインで **Settings** を選択します。
4. 自分のリポジトリーへのアクセスを許可するユーザーの名前を入力します。入力すると、名前が表示されます。以下に例を示します。



5. パーミッションボックスで、以下のいずれかを選択します。
   - **Read**。リポジトリーの表示とリポジトリーからのプルをユーザーに許可します。
   - **Write**。リポジトリーの表示、リポジトリーからのプル、リポジトリーへのイメージのプッシュをユーザーに許可します。
   - **Admin**。リポジトリーに対するすべての管理設定と、すべての **Read** および **Write** 権限をユーザーに提供します。
6. **Add Permission** ボタンを選択します。これで、ユーザーに権限が割り当てられました。
7. オプション: リポジトリーに対するユーザー権限を削除または変更するには、**Options** アイコンを選択し、**Delete Permission** を選択します。

5.4.1.2. ユーザーリポジトリーへのロボットアクセスの許可

ロボットアカウントは、Quay.io レジストリー内のリポジトリーへの自動アクセスを設定するために使用されます。ロボットアカウントは OpenShift Container Platform のサービスアカウントに似ています。

ロボットアカウントを設定すると、以下が実行されます。

- ロボットアカウントに関連付けられた認証情報が生成されます。

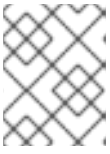
- ロボットアカウントがイメージをプッシュおよびプルできるリポジトリーとイメージが特定されます。
- 生成された認証情報をコピー/ペーストして、Docker、Podman、Kubernetes、Mesos などのさまざまなコンテナークライアントで使用し、定義された各リポジトリーにアクセスできます。

各ロボットアカウントは、1 つのユーザー名前空間または組織に制限されます。たとえば、ロボットアカウントは、ユーザー **jsmith** にすべてのリポジトリーへのアクセスを提供できます。しかし、ユーザーのリポジトリーリストにないリポジトリーへのアクセスは提供できません。

次の手順を使用して、リポジトリーへのアクセスを許可できるロボットアカウントを設定します。

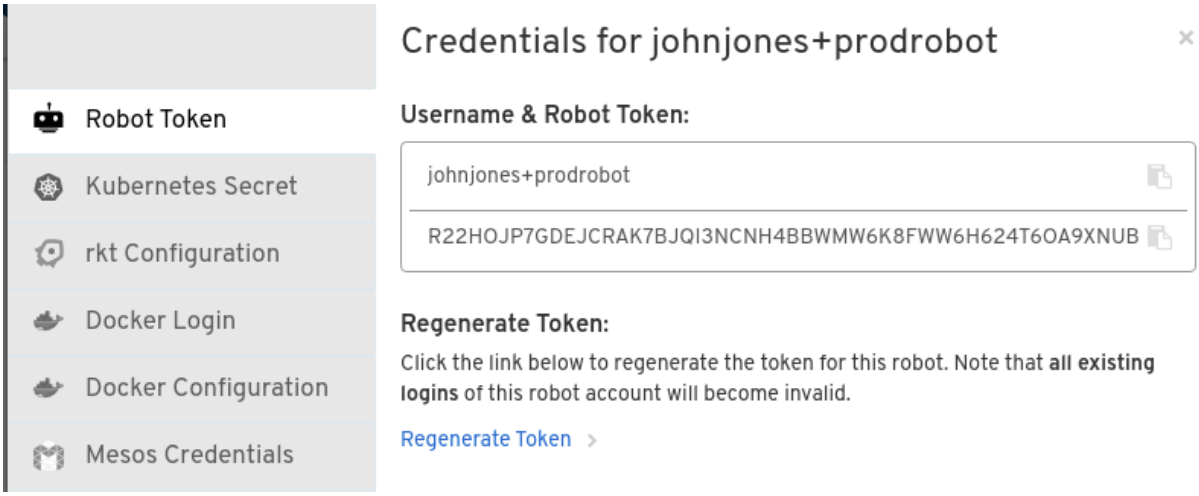
手順

1. **Repositories** ランディングページで、ユーザーの名前をクリックします。
2. ナビゲーションペインで **Robot Accounts** をクリックします。
3. **Create Robot Account** をクリックします。
4. ロボットアカウントの名前を入力します。
5. オプション: ロボットアカウントの説明を入力します。
6. **Create Robot Account** をクリックします。ロボットアカウントの名前は、ユーザー名とロボットの名前を組み合わせたものになります (例: **jsmith+robot)**。
7. ロボットアカウントを関連付けるリポジトリーを選択します。
8. ロボットアカウントの権限を次のいずれかに設定します。
   - **None**。ロボットアカウントにリポジトリーに対する権限は付与されません。
   - **Read**。ロボットアカウントがリポジトリーの表示とリポジトリーからのプルを行えるようになります。
   - **Write**。ロボットアカウントがリポジトリーからの読み取り (プル) とリポジトリーへの書き込み (プッシュ) を行えるようになります。
   - **Admin**。プルおよびプッシュを行うためのリポジトリーへのフルアクセスに加えて、リポジトリーに関連する管理作業を行う権限を付与します。
9. **Add permissions** ボタンをクリックして設定を適用します。
10. **Robot Accounts** ページで、ロボットアカウントを選択して、そのロボットの認証情報を表示します。
11. **Robot Account** オプションで、**Copy to Clipboard** をクリックして、ロボット用に生成されたトークンをコピーします。新しいトークンを生成するには、**Regenerate Token** をクリックします。



注記

トークンを再生成すると、このロボットの以前のトークンがすべて無効になります。



12. 生成された認証情報を次の方法で取得します。

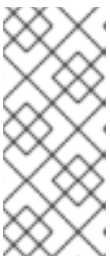
    - **Kubernetes Secret**: Kubernetes プルシークレット yaml ファイルの形式で認証情報をダウンロードするには、これを選択します。
    - **rkt Configuration**: rkt コンテナーランタイムの認証情報を **.json** ファイルの形式でダウンロードするには、これを選択します。
    - **Docker Login**: 認証情報を含む完全な **docker login** コマンドラインをコピーするには、これを選択します。
    - **Docker Configuration**: Docker **config.json** ファイルとして使用するファイルがダウンロードし、クライアントシステムに認証情報を永続的に保存するには、これを選択します。
    - **Mesos Credentials**: Mesos 設定ファイルの URI フィールドで識別できる認証情報を提供する tarball をダウンロードするには、これを選択します。

5.4.2. 組織リポジトリー

組織を作成すると、リポジトリーのセットを直接その組織に関連付けることができます。組織リポジトリーは、組織がユーザーのグループを通じて共有リポジトリーをセットアップすることを目的としているという点で、基本的なリポジトリーとは異なります。Quay.io では、ユーザーのグループは、**チーム**、同じ権限を持つユーザーのセット、または **個々のユーザー** のいずれかです。

組織に関するその他の有用な情報を以下に示します。

- 組織を別の組織内に組み込むことはできません。組織を細分化するには、チームを使用します。
- 組織にユーザーを直接含めることはできません。まずチームを追加してから、各チームに 1 人以上のユーザーを追加する必要があります。



注記

個々のユーザーは、組織内の特定のリポジトリーに追加できます。そのため、これらのユーザーは、**Repository Settings** ページのどのチームのメンバーでもありません。**Teams and Memberships** ページの **Collaborators View** に、特に組織に所属する必要がなく、その組織内の特定のリポジトリーに直接アクセスできるユーザーが表示されます。

- チームは、リポジトリーおよび関連イメージを使用する単なるメンバーとして、または組織を管理するための特別な権限を持つ管理者として、組織内に設定できます。

5.4.2.1. 組織の作成

組織を作成するには、次の手順を実行します。

手順

1. **Repositories** ランディングページで、**Create New Organization** をクリックします。
2. **Organization Name** に、2 文字以上 225 文字未満の名前を入力します。
3. **Organization Email** に、アカウントのメールアドレスとは異なるメールアドレスを入力します。
4. 組織のプランをを、無料プランまたはいずれかの有料プランの中から選択します。
5. **Create Organization** をクリックして作成を完了します。

5.4.2.1.1. API を使用した別の組織の作成

API を使用して別の組織を作成できます。これを行うには、UI を使用して最初の組織を作成しておく必要があります。OAuth アクセストークンも生成しておく必要があります。

Red Hat Quay API エンドポイントを使用して別の組織を作成するには、次の手順を実行します。

前提条件

- UI を使用して少なくとも 1 つの組織をすでに作成している。
- OAuth アクセストークンを生成している。詳細は、「OAuth アクセストークンの作成」を参照してください。

手順

1. 次のコマンドを入力して、**data.json** という名前のファイルを作成します。

```
$ touch data.json
```

2. 次の内容をファイルに追加します。これが新しい組織の名前になります。

```
{"name":"testorg1"}
```

3. 次のコマンドを入力して、OAuth アクセストークンと Red Hat Quay レジストリーエンドポイントを渡し、API エンドポイントを使用して新しい組織を作成します。

```
$ curl -X POST -k -d @data.json -H "Authorization: Bearer <access_token>" -H "Content-Type: application/json" http://<quay-server.example.com>/api/v1/organization/
```

出力例

```
"Created"
```

5.4.2.2. 組織へのチームの追加

組織のためにチームを作成する際に、チーム名を選択し、チームが利用できるリポジトリーを選択し、チームのアクセスレベルを決定できます。

組織のチームを作成するには、次の手順を実行します。

前提条件

- 組織を作成している。

手順

1. **Repositories** ランディングページで、チームを追加する組織を選択します。
2. ナビゲーションペインで、**Teams and Membership** を選択します。デフォルトでは、組織を作成したユーザーの **Admin** 権限を持つ **owners** チームが存在します。
3. **Create New Team** をクリックします。
4. 新しいチームの名前を入力します。チーム名の先頭は小文字である必要があります。また、使用できるのは小文字と数字のみです。大文字や特殊文字は使用できません。
5. **Create team** をクリックします。
6. チームの名前をクリックすると、**Team** ページにリダイレクトされます。ここで、チームの説明を追加したり、登録ユーザー、ロボット、メールアドレスなどのチームメンバーを追加したりできます。詳細は、「チームへのユーザーの追加」を参照してください。
7. **No repositories** テキストをクリックすると、使用可能なリポジトリーのリストが表示されます。チームにアクセスを許可する各リポジトリーのボックスを選択します。
8. チームに付与する適切な権限を選択します。
    - **None**。チームメンバーにリポジトリーに対する権限は付与されません。
    - **Read**。チームメンバーがリポジトリーの表示とリポジトリーからのプルを行えるようになります。
    - **Write**。チームメンバーがリポジトリーの読み取り (プル) とリポジトリーへの書き込み (プッシュ) を行えるようになります。
    - **Admin**。プルおよびプッシュを行うためのリポジトリーへのフルアクセスに加えて、リポジトリーに関連する管理作業を行う権限を付与します。
9. **Add permissions** を選択して、チームのリポジトリー権限を保存します。

5.4.2.3. チームロールの設定

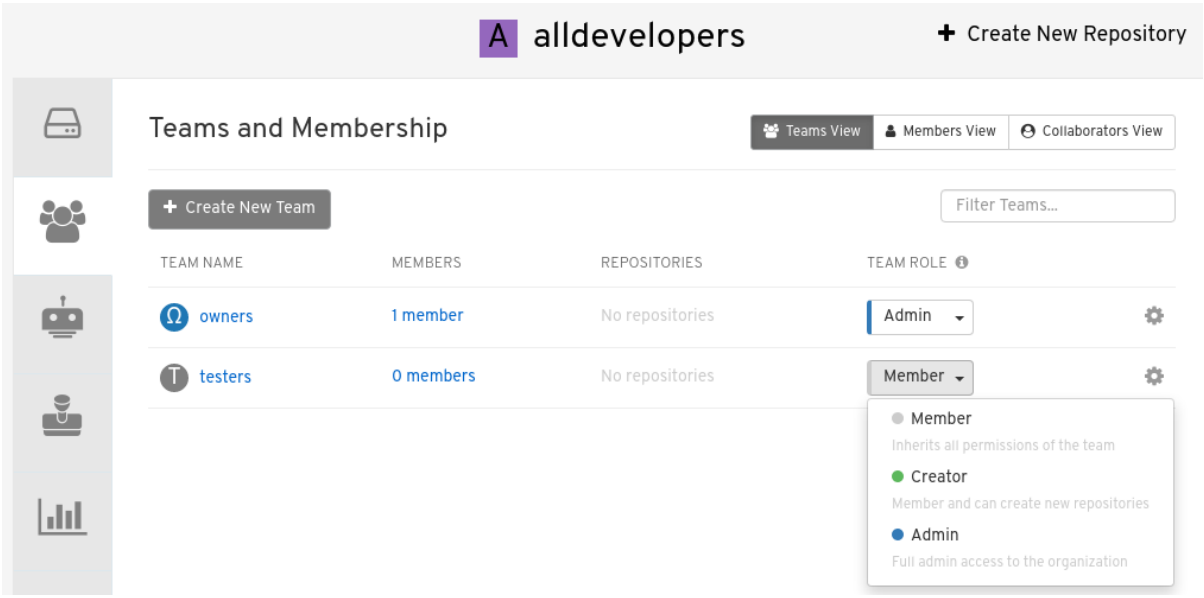
チームを追加したら、そのチームの組織内でのロールを設定できます。

前提条件

- チームを作成している。

手順

1. **Repository** ランディングページで、組織の名前をクリックします。
2. ナビゲーションウィンドウで、**Teams and Membership** をクリックします。
3. 次の図に示すように、**TEAM ROLE** ドロップダウンメニューを選択します。



4. 選択したチームについて、以下のロールのいずれかを選択します。
   - **Member**。チームに設定されているすべての権限を継承します。
   - **Creator**。メンバーのすべての権限に加えて、新しいリポジトリーを作成する権限を付与します。
   - **Admin**。チームの作成、メンバーの追加、権限の設定など、組織への完全な管理アクセス権を付与します。

5.4.2.4. チームへのユーザーの追加

組織に対する管理者権限を持っていれば、ユーザーとロボットアカウントをチームに追加できます。ユーザーを追加すると、Quay.io はそのユーザーに電子メールを送信します。そのユーザーが招待を受け入れるまで、ユーザーは保留状態のままになります。

ユーザーまたはロボットアカウントをチームに追加するには、次の手順を実行します。

手順

1. **Repository** ランディングページで、組織の名前をクリックします。
2. ナビゲーションウィンドウで、**Teams and Membership** をクリックします。
3. ユーザーまたはロボットアカウントを追加するチームを選択します。
4. **Team Members** ボックスに、次のいずれかの情報を入力します。
   - レジストリー上のアカウントからのユーザー名。
   - レジストリー上のユーザーアカウントのメールアドレス。
   - ロボットアカウントの名前。名前は、<組織名>+<ロボット名> の形式である必要があります。

注記

ロボットアカウントはすぐにチームに追加されます。ユーザーアカウントの場合、参加の招待がユーザーにメールで送信されます。ユーザーがその招待を受け入れるまで、ユーザーは **INVITED TO JOIN** 状態のままになります。ユーザーがチームへの参加招待メールを受け入れると、**INVITED TO JOIN** リストから組織の **MEMBERS** リストに移動します。

5.5. USER SETTINGS

User Settings ページでは、電子メールアドレス、パスワード、アカウントタイプの設定、デスクトップ通知の設定、アバターの選択、アカウントの削除、**time machine** 設定の調整、および請求情報の表示を行う方法がユーザーに提供されます。

5.5.1. ユーザー設定ページへの移動

次の手順に従って、**User Settings** ページに移動します。

手順

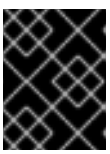
1. Quay.io で、ヘッダーにあるユーザー名をクリックします。
2. **Account Settings** を選択します。**User Settings** ページにリダイレクトされます。

5.5.2. ユーザー設定の調整

ユーザー設定を調整するには、次の手順を使用します。

手順

- 電子メールアドレスを変更するには、**Email Address** で現在の電子メールアドレスを選択します。ポップアップウィンドウで新しい電子メールアドレスを入力し、**Change Email** をクリックします。変更が適用される前に、確認メールが送信されます。
- パスワードを変更するには、**Change password** をクリックします。両方のボックスに新しいパスワードを入力し、**Change Password** をクリックします。
- **Individual Account** をクリックするか、**Account Type** の横のオプションをクリックして、アカウントの種類を変更します。場合によっては、アカウントの種類を変更する前に組織を離れないといけない場合があります。
- デスクトップ通知の横にあるオプションをクリックして、**Desktop Notifications** を調整します。ユーザーはこの機能を有効または無効にできます。
- **Begin deletion** をクリックしてアカウントを削除できます。アクティブなプランがある場合、または自分が唯一の管理者である組織のメンバーである場合は、アカウントを削除できません。namespace を入力して削除を確認する必要があります。



重要

アカウントを削除すると元に戻すことはできず、リポジトリー、作成されたビルドトリガー、通知を含むアカウントのデータがすべて削除されます。

- **Time Machine** の横にあるドロップボックスをクリックして、**time machine** 機能を設定できま

す。この機能は、タグが削除された後、ガベージコレクションが行われるまでにタイムマシンでタグにアクセスできるようになる時間を指定します。時間を選択した後、**Save Expiration Time** をクリックします。

5.5.3. 請求情報

User Settings で請求情報を確認できます。このセクションでは、次の情報が提供されます。

- **Current Plan**。このセクションでは、サインアップしている現在の Quay.io プランを示します。所有しているプライベートリポジトリーの量も表示されます。
- **Invoices**有料プランを使用している場合は、**View Invoices** をクリックして請求書のリストを表示できます。
- **Receipts**。有料プランを使用している場合は、支払いの領収書を自分または別のユーザーに電子メールで送信するか、領収書を完全にオプトアウトするかを選択できます。

第6章 タグの使用

イメージタグ は、コンテナーイメージの特定のバージョンまたはバリアントに割り当てられたラベルまたは識別子を指します。コンテナーイメージは通常、イメージのさまざまな部分を表す複数のレイヤーで構成されます。イメージタグは、異なるバージョンのイメージを区別したり、イメージに関する追加情報を提供したりするために使用します。

イメージタグには次の利点があります。

- **バージョン管理とリリース**: イメージタグを使用すると、アプリケーションまたはソフトウェアのさまざまなバージョンまたはリリースを示すことができます。たとえば、初期リリースを表すために **v1.0** とイメージにタグ付けしたり、更新されたバージョンを表すために **v1.1** とタグ付けしたりできます。これは、イメージのバージョンの明確な履歴を維持するのに役立ちます。

- **ロールバックとテスト**: 新しいイメージのバージョンで問題が発生した場合は、タグを指定することで以前のバージョンに簡単に戻すことができます。これは、デバッグ段階とテスト段階で特に役立ちます。

- **開発環境**: イメージタグは、さまざまな環境で作業する場合に役立ちます。たとえば、開発バージョンには **dev** タグ、品質保証テストには **qa**、本番環境には **prod** タグを使用して、それぞれ機能と設定を異なるものにすることができます。

- **継続的インテグレーション/継続的デプロイメント (CI/CD)**: CI/CD パイプラインでは、イメージタグを使用してデプロイメントプロセスを自動化することがよくあります。新しいコードの変更により、特定のタグを持つ新しいイメージの作成をトリガーすることで、シームレスな更新が可能になります。

- **機能ブランチ**: 複数の開発者が異なる機能やバグ修正に取り組んでいる場合は、変更ごとに個別のイメージタグを作成できます。これは、個々の機能を分離してテストするのに役立ちます。

- **カスタマイズ**: イメージタグを使用すると、各バリアントを追跡しながら、さまざまな設定、依存関係、または最適化を使用してイメージをカスタマイズできます。

- **セキュリティーとパッチ適用**: セキュリティーの脆弱性が発見された場合は、更新されたタグを使用して、パッチが適用されたバージョンのイメージを作成し、セキュアな最新バージョンをシステムで確実に使用できます。

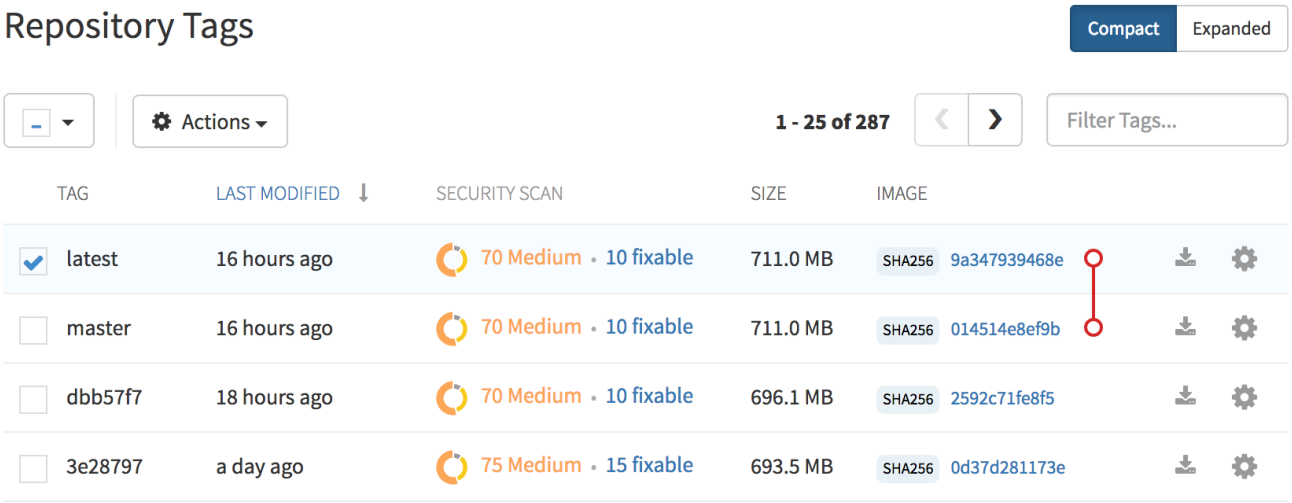
- **Dockerfile の変更**: Dockerfile またはビルドプロセスを変更する場合は、イメージタグを使用して、以前の Dockerfile と更新された Dockerfile からビルドしたイメージを区別できます。

全体として、イメージタグはコンテナーイメージを管理および整理するための構造化された方法を提供し、開発、デプロイ、および保守の効率的なワークフローを可能にします。

6.1. タグの表示と変更

Quay.io でイメージタグを表示するには、リポジトリーに移動し、**Tags** タブをクリックします。以下に例を示します。

リポジトリーからのタグの表示および変更



6.1.1. イメージへの新しいイメージタグの追加

Quay.io のイメージに新しいタグを追加できます。

手順

1. タグの横にある **Settings** または **歯車** アイコンをクリックし、**Add New Tag** をクリックします。
2. タグの名前を入力し、**Create Tag** をクリックします。
   新しいタグが **Repository Tags** ページにリストされます。

6.1.2. イメージタグの移動

必要に応じて、タグを別のイメージに移動できます。

手順

- タグの横にある **Settings** または **歯車** アイコンをクリックし、**Add New Tag** をクリックして既存のタグ名を入力します。Quay.io は、タグを追加するのではなく移動することを確認します。

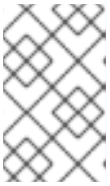
6.1.3. イメージタグの削除

イメージタグを削除すると、その特定のバージョンのイメージがレジストリーから実質的に削除されます。

イメージタグを削除するには、次の手順を実行します。

手順

1. リポジトリーの **Tags** ページに移動します。
2. **Delete Tag** をクリックします。これにより、タグとそのタグに固有のイメージが削除されます。



注記

イメージタグの削除は、**タイムマシン** 機能に割り当てられている時間に基づいて元に戻すことができます。詳細は、「タグの変更を元に戻す」を参照してください。

6.1.3.1. タグの履歴の表示

Quay.io では、イメージとそれぞれのイメージタグの包括的な履歴を確認できます。

手順

- リポジトリーの **Tag History** ページに移動して、イメージタグの履歴を表示します。

6.1.3.2. タグの変更を元に戻す

Quay.io は、古いイメージタグを一定期間リポジトリーに保持できる包括的な **タイムマシン** 機能を備えており、タグに加えられた変更を元に戻すことができます。この機能を使用すると、タグの削除などのタグの変更を元に戻すことができます。

手順

1. リポジトリーの **Tag History** ページに移動します。
2. タイムライン内でイメージタグが変更または削除された時点を見つけます。次に、**Revert** の下のオプションをクリックしてタグをイメージに復元するか、**Permanently Delete** の下のオプションをクリックしてイメージタグを完全に削除します。

6.1.4. タグやダイジェストによるイメージの取得

Quay.io では、Docker クライアントと Podman クライアントを使用して、複数の方法でイメージをプルできます。

手順

1. リポジトリーの **Tags** ページに移動します。
2. **Manifest** で、**Fetch Tag** アイコンをクリックします。
3. ポップアップボックスが表示され、次のオプションが表示されます。
   - Podman Pull (by tag)
   - Docker Pull (by tag)
   - Podman Pull (by digest)
   - Docker Pull (by digest)
     4 つのオプションのいずれかを選択すると、イメージのプルに使用できる各クライアント用のコマンドが返されます。
4. **Copy Command** をクリックしてコマンドをコピーします。このコマンドはコマンドラインインターフェイス (CLI) で使用できます。以下に例を示します。

```
$ podman pull quay.io/quayadmin/busybox:test2
```

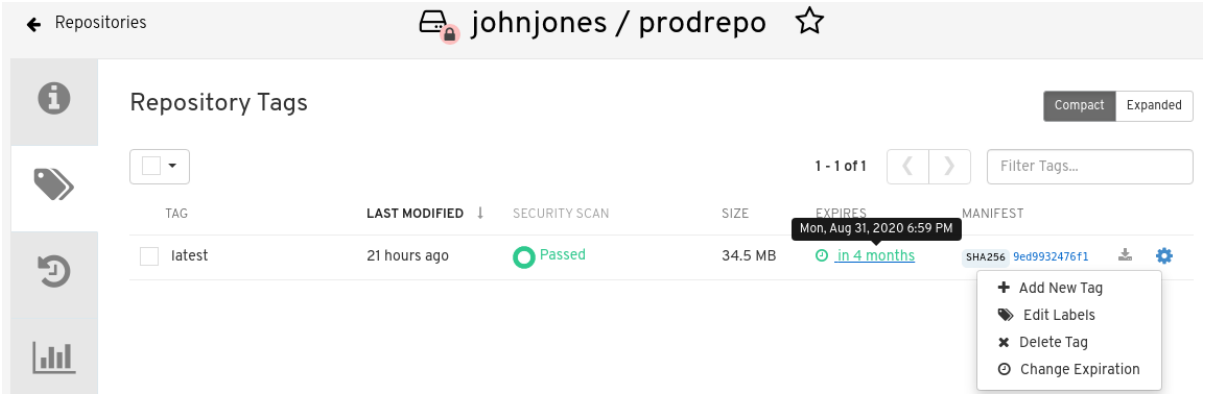
6.2. タグの有効期限

タグの有効期限 機能を使用すると、選択した日時に Quay.io リポジトリーのイメージが期限切れになるように設定できます。この機能には次の特徴があります。

- イメージタグの有効期限が切れると、そのタグがリポジトリーから削除されます。特定のイメージに対する最後のタグであれば、そのイメージも削除するように設定されます。
- 有効期限はタグごとに設定されます。リポジトリー全体に対して設定されるものではありません。
- タグは期限切れになったり、削除されたりしても、レジストリーからすぐには削除されません。これは、**タイムマシン** 機能で設計された割り当て時間によって決まります。この時間により、タグを完全に削除するタイミング、またはガベージコレクションを行うタイミングが定義されます。デフォルトでは、この値は **14 日** に設定されていますが、管理者は複数のオプションから 1 つ選択することで、この時間を調整できます。ガベージコレクションが発生するまでは、タグの変更を元に戻すことができます。

タグの有効期限は、次の 2 つの方法のいずれかで設定できます。

- イメージの作成時に Dockerfile で **quay.expires-after=** LABEL を設定する方法。これは、イメージをビルドした時点からの有効期間を設定するものです。
- Quay.io UI で有効期限を選択する方法。以下に例を示します。



6.2.1. Dockerfile からのタグの有効期限の設定

docker label コマンドを使用してラベル (例: **quay.expires-after=20h**) を追加すると、指定した時間の経過後に、タグが自動的に期限切れになります。以下に示す時間、日、または週の値を指定できます。

- **1h**
- **2d**
- **3w**

有効期限は、イメージがレジストリーにプッシュされた時点から始まります。

6.2.2. リポジトリーからのタグの有効期限の設定

タグの有効期限は Quay.io UI で設定できます。

手順

1. リポジトリーに移動し、ナビゲーションペインで **Tags** をクリックします。

2. イメージタグの **Settings** または **gear** アイコンをクリックし、**Change Expiration** を選択します。

3. プロンプトが表示されたら日付と時刻を選択し、**Change Expiration** を選択します。有効期限に達するとタグがリポジトリーから削除されるように設定されます。

6.3. CLAIR セキュリティースキャンの表示

Quay.io には Clair セキュリティースキャナーが装備されています。Quay.io の Clair の詳細は、「Clair セキュリティースキャナー」を参照してください。

手順

1. リポジトリーに移動し、ナビゲーションペインで **Tags** をクリックします。このページに、セキュリティースキャンの結果が表示されます。

2. マルチアーキテクチャーイメージに関する詳細情報を表示するには、**See Child Manifests** をクリックして、展開されたビューでマニフェストのリストを確認します。

3. **See Child Manifests** の下の関連リンクをクリックします。たとえば、**1 Unknown** をクリックすると、**Security Scanner** ページにリダイレクトされます。

4. **Security Scanner** ページには、イメージがどの CVE の影響を受けるか、利用可能な修復オプションなど、タグに関する情報が表示されます。

注記

イメージスキャンでは、Clair セキュリティースキャナーによって検出された脆弱性がリストされるだけです。発見された脆弱性への対処は、当該ユーザーに委ねられています。

第7章 ログの表示とエクスポート

アクティビティーログは、Quay.io のすべてのリポジトリーと namespace に対して収集されます。

Quay.io の使用状況ログを表示すると、運用とセキュリティーの両方の目的で貴重な洞察と利点を得ることができます。使用状況ログから次の情報が明らかになる可能性があります。

- **リソースプランニング**: 使用状況ログは、イメージのプル、プッシュの数、レジストリーへの全体的なトラフィックに関するデータを提供します。
- **ユーザーアクティビティー**: ログを使用すると、ユーザーアクティビティーを追跡し、どのユーザーがレジストリー内のイメージにアクセスして操作しているかを把握できます。これは、監査、ユーザー行動の理解、アクセス制御の管理に役立ちます。
- **使用パターン**: 使用パターンを調査することで、よく使用されるイメージ、頻繁に使用されるバージョン、ほとんどアクセスされないイメージについて詳細な情報を得ることができます。この情報は、イメージのメンテナンスとクリーンアップの作業に優先順位を付けるのに役立ちます。
- **セキュリティー監査**: 使用状況ログにより、誰がいつイメージにアクセスしたかを追跡できます。これは、セキュリティー監査、コンプライアンス、および不正または不審なアクティビティーの調査にとって非常に重要です。
- **イメージのライフサイクル管理**: ログにより、どのイメージがプル、プッシュ、削除されているかが明らかになります。この情報は、古いイメージを廃止し、許可されたイメージだけを確実に使用させるなど、イメージのライフサイクル管理に不可欠です。
- **コンプライアンスと規制要件**: 多くの業界には、機密リソースへのアクセスの追跡と監査を義務付けるコンプライアンス要件があります。使用状況ログは、そのような規制への準拠を証明するのに役立ちます。
- **異常な動作の特定**: 使用状況ログ内の正常でないパターンや異常なパターンは、潜在的なセキュリティー違反または悪意のあるアクティビティーを示している可能性があります。このログを監視すると、セキュリティーインシデントをより効果的に検出して対応することができます。
- **傾向分析**: 使用状況ログは、レジストリーがどのように使用されているかに関する経時的な傾向と詳細情報を提供します。これは、リソースの割り当て、アクセス制御、イメージ管理戦略について、情報に基づいた意思決定を行うのに役立ちます。

ログファイルにアクセスする方法は、以下のように複数あります。

- Web UI によりログを閲覧する
- 外部に保存できるようにログをエクスポートする
- API を使用してログエントリーへのアクセスする

ログにアクセスするには、選択したリポジトリーまたは名前空間の管理者権限が必要です。

注記

API を介して一度に利用できるログ結果は最大 100 件です。それ以上の結果を集めるには、この章で紹介するログエクスポーター機能を使う必要があります。

7.1. UI を使用したログの表示

Web UI を使用してリポジトリーまたは名前空間のログエントリーを表示するには、次の手順を実行します。

手順

1. 自分が管理者であるリポジトリーまたは名前空間に移動します。
2. ナビゲーションペインで、**Usage Logs** を選択します。

3. オプション: Usage Logs ページで以下を行います。

   a. **From** ボックスと **to** ボックスに日付を追加して、ログエントリーを表示する日付範囲を設定します。デフォルトでは、UI には最新週のログエントリーが表示されます。

   b. **Filter Logs** ボックスに文字列を入力すると、指定したキーワードのログエントリーが表示されます。たとえば、**delete** と入力してログをフィルタリングし、削除されたタグを表示できます。

   c. **Description** で、ログエントリーの矢印を切り替えると、特定のログエントリーに関連するテキストが表示されます。

7.2. リポジトリーログのエクスポート

ログのエクスポート 機能を使用すると、より多くのログファイルを取得し、Quay.io の外部に保存できます。この機能には次の利点と制約があります。

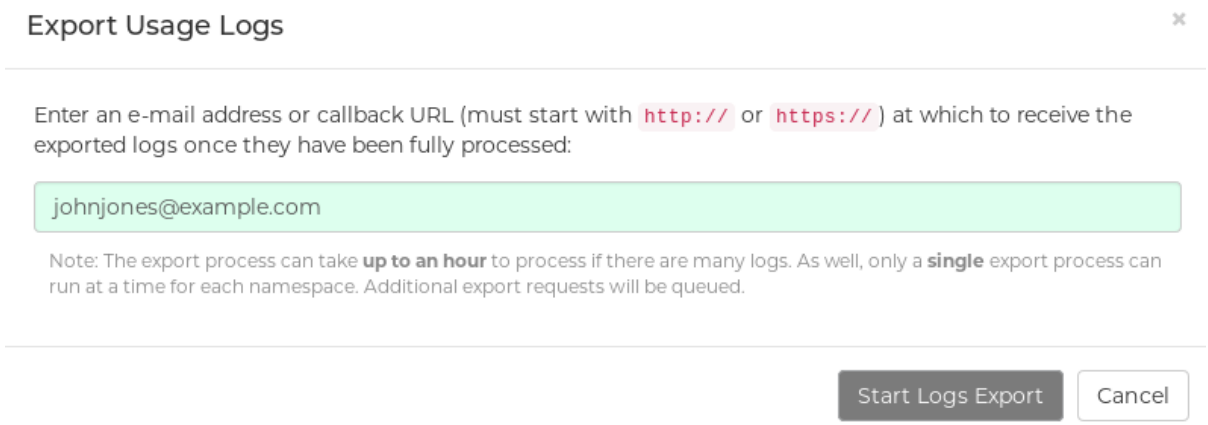
- リポジトリーから収集するログの日付の範囲を選択できます。
- ログをメールの添付ファイルで送信するか、コールバック URL に移動することを要求できます。
- ログをエクスポートするには、リポジトリーまたは名前空間の管理者である必要があります。
- すべてのユーザーのログが 30 日分保持されます。

- Export Logs 機能は、過去に生成されたログデータのみを収集します。ログ取得中のデータのストリーミングは行いません。
- ログが収集されて利用可能になったら、そのデータを保存する場合は、すぐにコピーする必要があります。デフォルトでは、データは 1 時間後に期限切れになります。

ログをエクスポートするには、次の手順を実行します。

手順

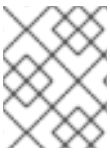
1. 管理者権限のあるリポジトリーを選択します。
2. ナビゲーションペインで、**Usage Logs** を選択します。
3. オプション: 特定の日付を指定する場合は、**From** ボックスと **to** ボックスに範囲を入力します。
4. **Export Logs** ボタンをクリックします。以下のような Export Usage Logs のポップアップが表示されます。



5. エクスポートされたログを受信するメールアドレスまたはコールバック URL を入力します。コールバック URL には、指定ドメインへの URL (例: <webhook.site>) を使用できます。
6. **Start Logs Export** を選択して、選択したログエントリーを収集するプロセスを開始します。収集されるログデータの量に応じて、これが完了するまでに数分から数時間かかる場合があります。
7. ログのエクスポートが完了すると、次の 2 つのイベントのいずれかが発生します。
   - 要求したエクスポート済みログエントリーが利用可能であることを通知するメールが受信されます。
   - webhook URL からのログエクスポート要求の成功ステータスが返されます。さらに、エクスポートされたデータへのリンクを使用して、ログをダウンロードできるようになります。

第8章 コンテナーイメージのビルド

コンテナーイメージをビルドするには、コンテナー化されたアプリケーションのブループリントを作成する必要があります。ブループリントは、アプリケーションのインストール方法と設定方法を定義する他のパブリックリポジトリーのベースイメージに依存します。

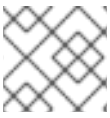


注記

ブループリントは他のパブリックリポジトリーのイメージに依存しているため、レート制限の対象となる可能性があります。その結果、ビルドが失敗する **可能性があります**。

Quay.io は、Docker および Podman コンテナーイメージを構築する機能をサポートしています。この機能は、コンテナーとコンテナーオーケストレーションを利用する開発者や組織に役立ちます。

Quay.io では、この機能は無料と有料の両方の階層プランで同じように機能します。



注記

Quay.io は、1 人のユーザーが一度に送信できる同時ビルドの数を制限します。

8.1. ビルドコンテキスト

Docker または Podman でイメージをビルドする際には、**ビルドコンテキスト** となるディレクトリーを指定します。これは手動ビルドとビルドトリガーの両方に当てはまります。Quay.io によって作成されるビルドは、ローカルマシン上で **docker build** または **podman build** を実行することと変わらないためです。

Quay.io のビルドコンテキストは、常にビルドセットアップから指定された **サブディレクトリー** であり、ディレクトリーが指定されていない場合はビルドソースのルートにフォールバックします。

ビルドがトリガーされると、Quay.io のビルドワーカーは Git リポジトリーをワーカーマシンにクローンし、ビルドを行う前にビルドコンテキストに入ります。

.tar アーカイブをベースにしたビルドでは、ビルドワーカーがアーカイブを抽出し、ビルドコンテキストに入ります。以下に例を示します。

展開されたビルドアーカイブ

```
example
├── .git
├── Dockerfile
├── file
└── subdir
    └── Dockerfile
```

上記の **展開されたビルドアーカイブ** は、**example** という Github リポジトリーのディレクトリー構造を持っていると考えてみてください。ビルドトリガーの設定でサブディレクトリーが指定されていない場合、またはビルドを手動で開始する場合、ビルドは example ディレクトリーで行われます。

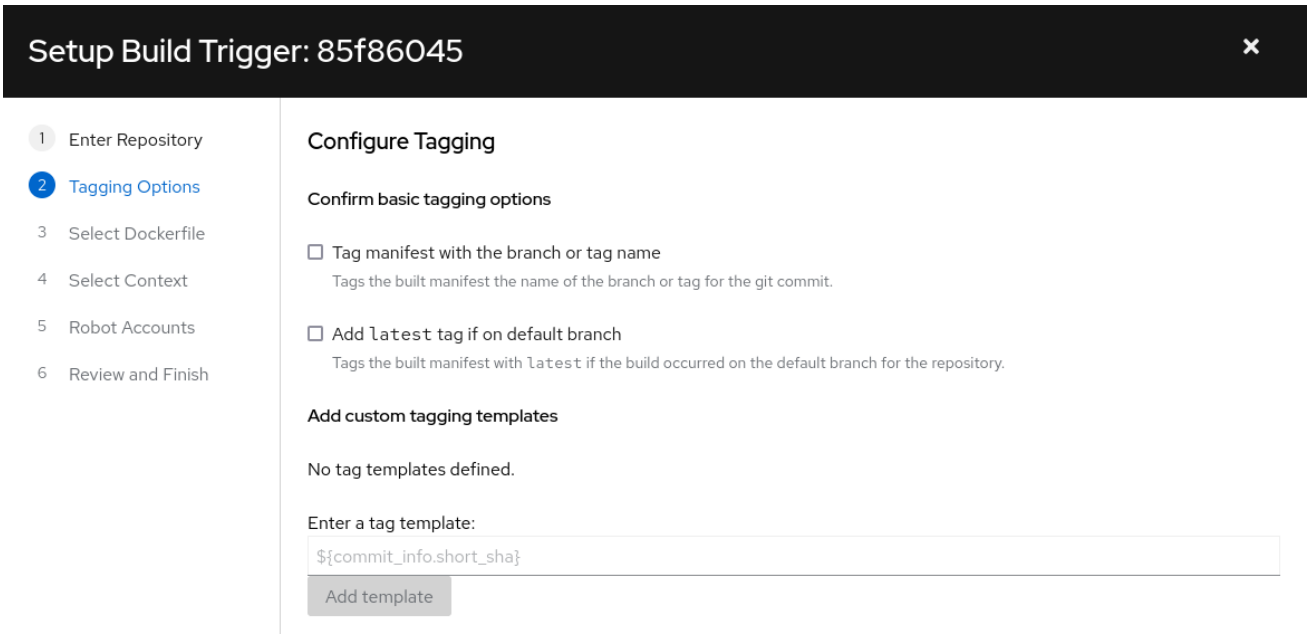
ビルドトリガーの設定でサブディレクトリー (**subdir** など) を指定した場合は、その中の Dockerfile のみがビルドの対象になります。つまり、Dockerfile の **ADD** コマンドを使用して **file** を追加することは、ビルドコンテキストの外にあるためできません。

Docker Hub とは異なり、Dockerfile は Quay.io のビルドコンテキストの一部です。そのため、Dockerfile を **.dockerignore** ファイル内に含めることはできません。

8.2. ビルドトリガーのタグ命名

カスタムタグは Quay.io で使用できます。

1 つの方法として、ビルドした各イメージにタグとして割り当てる文字列を含める方法があります。または、ビルドトリガーの **Configure Tagging** セクションで次のタグテンプレートを使用して、各コミットからの情報でイメージにタグ付けすることもできます。



- **${commit}**: 発行されたコミットの完全な SHA
- **${parsed_ref.branch}**: ブランチ情報 (利用可能な場合)
- **${parsed_ref.tag}**: タグ情報 (利用可能な場合)
- **${parsed_ref.remote}**: リモート名
- **${commit_info.date}**: コミットが発行された日付
- **${commit_info.author.username}**: コミットの作成者のユーザー名
- **${commit_info.short_sha}**: コミット SHA の最初の 7 文字
- **${committer.properties.username}**: コミッターのユーザー名

以上がすべてではありませんが、これらはタグ付けに最も役立つタグテンプレートです。完全なタグテンプレートスキーマは、こちらのページ を参照してください。

詳細は、Set up custom tag templates in build triggers for Red Hat Quay and Quay.io を参照してください。

8.3. ソースコントロールをトリガーとしたビルドのスキップ

Quay.io ビルドシステムがコミットを無視するように指定するには、コミットメッセージの任意の場所に **[skip build]** または **[build skip]** というテキストを追加します。

8.4. ビルドの表示および管理

リポジトリービルドは、Quay.io UI で表示および管理できます。

手順

1. Quay.io に移動し、リポジトリーを選択します。
2. ナビゲーションペインで、**Builds** を選択します。

8.5. 新しいビルドの作成

デフォルトでは、Quay.io ユーザーはすぐに新しいビルドを作成できます。

前提条件

- リポジトリーの **Builds** ページに移動している。

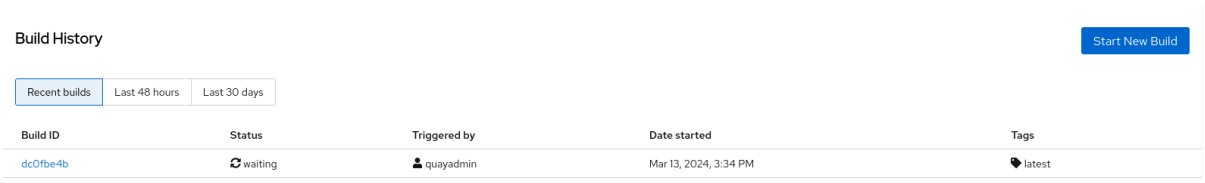
手順

1. **Builds** ページで、**Start New Build** をクリックします。
2. プロンプトが表示されたら、**Upload Dockerfile** をクリックして、ルートディレクトリーに Dockerfile または Dockerfile を含むアーカイブをアップロードします。
3. **Start Build** をクリックします。

注記

- 現在、ユーザーは手動でビルドを開始するときに Docker ビルドコンテキストを指定できません。
- 現在、BitBucket は Red Hat Quay v2 UI ではサポートされていません。

4. ビルドにリダイレクトされます。ビルドはリアルタイムで確認できます。Dockerfile ビルドが完了してプッシュされるまで待ちます。
5. オプション: **Download Logs** をクリックしてログをダウンロードしたり、**Copy Logs** をクリックしてログをコピーしたりできます。
6. 戻るボタンをクリックして **Repository Builds** ページに戻り、ビルド履歴を表示できます。



8.6. ビルドトリガー

ビルドトリガーは、ソースコントロールのプッシュ、webhook 呼び出しの作成 など、トリガー条件が満たされるとビルドを呼び出します。

8.6.1. ビルドトリガーの作成

次の手順に従って、カスタム Git リポジトリーを使用してビルドトリガーを作成します。

注記

以下の手順は、**config.yaml** ファイルに Github 認証情報が含まれていないことを前提としています。

前提条件

- リポジトリーの **Builds** ページに移動している。

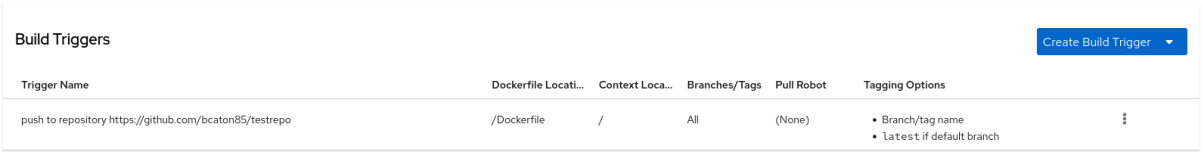
手順

1. **Builds** ページで、**Create Build Trigger**をクリックします。
2. Github、BitBucket、Gitlab などの目的のプラットフォームを選択するか、カスタム Git リポジトリーを使用します。この例では、Github のカスタム Git リポジトリーを使用しています。
3. カスタム Git リポジトリー名を入力します (例: **git@github.com:<username>/<repo>.git**)。**Next** をクリックします。
4. プロンプトが表示されたら、次のオプションのいずれかまたは両方を選択して、タグ付けオプションを設定します。
   - **Tag manifest with the branch or tag name**。このオプションを選択すると、ビルドされたマニフェストにブランチの名前または git コミットのタグがタグ付けされます。
   - **Add latest tag if on default branch**。このオプションを選択すると、リポジトリーのデフォルトブランチでビルドが行われた場合、ビルドされたマニフェストに latest がタグ付けされます。
     オプションで、カスタムのタグ付けテンプレートを追加できます。ここに入力できるタグテンプレートは複数あります。短い SHA ID、タイムスタンプ、作成者名、コミッター、コミットからのブランチ名をタグとして使用することもできます。詳細は、「ビルドトリガーのタグ命名」を参照してください。

     タグ付けを設定したら、**Next** をクリックします。
5. プロンプトが表示されたら、トリガーの呼び出し時にビルドする Dockerfile の場所を選択します。Dockerfile が git リポジトリーのルートにあり、Dockerfile という名前が付けられている場合は、Dockerfile パスとして **/Dockerfile** を入力します。**Next** をクリックします。
6. プロンプトが表示されたら、Docker ビルドのコンテキストを選択します。Dockerfile が Git リポジトリーのルートにある場合は、ビルドコンテキストディレクトリーとして / を入力します。**Next** をクリックします。
7. オプション: 任意のロボットアカウントを選択します。これにより、ビルドプロセス中にプライベートのベースイメージをプルできます。プライベートベースイメージが使用されていないことを把握している場合は、この手順を省略できます。
8. **Next** をクリックします。検証の警告がないか確認します。必要に応じて、**Finish** をクリックする前に問題を修正します。
9. トリガーが正常にアクティベートされたという警告が表示されます。このトリガーを使用するには、以下のアクションが必要になることに注意してください。
   - 以下の公開鍵に git リポジトリーへの読み取りアクセス権を与える必要があります。

- ビルドをトリガーするには、リポジトリーを **POST** に設定する必要があります。
  SSH 公開鍵を保存し、**Return to <organization_name>/<repository_name>** をクリックします。リポジトリーの **Builds** ページにリダイレクトされます。

10. **Builds** ページに、ビルドトリガーが表示されます。以下に例を示します。



8.6.2. ビルドの手動トリガー

次の手順を使用して、ビルドを手動でトリガーできます。

手順

1. **Builds** ページで、**Start new build** をクリックします。
2. プロンプトが表示されたら、**Invoke Build Trigger** を選択します。
3. **Run Trigger Now** をクリックして、プロセスを手動で開始します。
   ビルドが開始したら、**Repository Builds** ページでビルド ID を確認できます。

8.7. カスタム GIT トリガーの設定

カスタム Git トリガー は、Git サーバーをビルドトリガーとして機能させるための一般的な方法です。カスタム Git トリガーは SSH 鍵と webhook エンドポイントのみに依存します。それ以外の実装はユーザーに委ねられています。

8.7.1. トリガーの作成

カスタム Git トリガーの作成は、他のトリガーの作成と似ていますが、次の点が違います。

- Quay.io は、トリガーで使用する適切なロボットアカウントを自動的に検出することはできません。これは、作成時に手動で行う必要があります。
- トリガーの作成後に追加の手順を実施する必要があります。この手順については、次のセクションで詳しく説明します。

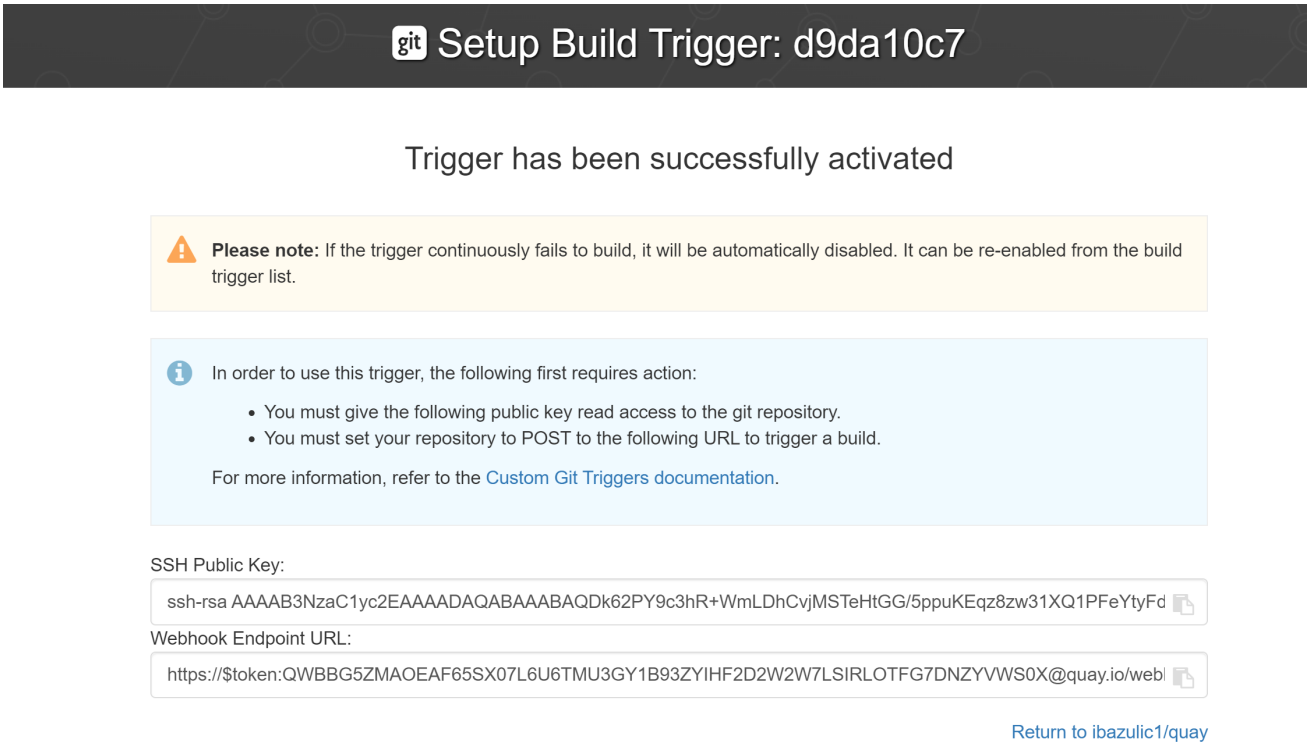
8.7.2. カスタムトリガー作成の設定

カスタム Git トリガーを作成する場合は、次の 2 つの追加手順が必要です。

1. トリガーの作成時に生成された SSH 公開鍵への読み取りアクセスを付与する必要があります。
2. ビルドをトリガーするには、Quay.io エンドポイントに POST する Webhook をセットアップする必要があります。

鍵と URL は、**Settings** または **歯車** アイコンから **View Credentials** を選択することで利用できます。

リポジトリーからのタグの表示および変更



8.7.2.1. SSH 公開鍵へのアクセス

Git サーバーの設定に応じて、Quay.io がカスタム Git トリガー用に生成する SSH 公開鍵をインストールする方法はさまざまです。

たとえば、Git のドキュメント では、小規模なサーバーのセットアップを説明しています。この場合は、鍵を **$HOME/.ssh/authorize_keys** に追加すると、ビルダーがリポジトリーをクローンするためのアクセス権が付与されます。公式にサポートされていない git リポジトリー管理ソフトウェアの場合は、通常、**Deploy Keys** というラベルが付いた、鍵を入力する場所があります。

8.7.2.2. Webhook

ビルドを自動的にトリガーするには、次の形式を使用して **.json** ペイロードを webhook URL に **POST** する必要があります。

これはサーバーの設定に応じてさまざまな方法で行うことができますが、ほとんどの場合は **post-receive** Git フック で行うことができます。

注記

このリクエストが有効であるためには、**application/json** を含む **Content-Type** ヘッダーが必要です。

webhook の例

```
{
  "commit": "1c002dd",                                  // required
  "ref": "refs/heads/master",                           // required
  "default_branch": "master",                           // required
  "commit_info": {                                      // optional
    "url": "gitsoftware.com/repository/commits/1234567", // required
    "message": "initial commit",                        // required
```

```
    "date": "timestamp",                        // required
    "author": {                                 // optional
      "username": "user",                       // required
      "avatar_url": "gravatar.com/user.png",    // required
      "url": "gitsoftware.com/users/user"       // required
    },
    "committer": {                              // optional
      "username": "user",                       // required
      "avatar_url": "gravatar.com/user.png",    // required
      "url": "gitsoftware.com/users/user"       // required
    }
  }
}
```

第9章 リポジトリー通知

Quay.io は、リポジトリーのライフサイクルで発生するさまざまなイベントに対して、リポジトリーへの **通知** の追加をサポートしています。

9.1. 通知の作成

通知を追加するには、次の手順を実行します。

前提条件

- リポジトリーを作成している。
- リポジトリーの管理者権限がある。

手順

1. Quay.io のリポジトリーに移動します。
2. ナビゲーションペインで、**Settings** をクリックします。
3. **Events and Notifications** カテゴリーで、**Create Notification** をクリックして、リポジトリーイベントの新しい通知を追加します。**Create repository notification** ページにリダイレクトされます。
4. **Create repository notification** ページで、ドロップダウンメニューを選択してイベントのリストを表示します。次のタイプのイベントの通知を選択できます。
   - リポジトリーへのプッシュ
   - Dockerfile ビルドのキューへの追加
   - Dockerfile ビルドの開始
   - Dockerfile ビルドの正常な完了
   - Docker ビルドのキャンセル
   - パッケージの脆弱性の検出
5. イベントの種類を選択したら、通知方法を選択します。次の方法を使用できます。
   - Quay 通知
   - メール
   - Webhook POST
   - Flowdock チーム通知
   - HipChat ルーム通知
   - Slack ルーム通知

   選択した方法に応じて、追加情報を入力する必要があります。たとえば、**E-mail** を選択した場合は、メールアドレスと通知タイトル (省略可能) を入力する必要があります。

6. イベントと通知方法を選択したら、**Create Notification** をクリックします。

9.2. リポジトリーイベントの説明

次のセクションでは、リポジトリーイベントについて詳しく説明します。

9.2.1. リポジトリープッシュ

1 つまたは複数のイメージのリポジトリーへのプッシュが成功しました。

```
{
  "name": "repository",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "homepage": "https://quay.io/repository/dgangaia/repository",
  "updated_tags": [
    "latest"
  ]
}
```

9.2.2. Dockerfile ビルドのキューへの追加

次の例は、ビルドシステムのキューに追加された Dockerfile ビルドからの応答です。

注記

応答は、オプションの属性の使用によって異なる場合があります。

```
{
  "build_id": "296ec063-5f86-4706-a469-f0a400bf9df2",
  "trigger_kind": "github",                                    //Optional
  "name": "test",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",                  //Optional
  "docker_tags": [
    "master",
    "latest"
  ],
  "repo": "test",
  "trigger_metadata": {
    "default_branch": "master",
    "commit": "b7f7d2b948aacbe844ee465122a85a9368b2b735",
    "ref": "refs/heads/master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {                                           //Optional
      "url": "https://github.com/dgangaia/test/commit/b7f7d2b948aacbe844ee465122a85a9368b2b735",
      "date": "2019-03-06T12:48:24+11:00",
      "message": "adding 5",
      "author": {                                              //Optional
        "username": "dgangaia",
```

```
      "url": "https://github.com/dgangaia",                                    //Optional
      "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"    //Optional
     },
     "committer": {
      "username": "web-flow",
      "url": "https://github.com/web-flow",
      "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"
     }
    }
  },
  "is_manual": false,
  "manual_user": null,
  "homepage": "https://quay.io/repository/dgangaia/test/build/296ec063-5f86-4706-a469-
f0a400bf9df2"
}
```

9.2.3. Dockerfile ビルドの開始

次の例は、ビルドシステムのキューに追加された Dockerfile ビルドからの応答です。

注記

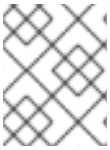
応答は、オプションの属性の使用によって異なる場合があります。

```
{
  "build_id": "a8cc247a-a662-4fee-8dcb-7d7e822b71ba",
  "trigger_kind": "github",                                                    //Optional
  "name": "test",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",                        //Optional
  "docker_tags": [
    "master",
    "latest"
  ],
  "build_name": "50bc599",
  "trigger_metadata": {                                                        //Optional
    "commit": "50bc5996d4587fd4b2d8edc4af652d4cec293c42",
    "ref": "refs/heads/master",
    "default_branch": "master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {                                                           //Optional
      "url": "https://github.com/dgangaia/test/commit/50bc5996d4587fd4b2d8edc4af652d4cec293c42",
      "date": "2019-03-06T14:10:14+11:00",
      "message": "test build",
      "committer": {                                                           //Optional
        "username": "web-flow",
        "url": "https://github.com/web-flow",                                  //Optional
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"  //Optional
      },
      "author": {                                                              //Optional
        "username": "dgangaia",
        "url": "https://github.com/dgangaia",                                  //Optional
```

```
        "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"   //Optional
      }
    }
  },
  "homepage": "https://quay.io/repository/dgangaia/test/build/a8cc247a-a662-4fee-8dcb-
7d7e822b71ba"
}
```

9.2.4. Dockerfile ビルドの正常な完了

次の例は、ビルドシステムによって正常に完了した Dockerfile ビルドからの応答です。



注記

このイベントは、ビルドされたイメージの **リポジトリープッシュ** イベントと同時に発生します。

```
{
  "build_id": "296ec063-5f86-4706-a469-f0a400bf9df2",
  "trigger_kind": "github",                                          //Optional
  "name": "test",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",                    //Optional
  "docker_tags": [
    "master",
    "latest"
  ],
  "build_name": "b7f7d2b",
  "image_id": "sha256:0339f178f26ae24930e9ad32751d6839015109eabdf1c25b3b0f2abf8934f6cb",
  "trigger_metadata": {
    "commit": "b7f7d2b948aacbe844ee465122a85a9368b2b735",
    "ref": "refs/heads/master",
    "default_branch": "master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {                                                 //Optional
      "url": "https://github.com/dgangaia/test/commit/b7f7d2b948aacbe844ee465122a85a9368b2b735",
      "date": "2019-03-06T12:48:24+11:00",
      "message": "adding 5",
      "committer": {                                                 //Optional
        "username": "web-flow",
        "url": "https://github.com/web-flow",                              //Optional
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"
//Optional
      },
      "author": {                                                    //Optional
        "username": "dgangaia",
        "url": "https://github.com/dgangaia",                              //Optional
        "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"     //Optional
      }
    }
  },
  "homepage": "https://quay.io/repository/dgangaia/test/build/296ec063-5f86-4706-a469-
```

```
f0a400bf9df2",
  "manifest_digests": [

"quay.io/dgangaia/test@sha256:2a7af5265344cc3704d5d47c4604b1efcbd227a7a6a6ff73d6e4e08a27fd7d99",

"quay.io/dgangaia/test@sha256:569e7db1a867069835e8e97d50c96eccafde65f08ea3e0d5debaf16e2545d9d1"
  ]
}
```

9.2.5. Dockerfile ビルドの失敗

次の例は、失敗した Dockerfile ビルドからの応答です。

```
{
  "build_id": "5346a21d-3434-4764-85be-5be1296f293c",
  "trigger_kind": "github",                                          //Optional
  "name": "test",
  "repository": "dgangaia/test",
  "docker_url": "quay.io/dgangaia/test",
  "error_message": "Could not find or parse Dockerfile: unknown instruction: GIT",
  "namespace": "dgangaia",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",                  //Optional
  "docker_tags": [
    "master",
    "latest"
  ],
  "build_name": "6ae9a86",
  "trigger_metadata": {                                              //Optional
    "commit": "6ae9a86930fc73dd07b02e4c5bf63ee60be180ad",
    "ref": "refs/heads/master",
    "default_branch": "master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {                                                 //Optional
      "url": "https://github.com/dgangaia/test/commit/6ae9a86930fc73dd07b02e4c5bf63ee60be180ad",
      "date": "2019-03-06T14:18:16+11:00",
      "message": "failed build test",
      "committer": {                                               //Optional
        "username": "web-flow",
        "url": "https://github.com/web-flow",                          //Optional
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"     //Optional
      },
      "author": {                                                  //Optional
        "username": "dgangaia",
        "url": "https://github.com/dgangaia",                          //Optional
        "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"     //Optional
      }
    }
  },
  "homepage": "https://quay.io/repository/dgangaia/test/build/5346a21d-3434-4764-85be-5be1296f293c"
}
```

9.2.6. Dockerfile ビルドのキャンセル

次の例は、キャンセルされた Dockerfile ビルドからの応答です。


```
{
  "build_id": "cbd534c5-f1c0-4816-b4e3-55446b851e70",
  "trigger_kind": "github",
  "name": "test",
  "repository": "dgangaia/test",
  "namespace": "dgangaia",
  "docker_url": "quay.io/dgangaia/test",
  "trigger_id": "38b6e180-9521-4ff7-9844-acf371340b9e",
  "docker_tags": [
    "master",
    "latest"
  ],
  "build_name": "cbce83c",
  "trigger_metadata": {
    "commit": "cbce83c04bfb59734fc42a83aab738704ba7ec41",
    "ref": "refs/heads/master",
    "default_branch": "master",
    "git_url": "git@github.com:dgangaia/test.git",
    "commit_info": {
      "url": "https://github.com/dgangaia/test/commit/cbce83c04bfb59734fc42a83aab738704ba7ec41",
      "date": "2019-03-06T14:27:53+11:00",
      "message": "testing cancel build",
      "committer": {
        "username": "web-flow",
        "url": "https://github.com/web-flow",
        "avatar_url": "https://avatars3.githubusercontent.com/u/19864447?v=4"
      },
      "author": {
        "username": "dgangaia",
        "url": "https://github.com/dgangaia",
        "avatar_url": "https://avatars1.githubusercontent.com/u/43594254?v=4"
      }
    }
  },
  "homepage": "https://quay.io/repository/dgangaia/test/build/cbd534c5-f1c0-4816-b4e3-
55446b851e70"
}
```


9.2.7. 脆弱性の検出

次の例は、リポジトリーで脆弱性を検出した Dockerfile ビルドからの応答です。


```
{
  "repository": "dgangaia/repository",
  "namespace": "dgangaia",
  "name": "repository",
  "docker_url": "quay.io/dgangaia/repository",
  "homepage": "https://quay.io/repository/dgangaia/repository",

  "tags": ["latest", "othertag"],
```

```
  "vulnerability": {
    "id": "CVE-1234-5678",
    "description": "This is a bad vulnerability",
    "link": "http://url/to/vuln/info",
    "priority": "Critical",
    "has_fix": true
  }
}
```


9.3. 通知アクション

9.3.1. 通知の追加

通知は **Events and Notifications** ページの **Repository Settings** セクションに追加されます。通知は **Notifications** ウィンドウにも追加されます。このウィンドウは Quay.io のナビゲーションペインにある **ベル** アイコンをクリックすると表示されます。

Quay.io の通知は、**ユーザー**、**チーム**、または**組織** 全体に送信するように設定できます。

9.3.2. メール通知

指定のイベントを説明するメールが、指定のアドレスに送信されます。メールアドレスは **リポジトリーごとに** 認証する必要があります。

9.3.3. Webhook POST 通知

HTTP **POST** 呼び出しは、イベントのデータを使用して、指定の URL に対して行われます。イベントデータの詳細は、「リポジトリーイベントの説明」を参照してください。

URL が HTTPS の場合、呼び出しには Quay.io の SSL クライアント証明書が設定されます。この証明書を検証することで、Quay.io からの呼び出しが証明されます。ステータスコードが **2xx** の範囲の応答は成功とみなされます。それ以外のステータスコードを持つ応答は失敗とみなされ、webhook 通知が再試行されます。

9.3.4. Flowdock 通知

Flowdock にメッセージを投稿します。

9.3.5. Hipchat 通知

HipChat にメッセージを投稿します。

9.3.6. Slack 通知

Slack にメッセージを投稿します。

第10章 OPEN CONTAINER INITIATIVE のサポート

コンテナーレジストリーは、当初 Docker イメージ形式のコンテナーイメージをサポートするように設計されていました。Docker 以外で追加のランタイムの使用をプロモートするために、コンテナーランタイムとイメージ形式に関連する標準化を提供するために Open Container Initiative (OCI) が作成されました。ほとんどのコンテナーレジストリーは、Docker イメージマニフェスト V2、Schema 2 形式をベースとして OCI 標準化をサポートします。

コンテナーイメージのほかにも、個別のアプリケーションだけでなく、Kubernetes プラットフォームを全体としてサポートする各種のアーティファクトが新たに出現しました。これらは、アプリケーションのデプロイメントを支援するセキュリティーおよびガバナンスの Open Policy Agent (OPA) ポリシーから Helm チャートおよび Operator に及びます。

Quay.io は、コンテナーイメージを格納するだけでなく、コンテナーの管理を支援するツールのエコシステム全体もサポートするプライベートコンテナーレジストリーです。Quay.io は、OCI 1.0 **Image and Distribution specifications** と可能な限り互換性を保つよう努めており、**Helm チャート** (OCI をサポートする Helm のバージョンを使用してプッシュされている場合に限る) などの一般的なメディアタイプや、コンテナーイメージのマニフェストまたはレイヤーコンポーネント内部のさまざまな任意のメディアタイプをサポートしています。このような新しいメディアタイプのサポートが、以前の Quay.io のバージョンと異なる点です。以前のバージョンでは、受け入れるメディアタイプについてより厳しい制限がレジストリーに設けられていました。Quay.io は、以前はサポート範囲外だったメディアタイプも含め、より幅広いメディアタイプに対応できるようになりました。そのため、標準のコンテナーイメージ形式だけでなく、新しいタイプや従来とは異なるタイプにも対応できるようになり、より多用途になりました。

新しいメディアタイプへのサポート拡張に加えて、Quay.io は、V2_2 および V2_1 形式を含む Docker イメージとの互換性を確保しています。このような Docker V2_2 および V2_1 イメージとの互換性は、Docker ユーザーにシームレスなエクスペリエンスを提供するという Quay.io の取り組みを表すものです。さらに、Quay.io は、引き続き Docker V1 プルのサポートを拡張し、このような以前のバージョンの Docker イメージを現在も利用している可能性のあるユーザーに対応します。

OCI アーティファクトのサポートはデフォルトで有効になっています。

10.1. HELM および OCI の前提条件

Helm は、アプリケーションのパッケージ化とデプロイの方法を簡素化します。Helm は、アプリケーションを表す Kubernetes リソースが含まれる **チャート** というパッケージ形式を使用します。Quay.io は、OCI でサポートされているバージョンである限り、Helm チャートをサポートします。

次の手順に従って、Helm およびその他の OCI メディアタイプを使用するようにシステムを事前設定します。

10.1.1. Helm のインストール

Helm クライアントをインストールするには、次の手順を実行します。

手順

1. Helm リリース ページから Helm の最新バージョンをダウンロードします。
2. 次のコマンドを入力して Helm バイナリーを解凍します。

```
$ tar -zxvf helm-v3.8.2-linux-amd64.tar.gz
```

3. Helm バイナリーを目的の場所に移動します。

```
$ mv linux-amd64/helm /usr/local/bin/helm
```

Helm のインストールの詳細は、Helm のインストール ドキュメントを参照してください。

10.1.2. Helm 3.8 へのアップグレード

OCI レジストリーチャートをサポートするには、Helm が少なくとも 3.8 にアップグレードされている必要があります。すでに Helm をダウンロードしていて、Helm 3.8 にアップグレードする必要がある場合は、Helm アップグレード のドキュメントを参照してください。

10.2. HELM チャートの使用

以下の例を使用して、Red Hat Community of Practice (CoP) リポジトリーから etherpad チャートをダウンロードしてプッシュします。

前提条件

- Quay.io にログインしている。

手順

1. 次のコマンドを入力して、チャートリポジトリーを追加します。

```
$ helm repo add redhat-cop https://redhat-cop.github.io/helm-charts
```

2. 次のコマンドを入力して、チャートリポジトリーから、ローカルで使用可能なチャートの情報を更新します。

```
$ helm repo update
```

3. 次のコマンドを入力して、リポジトリーからチャートを取得します。

```
$ helm pull redhat-cop/etherpad --version=0.0.4 --untar
```

4. 次のコマンドを入力して、チャートをチャートアーカイブにパッケージ化します。

```
$ helm package ./etherpad
```

出力例

```
Successfully packaged chart and saved it to: /home/user/linux-amd64/etherpad-0.0.4.tgz
```

5. **helm registry login** を使用して Quay.io にログインします。

```
$ helm registry login quay.io
```

6. **helm push** コマンドを使用して、チャートをリポジトリーにプッシュします。

```
helm push etherpad-0.0.4.tgz oci://quay.io/<organization_name>/helm
```

出力例:

```
Pushed: quay370.apps.quayperf370.perfscale.devcluster.openshift.com/etherpad:0.0.4
Digest: sha256:a6667ff2a0e2bd7aa4813db9ac854b5124ff1c458d170b70c2d2375325f2451b
```

7. ローカルコピーを削除してから、リポジトリーからチャートをプルして、プッシュが機能したことを確認します。

```
$ rm -rf etherpad-0.0.4.tgz
```

```
$ helm pull oci://quay.io/<organization_name>/helm/etherpad --version 0.0.4
```

出力例:

```
Pulled: quay370.apps.quayperf370.perfscale.devcluster.openshift.com/etherpad:0.0.4
Digest: sha256:4f627399685880daf30cf77b6026dc129034d68c7676c7e07020b70cf7130902
```

10.3. COSIGN OCI のサポート

Cosign は、コンテナーイメージの署名および検証に使用できるツールです。**ECDSA-P256** 署名アルゴリズムおよび Red Hat の Simple Signing ペイロード形式を使用して、PKIX ファイルに保存される公開鍵を作成します。秘密鍵は暗号化された PEM ファイルとして保存されます。

Cosign は現在、以下をサポートしています。

- ハードウェアおよび KMS の署名
- 自身の PKI を使用
- OIDC PKI
- 組み込みのバイナリー透過性およびタイムスタンプサービス

Cosign を直接インストールするには、次の手順を実行します。

前提条件

- Go バージョン 1.16 以降がインストールされている。

手順

1. 次の **go** コマンドを入力して、Cosign を直接インストールします。

```
$ go install github.com/sigstore/cosign/cmd/cosign@v1.0.0
```

出力例

```
go: downloading github.com/sigstore/cosign v1.0.0
go: downloading github.com/peterbourgon/ff/v3 v3.1.0
```

2. 次のコマンドを入力して、Cosign 用のキーと値のペアを生成します。

```
$ cosign generate-key-pair
```

出力例

```
Enter password for private key:
Enter again:
Private key written to cosign.key
Public key written to cosign.pub
```

3. 次のコマンドを入力して、キーと値のペアに署名します。

```
$ cosign sign -key cosign.key quay.io/user1/busybox:test
```

出力例

```
Enter password for private key:
Pushing signature to: quay-server.example.com/user1/busybox:sha256-
ff13b8f6f289b92ec2913fa57c5dd0a874c3a7f8f149aabee50e3d01546473e3.sig
```

認証に **~./docker/config.json** に依存していることが原因で起こる **error: signing quay-server.example.com/user1/busybox:test: getting remote image: GET https://quay-server.example.com/v2/user1/busybox/manifests/test: UNAUTHORIZED: access to the requested resource is not authorized; map[]** エラーが発生した場合は、次のコマンドを実行する必要がある場合があります。

```
$ podman login --authfile ~/.docker/config.json quay.io
```

出力例

```
Username:
Password:
Login Succeeded!
```

4. 次のコマンドを入力して、更新された認可設定を確認します。

```
$ cat ~/.docker/config.json
{
 "auths": {
  "quay-server.example.com": {
   "auth": "cXVheWFkbWluOnBhc3N3b3Jk"
  }
 }
```

10.4. COSIGN のインストールと使用

Cosign を直接インストールするには、次の手順を実行します。

前提条件

- Go バージョン 1.16 以降がインストールされている。
- **config.yaml** ファイルで **FEATURE_GENERAL_OCI_SUPPORT** を **true** に設定している。

手順

1. 次の **go** コマンドを入力して、Cosign を直接インストールします。

```
$ go install github.com/sigstore/cosign/cmd/cosign@v1.0.0
```

出力例

```
go: downloading github.com/sigstore/cosign v1.0.0
go: downloading github.com/peterbourgon/ff/v3 v3.1.0
```

2. 次のコマンドを入力して、Cosign 用のキーと値のペアを生成します。

```
$ cosign generate-key-pair
```

出力例

```
Enter password for private key:
Enter again:
Private key written to cosign.key
Public key written to cosign.pub
```

3. 次のコマンドを入力して、キーと値のペアに署名します。

```
$ cosign sign -key cosign.key quay.io/user1/busybox:test
```

出力例

```
Enter password for private key:
Pushing signature to: quay-server.example.com/user1/busybox:sha256-
ff13b8f6f289b92ec2913fa57c5dd0a874c3a7f8f149aabee50e3d01546473e3.sig
```

認証に **~./docker/config.json** に依存していることが原因で起こる **error: signing quay-server.example.com/user1/busybox:test: getting remote image: GET https://quay-server.example.com/v2/user1/busybox/manifests/test: UNAUTHORIZED: access to the requested resource is not authorized; map[]** エラーが発生した場合は、次のコマンドを実行する必要がある場合があります。

```
$ podman login --authfile ~/.docker/config.json quay.io
```

出力例

```
Username:
Password:
Login Succeeded!
```

4. 次のコマンドを入力して、更新された認可設定を確認します。

```
$ cat ~/.docker/config.json
{
 "auths": {
  "quay-server.example.com": {
```

```
    "auth": "cXVheWFkbWluOnBhc3N3b3Jk"
  }
 }
```