Red Hat Quay 3.12

Troubleshooting Red Hat Quay

Troubleshooting Red Hat Quay

Last Updated: 2024-12-09

Red Hat Quay 3.12 Troubleshooting Red Hat Quay

Troubleshooting Red Hat Quay

Legal Notice

Abstract

Troubleshooting Red Hat Quay

Table of Contents

PREFACE

Red Hat offers administrators tools for gathering data for your Red Hat Quay deployment. You can use this data to troubleshoot your Red Hat Quay deployment yourself, or file a support ticket.

CHAPTER 1. GETTING SUPPORT

If you experience difficulty with a procedure described in this documentation, or with Red Hat Quay in general, visit the Red Hat Customer Portal . From the Customer Portal, you can:

- Search or browse through the Red Hat Knowledgebase of articles and solutions relating to Red Hat products.
- Submit a support case to Red Hat Support.
- Access other product documentation.

To identify issues with your deployment, you can use the Red Hat Quay debugging tool, or check the health endpoint of your deployment to obtain information about your problem. After you have debugged or obtained health information about your deployment, you can search the Red Hat Knowledgebase for a solution or file a support ticket.

If you have a suggestion for improving this documentation or have found an error, submit a Jira issue to the **ProjectQuay** project. Provide specific details, such as the section name and Red Hat Quay version.

1.1. ABOUT THE RED HAT KNOWLEDGEBASE

The Red Hat Knowledgebase provides rich content aimed at helping you make the most of Red Hat's products and technologies. The Red Hat Knowledgebase consists of articles, product documentation, and videos outlining best practices on installing, configuring, and using Red Hat products. In addition, you can search for solutions to known issues, each providing concise root cause descriptions and remedial steps.

The Red Hat Quay Support Team also maintains a Consolidate troubleshooting article for Red Hat Quay that details solutions to common problems. This is an evolving document that can help users navigate various issues effectively and efficiently.

1.2. SEARCHING THE RED HAT KNOWLEDGEBASE

In the event of an Red Hat Quay issue, you can perform an initial search to determine if a solution already exists within the Red Hat Knowledgebase.

Prerequisites

- You have a Red Hat Customer Portal account.

Procedure

1. Log in to the Red Hat Customer Portal .
2. In the main Red Hat Customer Portal search field, input keywords and strings relating to the problem, including:
   - Red Hat Quay components (such as **database**)
   - Related procedure (such as **installation**)
   - Warnings, error messages, and other outputs related to explicit failures
3. Click **Search**.

4. Select the **Red Hat Quay** product filter.
5. Select the **Knowledgebase** content type filter.

1.3. SUBMITTING A SUPPORT CASE

Prerequisites

- You have a Red Hat Customer Portal account.
- You have a Red Hat standard or premium Subscription.

Procedure

1. Log in to the Red Hat Customer Portal and select **Open a support case**.
2. Select the **Troubleshoot** tab.
3. For **Summary**, enter a concise but descriptive problem summary and further details about the symptoms being experienced, as well as your expectations.
4. Review the list of suggested Red Hat Knowledgebase solutions for a potential match against the problem that is being reported. If the suggested articles do not address the issue, continue to the following step.
5. For **Product**, select **Red Hat Quay**.
6. Select the version of Red Hat Quay that you are using.
7. Click **Continue**.
8. Optional. Drag and drop, paste, or browse to upload a file. This could be debug logs gathered from your Red Hat Quay deployment.
9. Click **Get support** to file your ticket.

CHAPTER 2. RUNNING RED HAT QUAY IN DEBUG MODE

Red Hat recommends gathering your debugging information when opening a support case. Running Red Hat Quay in debug mode provides verbose logging to help administrators find more information about various issues. Enabling debug mode can speed up the process to reproduce errors and validate a solution for things like geo-replication deployments, Operator deployments, standalone Red Hat Quay deployments, object storage issues, and so on. Additionally, it helps the Red Hat Support to perform a root cause analysis.

2.1. RED HAT QUAY DEBUG VARIABLES

Red Hat Quay offers two configuration fields that can be added to your **config.yaml** file to help diagnose issues or help obtain log information.

Table 2.1. Debug configuration variables

Variable	Type	Description
DEBUGLOG	Boolean	Whether to enable or disable debug logs. Must be **true** or **false**.
USERS_DEBUG	Integer. Either **0** or **1**.	Used to debug LDAP operations in clear text, including passwords. Must be used with **DEBUGLOG=TRUE**.  **IMPORTANT** Setting **USERS_DEBUG=1** exposes credentials in clear text. This variable should be removed from the Red Hat Quay deployment after debugging. The log file that is generated with this environment variable should be scrutinized, and passwords should be removed before sending to other users. Use with caution.

2.2. RUNNING A STANDALONE RED HAT QUAY DEPLOYMENT IN DEBUG MODE

Running Red Hat Quay in debug mode provides verbose logging to help administrators find more information about various issues. Enabling debug mode can speed up the process to reproduce errors and validate a solution.

Use the following procedure to run a standalone deployment of Red Hat Quay in debug mode.

Procedure

1. Enter the following command to run your standalone Red Hat Quay deployment in debug mode:

```
$ podman run -p 443:8443 -p 80:8080 -e DEBUGLOG=true  -v /config:/conf/stack -v
/storage:/datastorage -d {productrepo}/{quayimage}:{productminv}
```

2. To view the debug logs, enter the following command:

```
$ podman logs <quay_container_name>
```

2.3. RUNNING AN LDAP RED HAT QUAY DEPLOYMENT IN DEBUG MODE

Use the following procedure to run an LDAP deployment of Red Hat Quay in debug mode.

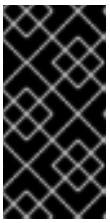
Procedure

1. Enter the following command to run your LDAP Red Hat Quay deployment in debug mode:

```
$ podman run -p 443:8443 -p 80:8080 -e DEBUGLOG=true -e USERS_DEBUG=1  -v
/config:/conf/stack -v /storage:/datastorage -d {productrepo}/{quayimage}:{productminv}
```

2. To view the debug logs, enter the following command:

```
$ podman logs <quay_container_name>
```



IMPORTANT

Setting **USERS_DEBUG=1** exposes credentials in clear text. This variable should be removed from the Red Hat Quay deployment after debugging. The log file that is generated with this environment variable should be scrutinized, and passwords should be removed before sending to other users. Use with caution.

2.4. RUNNING THE RED HAT QUAY OPERATOR IN DEBUG MODE

Use the following procedure to run the Red Hat Quay Operator in debug mode.

Procedure

1. Enter the following command to edit the **QuayRegistry** custom resource definition:

```
$ oc edit quayregistry <quay_registry_name> -n <quay_namespace>
```

2. Update the **QuayRegistry** to add the following parameters:

-

```
spec:
  - kind: quay
    managed: true
    overrides:
      env:
      - name: DEBUGLOG
        value: "true"
```

3. After the Red Hat Quay Operator has restarted with debugging enabled, try pulling an image from the registry. If it is still slow, dump all logs from all **Quay** pods to a file, and check the files for more information.

CHAPTER 3. LOGGING INFORMATION FOR RED HAT QUAY

Obtaining log information using can be beneficial in various ways for managing, monitoring, and troubleshooting applications running in containers or pods. Some of the reasons why obtaining log information is valuable include the following:

- **Debugging and Troubleshooting**: Logs provide insights into what's happening inside the application, allowing developers and system administrators to identify and resolve issues. By analyzing log messages, one can identify errors, exceptions, warnings, or unexpected behavior that might occur during the application's execution.
- **Performance Monitoring**: Monitoring logs helps to track the performance of the application and its components. Monitoring metrics like response times, request rates, and resource utilization can help in optimizing and scaling the application to meet the demand.
- **Security Analysis**: Logs can be essential in auditing and detecting potential security breaches. By analyzing logs, suspicious activities, unauthorized access attempts, or any abnormal behavior can be identified, helping in detecting and responding to security threats.
- **Tracking User Behavior**: In some cases, logs can be used to track user activities and behavior. This is particularly important for applications that handle sensitive data, where tracking user actions can be useful for auditing and compliance purposes.
- **Capacity Planning**: Log data can be used to understand resource utilization patterns, which can aid in capacity planning. By analyzing logs, one can identify peak usage periods, anticipate resource needs, and optimize infrastructure accordingly.
- **Error Analysis**: When errors occur, logs can provide valuable context about what happened leading up to the error. This can help in understanding the root cause of the issue and facilitating the debugging process.
- **Verification of Deployment** Logging during the deployment process can help verify if the application is starting correctly and if all components are functioning as expected.
- **Continuous Integration/Continuous Deployment (CI/CD)**: In CI/CD pipelines, logging is essential to capture build and deployment statuses, allowing teams to monitor the success or failure of each stage.

3.1. OBTAINING LOG INFORMATION FOR RED HAT QUAY

Log information can be obtained for all types of Red Hat Quay deployments, including geo-replication deployments, standalone deployments, and Operator deployments. Log information can also be obtained for mirrored repositories. It can help you troubleshoot authentication and authorization issues, and object storage issues. After you have obtained the necessary log information, you can search the Red Hat Knowledgebase for a solution, or file a support ticket with the Red Hat Support team.

Use the following procedure to obtain logs for your Red Hat Quay deployment.

Procedure

- If you are using the Red Hat Quay Operator on OpenShift Container Platform, enter the following command to view the logs:

```
$ oc logs <quay_pod_name>
```

- If you are on a standalone Red Hat Quay deployment, enter the following command:

```
$ podman logs <quay_container_name>
```

Example output

```
...
gunicorn-web stdout | 2023-01-20 15:41:52,071 [205] [DEBUG] [app] Starting request:
urn:request:0d88de25-03b0-4cf9-b8bc-87f1ac099429 (/oauth2/azure/callback) {'X-
Forwarded-For': '174.91.79.124'}
...
```

3.2. EXAMINING VERBOSE LOGS

Red Hat Quay does not have verbose logs, however, with the following procedures, you can obtain a detailed status check of your database pod or container.

NOTE

Additional debugging information can be returned if you have deployed Red Hat Quay in one of the following ways:

- You have deployed Red Hat Quay by passing in the **DEBUGLOG=true** variable.
- You have deployed Red Hat Quay with LDAP authentication enabled by passing in the **DEBUGLOG=true** and **USERS_DEBUG=1** variables.
- You have configured Red Hat Quay on OpenShift Container Platform by updating the **QuayRegistry** resource to include **DEBUGLOG=true**.

For more information, see "Running Red Hat Quay in debug mode".

Procedure

1. Enter the following commands to examine verbose database logs.
   a. If you are using the Red Hat Quay Operator on OpenShift Container Platform, enter the following commands:

```
$ oc logs <quay_pod_name> --previous
```

```
$ oc logs <quay_pod_name> --previous -c <container_name>
```

```
$ oc cp <quay_pod_name>:/var/lib/pgsql/data/userdata/log/*
/path/to/desired_directory_on_host
```

   b. If you are using a standalone deployment of Red Hat Quay, enter the following commands:

```
$ podman logs <quay_container_id> --previous
```

```
$ podman logs <quay_container_id> --previous -c <container_name>
```

```
$ podman cp <quay_container_id>:/var/lib/pgsql/data/userdata/log/*
/path/to/desired_directory_on_host
```

CHAPTER 4. CONFIGURATION INFORMATION FOR RED HAT QUAY

Checking a configuration YAML can help identify and resolve various issues related to the configuration of Red Hat Quay. Checking the configuration YAML can help you address the following issues:

- **Incorrect Configuration Parameters**: If the database is not functioning as expected or is experiencing performance issues, your configuration parameters could be at fault. By checking the configuration YAML, administrators can ensure that all the required parameters are set correctly and match the intended settings for the database.
- **Resource Limitations**: The configuration YAML might specify resource limits for the database, such as memory and CPU limits. If the database is running into resource constraints or experiencing contention with other services, adjusting these limits can help optimize resource allocation and improve overall performance.
- **Connectivity Issues**: Incorrect network settings in the configuration YAML can lead to connectivity problems between the application and the database. Ensuring that the correct network configurations are in place can resolve issues related to connectivity and communication.
- **Data Storage and Paths**: The configuration YAML may include paths for storing data and logs. If the paths are misconfigured or inaccessible, the database may encounter errors while reading or writing data, leading to operational issues.
- **Authentication and Security**: The configuration YAML may contain authentication settings, including usernames, passwords, and access controls. Verifying these settings is crucial for maintaining the security of the database and ensuring only authorized users have access.
- **Plugin and Extension Settings**: Some databases support extensions or plugins that enhance functionality. Issues may arise if these plugins are misconfigured or not loaded correctly. Checking the configuration YAML can help identify any problems with plugin settings.
- **Replication and High Availability Settings**: In clustered or replicated database setups, the configuration YAML may define replication settings and high availability configurations. Incorrect settings can lead to data inconsistency and system instability.
- **Backup and Recovery Options** The configuration YAML might include backup and recovery options, specifying how data backups are performed and how data can be recovered in case of failures. Validating these settings can ensure data safety and successful recovery processes.

By checking your configuration YAML, Red Hat Quay administrators can detect and resolve these issues before they cause significant disruptions to the application or service relying on the database.

4.1. OBTAINING CONFIGURATION INFORMATION FOR RED HAT QUAY

Configuration information can be obtained for all types of Red Hat Quay deployments, include standalone, Operator, and geo-replication deployments. Obtaining configuration information can help you resolve issues with authentication and authorization, your database, object storage, and repository mirroring. After you have obtained the necessary configuration information, you can update your **config.yaml** file, search the Red Hat Knowledgebase for a solution, or file a support ticket with the Red Hat Support team.

Procedure

1. To obtain configuration information on Red Hat Quay Operator deployments, you can use **oc exec**, **oc cp**, or **oc rsync**.

   a. To use the **oc exec** command, enter the following command:

   ```
   $ oc exec -it <quay_pod_name> -- cat /conf/stack/config.yaml
   ```

   This command returns your **config.yaml** file directly to your terminal.

   b. To use the **oc copy** command, enter the following commands:

   ```
   $ oc cp <quay_pod_name>:/conf/stack/config.yaml /tmp/config.yaml
   ```

   To display this information in your terminal, enter the following command:

   ```
   $ cat /tmp/config.yaml
   ```

   c. To use the **oc rsync** command, enter the following commands:

   ```
   oc rsync <quay_pod_name>:/conf/stack/ /tmp/local_directory/
   ```

   To display this information in your terminal, enter the following command:

   ```
   $ cat /tmp/local_directory/config.yaml
   ```

   Example output

   ```
   DISTRIBUTED_STORAGE_CONFIG:
   local_us:
   - RHOCSStorage
   - access_key: redacted
     bucket_name: lht-quay-datastore-68fff7b8-1b5e-46aa-8110-c4b7ead781f5
     hostname: s3.openshift-storage.svc.cluster.local
     is_secure: true
     port: 443
     secret_key: redacted
     storage_path: /datastorage/registry
   DISTRIBUTED_STORAGE_DEFAULT_LOCATIONS:
   - local_us
   DISTRIBUTED_STORAGE_PREFERENCE:
   - local_us
   ```

2. To obtain configuration information on standalone Red Hat Quay deployments, you can use **podman cp** or **podman exec**.

   a. To use the **podman copy** command, enter the following commands:

   ```
   $ podman cp <quay_container_id>:/conf/stack/config.yaml /tmp/local_directory/
   ```

   To display this information in your terminal, enter the following command:

   ```
   $ cat /tmp/local_directory/config.yaml
   ```

b. To use **podman exec**, enter the following commands:

```
$ podman exec -it <quay_container_id> cat /conf/stack/config.yaml
```

Example output

```
BROWSER_API_CALLS_XHR_ONLY: false
ALLOWED_OCI_ARTIFACT_TYPES:
  application/vnd.oci.image.config.v1+json:
    - application/vnd.oci.image.layer.v1.tar+zstd
  application/vnd.sylabs.sif.config.v1+json:
    - application/vnd.sylabs.sif.layer.v1+tar
AUTHENTICATION_TYPE: Database
AVATAR_KIND: local
BUILDLOGS_REDIS:
  host: quay-server.example.com
  password: strongpassword
  port: 6379
DATABASE_SECRET_KEY: 05ee6382-24a6-43c0-b30f-849c8a0f7260
DB_CONNECTION_ARGS: {}
---
```

4.2. OBTAINING DATABASE CONFIGURATION INFORMATION

You can obtain configuration information about your database by using the following procedure.

WARNING

Interacting with the PostgreSQL database is potentially destructive. It is highly recommended that you perform the following procedure with the help of a Red Hat Quay Support Specialist.

Procedure

- If you are using the Red Hat Quay Operator on OpenShift Container Platform, enter the following command:

```
$ oc exec -it <database_pod> -- cat /var/lib/pgsql/data/userdata/postgresql.conf
```

- If you are using a standalone deployment of Red Hat Quay, enter the following command:

```
$ podman exec -it <database_container> cat /var/lib/pgsql/data/userdata/postgresql.conf
```

CHAPTER 5. PERFORMING HEALTH CHECKS ON RED HAT QUAY DEPLOYMENTS

Health check mechanisms are designed to assess the health and functionality of a system, service, or component. Health checks help ensure that everything is working correctly, and can be used to identify potential issues before they become critical problems. By monitoring the health of a system, Red Hat Quay administrators can address abnormalities or potential failures for things like geo-replication deployments, Operator deployments, standalone Red Hat Quay deployments, object storage issues, and so on. Performing health checks can also help reduce the likelihood of encountering troubleshooting scenarios.

Health check mechanisms can play a role in diagnosing issues by providing valuable information about the system's current state. By comparing health check results with expected benchmarks or predefined thresholds, deviations or anomalies can be identified quicker.

5.1. RED HAT QUAY HEALTH CHECK ENDPOINTS

IMPORTANT

Links contained herein to any external website(s) are provided for convenience only. Red Hat has not reviewed the links and is not responsible for the content or its availability. The inclusion of any link to an external website does not imply endorsement by Red Hat of the website or its entities, products, or services. You agree that Red Hat is not responsible or liable for any loss or expenses that may result due to your use of (or reliance on) the external site or content.

Red Hat Quay has several health check endpoints. The following table shows you the health check, a description, an endpoint, and an example output.

Table 5.1. Health check endpoints

Health check	Description	Endpoint	Example output
instance	The **instance** endpoint acquires the entire status of the specific Red Hat Quay instance. Returns a **dict** with key-value pairs for the following: **auth**, **database**, **disk_space**, **registry_gunicorn**, **service_key**, and **web_gunicorn.** Returns a number indicating the health check response of either **200**, which indicates that the instance is healthy, or **503**, which indicates an issue with your deployment.	**https://{quay-ip-endpoint}/health/instance** *or* **https://{quay-ip-endpoint}/health**	**{"data":{"services":{"auth":true,"database":true,"disk_space":true,"registry_gunicorn":true,"service_key":true,"web_gunicorn":true}},"status_code":200}**

Health check	Description	Endpoint	Example output
endtoend	The **endtoend** endpoint conducts checks on all services of your Red Hat Quay instance. Returns a **dict** with key-value pairs for the following: **auth**, **database**, **redis**, **storage**. Returns a number indicating the health check response of either **200**, which indicates that the instance is healthy, or **503**, which indicates an issue with your deployment.	**https://{quay-ip-endpoint}/health/endtoend**	**{"data":{"services":{"auth":true,"database":true,"redis":true,"storage":true}},"status_code":200}**
warning	The **warning** endpoint conducts a check on the warnings. Returns a **dict** with key-value pairs for the following: **disk_space_warning**. Returns a number indicating the health check response of either **200**, which indicates that the instance is healthy, or **503**, which indicates an issue with your deployment.	**https://{quay-ip-endpoint}/health/warning**	**{"data":{"services":{"disk_space_warning":true}},"status_code":503}**

5.2. NAVIGATING TO A RED HAT QUAY HEALTH CHECK ENDPOINT

Use the following procedure to navigate to the **instance** endpoint. This procedure can be repeated for **endtoend** and **warning** endpoints.

Procedure

1. On your web browser, navigate to **https://{quay-ip-endpoint}/health/instance**.
2. You are taken to the health instance page, which returns information like the following:

```
{"data":{"services":
{"auth":true,"database":true,"disk_space":true,"registry_gunicorn":true,"service_key":true,"web_gunicorn":true}},"status_code":200}
```

For Red Hat Quay, **"status_code": 200** means that the instance is health. Conversely, if you receive **"status_code": 503**, there is an issue with your deployment.

CHAPTER 6. TROUBLESHOOTING RED HAT QUAY COMPONENTS

This document focuses on troubleshooting specific components within Red Hat Quay, providing targeted guidance for resolving issues that might arise. Designed for system administrators, operators, and developers, this resource aims to help diagnose and troubleshoot problems related to individual components of Red Hat Quay.

In addition to the following procedures, Red Hat Quay components can also be troubleshot by running Red Hat Quay in debug mode, obtaining log information, obtaining configuration information, and performing health checks on endpoints.

By using the following procedures, you are able to troubleshoot common component issues. Afterwards, you can search for solutions on the Red Hat Knowledgebase , or file a support ticket with the Red Hat Support team.

6.1. TROUBLESHOOTING THE RED HAT QUAY DATABASE

The PostgreSQL database used for Red Hat Quay store various types of information related to container images and their management. Some of the key pieces of information that the PostgreSQL database stores includes:

- **Image Metadata**. The database stores metadata associated with container images, such as image names, versions, creation timestamps, and the user or organization that owns the image. This information allows for easy identification and organization of container images within the registry.
- **Image Tags**. Red Hat Quay allows users to assign tags to container images, enabling convenient labeling and versioning. The PostgreSQL database maintains the mapping between image tags and their corresponding image manifests, allowing users to retrieve specific versions of container images based on the provided tags.
- **Image Layers**. Container images are composed of multiple layers, which are stored as individual objects. The database records information about these layers, including their order, checksums, and sizes. This data is crucial for efficient storage and retrieval of container images.
- **User and Organization Data**. Red Hat Quay supports user and organization management, allowing users to authenticate and manage access to container images. The PostgreSQL database stores user and organization information, including usernames, email addresses, authentication tokens, and access permissions.
- **Repository Information**. Red Hat Quay organizes container images into repositories, which act as logical units for grouping related images. The database maintains repository data, including names, descriptions, visibility settings, and access control information, enabling users to manage and share their repositories effectively.
- **Event Logs**. Red Hat Quay tracks various events and activities related to image management and repository operations. These event logs, including image pushes, pulls, deletions, and repository modifications, are stored in the PostgreSQL database, providing an audit trail and allowing administrators to monitor and analyze system activities.

The content in this section covers the following procedures:

- **Checking the type of deployment** Determine if the database is deployed as a container on a virtual machine or as a pod on OpenShift Container Platform.

- **Checking the container or pod status** Verify the status of the **database** pod or container using specific commands based on the deployment type.
- **Examining the database container or pod logs** Access and examine the logs of the database pod or container, including commands for different deployment types.
- **Checking the connectivity between Red Hat Quay and the database pod**Check the connectivity between Red Hat Quay and the **database** pod using relevant commands.
- **Checking the database configuration**: Check the database configuration at various levels (OpenShift Container Platform or PostgreSQL level) based on the deployment type.
- **Checking resource allocation**: Monitor resource allocation for the Red Hat Quay deployment, including disk usage and other resource usage.
- **Interacting with the Red Hat Quay database** Learn how to interact with the PostgreSQL database, including commands to access and query databases.

6.1.1. Troubleshooting Red Hat Quay database issues

Use the following procedures to troubleshoot the PostgreSQL database.

6.1.1.1. Interacting with the Red Hat Quay database

Use the following procedure to interact with the PostgreSQL database.

WARNING

Interacting with the PostgreSQL database is potentially destructive. It is highly recommended that you perform the following procedure with the help of a Red Hat Quay Support Specialist.

NOTE

Interacting with the PostgreSQL database can also be used to troubleshoot authorization and authentication issues.

Procedure

1. Exec into the Red Hat Quay database.
   a. Enter the following commands to exec into the Red Hat Quay database pod on OpenShift Container Platform:

```
$ oc exec -it <quay_database_pod> -- psql
```

   b. Enter the following command to exec into the Red Hat Quay database on a standalone deployment:

```
$ sudo podman exec -it <quay_container_name> /bin/bash
```

2. Enter the PostgreSQL shell.

WARNING

Interacting with the PostgreSQL database is potentially destructive. It is highly recommended that you perform the following procedure with the help of a Red Hat Quay Support Specialist.

a. If you are using the Red Hat Quay Operator, enter the following command to enter the PostgreSQL shell:

```
$ oc rsh <quay_pod_name> psql -U your_username -d your_database_name
```

b. If you are on a standalone Red Hat Quay deployment, enter the following command to enter the PostgreSQL shell:

```
bash-4.4$ psql -U your_username -d your_database_name
```

6.1.1.2. Troubleshooting crashloopbackoff states

Use the following procedure to troueblshoot **crashloopbackoff** states.

Procedure

1. If your container or pod is in a **crashloopbackoff** state, you can enter the following commands.

a. Enter the following command to scale down the Red Hat Quay Operator:

```
$ oc scale deployment/quay-operator.v3.8.z  --replicas=0
```

Example output

```
deployment.apps/quay-operator.v3.8.z scaled
```

b. Enter the following command to scale down the Red Hat Quay database:

```
$ oc scale deployment/<quay_database> --replicas=0
```

Example output

```
deployment.apps/<quay_database> scaled
```

c. Enter the following command to edit the Red Hat Quay database:

WARNING

Interacting with the PostgreSQL database is potentially destructive. It is highly recommended that you perform the following procedure with the help of a Red Hat Quay Support Specialist.

```
$ oc edit deployment <quay_database>
```

```
...
 template:
   metadata:
     creationTimestamp: null
     labels:
       quay-component: <quay_database>
       quay-operator/quayregistry: quay-operator.v3.8.z
   spec:
     containers:
     - env:
       - name: POSTGRESQL_USER
         value: postgres
       - name: POSTGRESQL_DATABASE
         value: postgres
       - name: POSTGRESQL_PASSWORD
         value: postgres
       - name: POSTGRESQL_ADMIN_PASSWORD
         value: postgres
       - name: POSTGRESQL_MAX_CONNECTIONS
         value: "1000"
       image: registry.redhat.io/rhel8/postgresql-
10@sha256:a52ad402458ec8ef3f275972c6ebed05ad64398f884404b9bb8e3010c5c95291

       imagePullPolicy: IfNotPresent
       name: postgres
       command: ["/bin/bash", "-c", "sleep 86400"] 1
...
```

1 Add this line in the same indentation.

Example output

```
deployment.apps/<quay_database> edited
```

d. Execute the following command inside of your **<quay_database>**:

```
$ oc exec -it <quay_database> -- cat /var/lib/pgsql/data/userdata/postgresql/logs/*
/path/to/desired_directory_on_host
```

6.1.1.3. Checking the connectivity between Red Hat Quay and the database pod

Use the following procedure to check the connectivity between Red Hat Quay and the database pod

Procedure

1. Check the connectivity between Red Hat Quay and the database pod.
   a. If you are using the Red Hat Quay Operator on OpenShift Container Platform, enter the following command:

```
$ oc exec -it _quay_pod_name_ -- curl -v telnet://<database_pod_name>:5432
```

   b. If you are using a standalone deployment of Red Hat Quay, enter the following command:

```
$ podman exec -it <quay_container_name >curl -v
telnet://<database_container_name>:5432
```

6.1.1.4. Checking resource allocation

Use the following procedure to check resource allocation.

Procedure

1. Obtain a list of running containers.
2. Monitor disk usage of your Red Hat Quay deployment.
   a. If you are using the Red Hat Quay Operator on OpenShift Container Platform, enter the following command:

```
$ oc exec -it <quay_database_pod_name> -- df -ah
```

   b. If you are using a standalone deployment of Red Hat Quay, enter the following command:

```
$ podman exec -it <quay_database_conatiner_name> df -ah
```

3. Monitor other resource usage.
   a. Enter the following command to check resource allocation on a Red Hat Quay Operator deployment:

```
$ oc adm top pods
```

   b. Enter the following command to check the status of a specific pod on a standalone deployment of Red Hat Quay:

```
$ podman pod stats <pod_name>
```

   c. Enter the following command to check the status of a specific container on a standalone deployment of Red Hat Quay:

```
$ podman stats <container_name>
```

   The following information is returned:

- **CPU %**. The percentage of CPU usage by the container since the last measurement. This value represents the container's share of the available CPU resources.
- **MEM USAGE / LIMIT**. The current memory usage of the container followed by its memory limit. The values are displayed in the format **current_usage** / **memory_limit**. For example, **300.4MiB** / **7.795GiB** indicates that the container is currently using 300.4 megabytes of memory out of a limit of 7.795 gigabytes.
- **MEM %**. The percentage of memory usage by the container in relation to its memory limit.
- **NET I/O**. The network I/O (input/output) statistics of the container. It displays the amount of data transmitted and received by the container over the network. The values are displayed in the format: **transmitted_bytes** / **received_bytes**.
- **BLOCK I/O**. The block I/O (input/output) statistics of the container. It represents the amount of data read from and written to the block devices (for example, disks) used by the container. The values are displayed in the format **read_bytes** / **written_bytes**.

6.1.2. Resetting superuser passwords on Red Hat Quay standalone deployments

Use the following procedure to reset a superuser's password.

Prerequisites

- You have created a Red Hat Quay superuser.
- You have installed Python 3.9.
- You have installed the **pip** package manager for Python.
- You have installed the **bcrypt** package for **pip**.

Procedure

1. Generate a secure, hashed password using the **bcrypt** package in Python 3.9 by entering the following command:

```
$ python3.9 -c 'import bcrypt; print(bcrypt.hashpw(b"newpass1234",
bcrypt.gensalt(12)).decode("utf-8"))'
```

Example output

```
$2b$12$T8pkgtOoys3G5ut7FV1She6vXlYgU.6TeoGmbbAVQtN8X8ch4knKm
```

2. Enter the following command to show the container ID of your Red Hat Quay container registry:

```
$ sudo podman ps -a
```

Example output

```
CONTAINER ID  IMAGE                                COMMAND       CREATED       STATUS
PORTS                   NAMES
70560beda7aa  registry.redhat.io/rhel8/redis-5:1    run-redis     2 hours ago   Up 2 hours
```

```
ago     0.0.0.0:6379->6379/tcp                  redis
8012f4491d10  registry.redhat.io/quay/quay-rhel8:v3.8.2  registry        3 minutes ago   Up 8
seconds ago   0.0.0.0:80->8080/tcp, 0.0.0.0:443->8443/tcp  quay
8b35b493ac05  registry.redhat.io/rhel8/postgresql-10:1   run-postgresql  39 seconds ago  Up
39 seconds ago  0.0.0.0:5432->5432/tcp                  postgresql-quay
```

3. Execute an interactive shell for the **postgresql** container image by entering the following command:

```
$ sudo podman exec -it 8b35b493ac05 /bin/bash
```

4. Re-enter the **quay** PostgreSQL database server, specifying the database, username, and host address:

```
bash-4.4$ psql -d quay -U quayuser -h 192.168.1.28 -W
```

5. Update the **password_hash** of the superuser admin who lost their password:

```
quay=> UPDATE public.user SET password_hash =
'$2b$12$T8pkgtOoys3G5ut7FV1She6vXlYgU.6TeoGmbbAVQtN8X8ch4knKm' where
username = 'quayadmin';
```

Example output

```
UPDATE 1
```

6. Enter the following to command to ensure that the **password_hash** has been updated:

```
quay=> select * from public.user;
```

Example output

```
id | uuid | username | password_hash | email | verified | stripe_id | organization | robot |
invoice_email | invalid_login_attempts | last_invalid_login |removed_tag_expiration_s |
enabled | invoice_email_address | company | family_name | given_name | location |
maximum_queued_builds_count | creation_date | last_accessed
----+--------------------------------------+----------+--------------------------------------------------
-------+----------------------+---
-------+----------+-------------+-------+--------------+----------------------+-------------------------
--+-------------------------+------
---+----------------------+---------+-------------+------------+----------+----------------------------+---
-------------------------+-----------
1 | 73f04ef6-19ba-41d3-b14d-f2f1eed94a4a | quayadmin |
$2b$12$T8pkgtOoys3G5ut7FV1She6vXlYgU.6TeoGmbbAVQtN8X8ch4knKm |
quayadmin@example.com | t | | f | f | f | 0 | 2023-02-23 07:54:39.116485 | 1209600 | t | | | | | |
| 2023-02-23 07:54:39.116492
```

7. Log in to your Red Hat Quay deployment using the new password:

```
$ sudo podman login -u quayadmin -p newpass1234 http://quay-server.example.com --tls-
verify=false
```

Example output

```
Login Succeeded!
```

Additional resources

For more information, see Resetting Superuser Password for Quay.

6.1.3. Resetting superuser passwords on the Red Hat Quay Operator

Prerequisites

- You have created a Red Hat Quay superuser.
- You have installed Python 3.9.
- You have installed the **pip** package manager for Python.
- You have installed the **bcrypt** package for **pip**.

Procedure

1. Log in to your Red Hat Quay deployment.
2. On the OpenShift Container Platform UI, navigate to **Workloads → Secrets**.
3. Select the namespace for your Red Hat Quay deployment, for example, **Project quay**.
4. Locate and store the PostgreSQL database credentials.
5. Generate a secure, hashed password using the **bcrypt** package in Python 3.9 by entering the following command:

```
$ python3.9 -c 'import bcrypt; print(bcrypt.hashpw(b"newpass1234",
bcrypt.gensalt(12)).decode("utf-8"))'
```

Example output

```
$2b$12$zoilcTG6XQeAoVuDuIZH0..UpvQEZcKh3V6puksQJaUQupHgJ4.4y
```

6. On the CLI, log in to the database, for example:

```
$ oc rsh quayuser-quay-quay-database-669c8998f-v9qsl
```

7. Enter the following command to open a connection to the **quay** PostgreSQL database server, specifying the database, username, and host address:

```
sh-4.4$ psql -U quayuser-quay-quay-database -d quayuser-quay-quay-database -W
```

8. Enter the following command to connect to the default database for the current user:

```
quay=> \c
```

9. Update the **password_hash** of the superuser admin who lost their password:

```
quay=> UPDATE public.user SET password_hash =
'$2b$12$zoilcTG6XQeAoVuDuIZH0..UpvQEZcKh3V6puksQJaUQupHgJ4.4y' where
username = 'quayadmin';
```

10. Enter the following to command to ensure that the **password_hash** has been updated:

```
quay=> select * from public.user;
```

Example output

```
id | uuid | username | password_hash | email | verified | stripe_id | organization | robot |
invoice_email | invalid_login_attempts | last_invalid_login |removed_tag_expiration_s |
enabled | invoice_email_address | company | family_name | given_name | location |
maximum_queued_builds_count | creation_date | last_accessed
----+--------------------------------------+-----------+--------------------------------------------------------
-------+----------------------+---
-------+-----------+--------------+-------+--------------+-----------------------+--------------------------
--+-------------------------+------
---+----------------------+---------+-------------+------------+----------+----------------------------+---
------------------------+-----------
1 | 73f04ef6-19ba-41d3-b14d-f2f1eed94a4a | quayadmin |
$2b$12$zoilcTG6XQeAoVuDuIZH0..UpvQEZcKh3V6puksQJaUQupHgJ4.4y |
quayadmin@example.com | t | | f | f | f | 0 | 2023-02-23 07:54:39.116485 | 1209600 | t | | | | | |
| 2023-02-23 07:54:39.116492
```

11. Navigate to your Red Hat Quay UI on OpenShift Container Platform and log in using the new credentials.

6.2. TROUBLESHOOTING RED HAT QUAY AUTHENTICATION

Authentication and authorization is crucial for secure access to Red Hat Quay. Together, they safeguard sensitive container images, verify user identities, enforce access controls, facilitate auditing and accountability, and enable seamless integration with external identity providers. By prioritizing authentication, organizations can bolster the overall security and integrity of their container registry environment.

The following authentication methods are supported by Red Hat Quay:

- **Username and password**. Users can authentication by providing their username and password, which are validated against the user database configured in Red Hat Quay. This traditional method requires users to enter their credentials to gain access.
- **OAuth**. Red Hat Quay supports OAuth authentication, which allows users to authenticate using their credentials from third party services like Google, GitHub, or Keycloak. OAuth enables a seamless and federated login experience, eliminating the need for separate account creation and simplifying user management.
- **OIDC**. OpenID Connect enables single sign-on (SSO) capabilities and integration with enterprise identity providers. With OpenID Connect, users can authenticate using their existing organizational credentials, providing a unified authentication experience across various systems and applications.

- **Token-based authentication**. Users can obtain unique tokens that grant access to specific resources within Red Hat Quay. Tokens can be obtained through various means, such as OAuth or by generating API tokens within the Red Hat Quay user interface. Token-based authentication is often used for automated or programmatic access to the registry.

- **External identity provider**. Red Hat Quay can integrate with external identity providers, such as LDAP or AzureAD, for authentication purposes. This integration allows organizations to use their existing identity management infrastructure, enabling centralized user authentication and reducing the need for separate user databases.

6.2.1. Troubleshooting Red Hat Quay authentication and authorization issues for specific users

Use the following procedure to troubleshoot authentication and authorization issues for specific users.

Procedure

1. Exec into the Red Hat Quay pod or container. For more information, see "Interacting with the Red Hat Quay database".

2. Enter the following command to show all users for external authentication:

```
quay=# select * from federatedlogin;
```

Example output

```
id | user_id | service_id |                service_ident                |                metadata_json
----+---------+------------+---------------------------------------------+-------------------------------------
------
1 |       1 |          3 | testuser0                                   | {}
2 |       1 |          8 | PK7Zpg2Yu2AnfUKG15hKNXqOXirqUog6G-oE7OgzSWc |
{"service_username": "live.com#testuser0"}
3 |       2 |          3 | testuser1                                   | {}
4 |       2 |          4 | 110875797246250333431                       | {"service_username":
"testuser1"}
5 |       3 |          3 | testuser2                                   | {}
6 |       3 |          1 | 26310880                                    | {"service_username": "testuser2"}
(6 rows)
```

3. Verify that the users are inserted into the **user** table:

```
quay=# select username, email from "user";
```

Example output

```
username  |        email
-----------+---------------------
testuser0 | testuser0@outlook.com
testuser1  | testuser1@gmail.com
testuser2  | testuser2@redhat.com
(3 rows)
```

6.3. TROUBLESHOOTING RED HAT QUAY OBJECT STORAGE

Object storage is a type of data storage architecture that manages data as discrete units called **objects**. Unlike traditional file systems that organize data into hierarchical directories and files, object storage treats data as independent entities with unique identifiers. Each object contains the data itself, along with metadata that describes the object and enables efficient retrieval.

Red Hat Quay uses object storage as the underlying storage mechanism for storing and managing container images. It stores container images as individual objects. Each container image is treated as an object, with its own unique identifier and associated metadata.

6.3.1. Troubleshooting Red Hat Quay object storage issues

Use the following options to troubleshoot Red Hat Quay object storage issues.

Procedure

- Enter the following command to see what object storage is used:

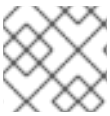
```
$ oc get quayregistry quay-registry-name -o yaml
```

- Ensure that the object storage you are using is officially supported by Red Hat Quay by checking the tested integrations page.
- Enable debug mode. For more information, see "Running Red Hat Quay in debug mode".
- Check your object storage configuration in your **config.yaml** file. Ensure that it is accurate and matches the settings provided by your object storage provider. You can check information like access credentials, endpoint URLs, bucket and container names, and other relevant configuration parameters.
- Ensure that Red Hat Quay has network connectivity to the object storage endpoint. Check the network configurations to ensure that there are no restrictions blocking the communication between Red Hat Quay and the object storage endpoint.
- If **FEATURE_STORAGE_PROXY** is enabled in your **config.yaml** file, check to see if its download URL is accessible. This can be found in the Red Hat Quay debug logs. For example:

```
$ curl -vvv
"https://QUAY_HOSTNAME/_storage_proxy/dhaWZKRjlyO.....Kuhc=/https/quay.hostname.com/quay-test/datastorage/registry/sha256/0e/0e1d17a1687fa270ba4f52a85c0f0e7958e13d3ded5123c3851a8031a9e55681?AWSAccessKeyId=xxxx&Signature=xxxxxx4%3D&Expires=1676066703"
```

- Try access the object storage service outside of Red Hat Quay to determine if the issue is specific to your deployment, or the underlying object storage. You can use command line tools like **aws**, **gsutil**, or **s3cmd** provided by the object storage provider to perform basic operations like listing buckets, containers, or uploading and downloading objects. This might help you isolate the problem.

6.4. GEO-REPLICATION



NOTE

Currently, the geo-replication feature is not supported on IBM Power.

Geo-replication allows multiple, geographically distributed Red Hat Quay deployments to work as a single registry from the perspective of a client or user. It significantly improves push and pull performance in a globally-distributed Red Hat Quay setup. Image data is asynchronously replicated in the background with transparent failover and redirect for clients.

Deployments of Red Hat Quay with geo-replication is supported on standalone and Operator deployments.

6.4.1. Troubleshooting geo-replication for Red Hat Quay

Use the following sections to troubleshoot geo-replication for Red Hat Quay.

6.4.1.1. Checking data replication in backend buckets

Use the following procedure to ensure that your data is properly replicated in all backend buckets.

Prerequisites

- You have installed the **aws** CLI.

Procedure

1. Enter the following command to ensure that your data is replicated in all backend buckets:

```
$ aws --profile quay_prod_s3 --endpoint=http://10.0.x.x:port s3 ls ocp-quay --recursive --human-readable --summarize
```

Example output

```
Total Objects: 17996
Total Size: 514.4 GiB
```

6.4.1.2. Checking the status of your backend storage

Use the following resources to check the status of your backend storage.

- **Amazon Web Service Storage (AWS)** Check the AWS S3 service health status on the AWS Service Health Dashboard. Validate your access to S3 by listing objects in a known bucket using the **aws** CLI or SDKs.
- **Google Cloud Storage (GCS)**. Check the Google Cloud Status Dashboard for the status of the GCS service. Verify your access to GCS by listing objects in a known bucket using the Google Cloud SDK or GCS client libraries.
- **NooBaa**. Check the NooBaa management console or administrative interface for any health or status indicators. Ensure that the NooBaa services and related components are running and accessible. Verify access to NooBaa by listing objects in a known bucket using the NooBaa CLI or SDK.
- **Red Hat OpenShift Data Foundation**. Check the OpenShift Container Platform Console or management interface for the status of the Red Hat OpenShift Data Foundation components. Verify the availability of Red Hat OpenShift Data Foundation S3 interface and services. Ensure

that the Red Hat OpenShift Data Foundation services are running and accessible. Validate access to Red Hat OpenShift Data Foundation S3 by listing objects in a known bucket using the appropriate S3-compatible SDK or CLI.

- **Ceph**. Check the status of Ceph services, including Ceph monitors, OSDs, and RGWs. Validate that the Ceph cluster is healthy and operational. Verify access to Ceph object storage by listing objects in a known bucket using the appropriate Ceph object storage API or CLI.
- **Azure Blob Storage**. Check the Azure Status Dashboard to see the health status of the Azure Blob Storage service. Validate your access to Azure Blob Storage by listing containers or objects using the Azure CLI or Azure SDKs.
- **OpenStack Swift**. Check the OpenStack Status page to verify the status of the OpenStack Swift service. Ensure that the Swift services, like the proxy server, container servers, object servers, are running and accessible. Validate your access to Swift by listing containers or objects using the appropriate Swift CLI or SDK.

After checking the status of your backend storage, ensure that all Red Hat Quay instances have access to all s3 storage backends.

6.5. REPOSITORY MIRRORING

Red Hat Quay repository mirroring lets you mirror images from external container registries, or another local registry, into your Red Hat Quay cluster. Using repository mirroring, you can synchronize images to Red Hat Quay based on repository names and tags.

From your Red Hat Quay cluster with repository mirroring enabled, you can perform the following:

- Choose a repository from an external registry to mirror
- Add credentials to access the external registry
- Identify specific container image repository names and tags to sync
- Set intervals at which a repository is synced
- Check the current state of synchronization

To use the mirroring functionality, you need to perform the following actions:

- Enable repository mirroring in the Red Hat Quay configuration file
- Run a repository mirroring worker
- Create mirrored repositories

All repository mirroring configurations can be performed using the configuration tool UI or by the Red Hat Quay API.

6.5.1. Troubleshooting repository mirroring

Use the following sections to troubleshoot repository mirroring for Red Hat Quay.

6.5.1.1. Verifying authentication and permissions

Ensure that the authentication credentials used for mirroring have the necessary permissions and access rights on both the source and destination Red Hat Quay instances.

On the Red Hat Quay UI, check the following settings:

- The access control settings. Ensure that the user or service account performing the mirroring operation has the required privileges.
- The permissions of your robot account on the Red Hat Quay registry.

6.6. CLAIR SECURITY SCANNER

6.6.1. Troubleshooting Clair issue

Use the following procedures to troubleshoot Clair.

6.6.1.1. Verifying image compatibility

If you are using Clair, ensure that the images you are trying to scan are supported by Clair. Clair has certain requirements and does not support all image formats or configurations.

For more information, see Clair vulnerability databases.

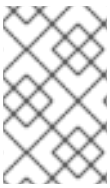
6.6.1.2. Allowlisting Clair updaters

If you are using Clair behind a proxy configuration, you must allowlist the updaters in your proxy or firewall configuration. For more information about updater URLs, see Clair updater URLs .

6.6.1.3. Updating Clair scanner and its dependencies

Ensure that you are using the latest version of Clair security scanner. Outdated versions might lack support for newer image formats, or might have known issues.

Use the following procedure to check your version of Clair.



NOTE

Checking Clair logs can also be used to check if there are any errors from the updaters microservice in your Clair logs. By default, Clair updates the vulnerability database every 30 minutes.

Procedure

1. Check your version of Clair.
   a. If you are running Clair on Red Hat Quay on OpenShift Container Platform, enter the following command:

```
$ oc logs clair-pod
```

   b. If you are running a standalone deployment of Red Hat Quay and using a Clair container, enter the following command:

```
$ podman logs clair-container
```

-

Example output

```
"level":"info",
"component":"main",
"version":"v4.5.1",
```

6.6.1.4. Enabling debug mode for Clair

By default, debug mode for Clair is disabled. You can enable debug mode for Clair by updating your **clair-config.yaml** file.

Prerequisites

- For Clair on Red Hat Quay on OpenShift Container Platform deployments, you must Running a custom Clair configuration with a managed Clair database.

Use the following procedure to enable debug mode for Clair.

Procedure

1. Update your **clair-config.yaml** file to include the debug option.
   a. On standalone Red Hat Quay deployments:
      i. Add the following configuration field to your **clair-config.yaml** file:

      ```
      log_level: debug
      ```

      ii. Restart your Clair deployment by entering the following command:

      ```
      $ podman restart <clair_container_name>
      ```

   b. On Red Hat Quay on OpenShift Container Platform deployments:
      i. On the OpenShift Container Platform web console, click **Operators → Installed Operators → Quay Registry**.
      ii. Click the name of your registry, for example, **Example Registry**. You are redirected to the **Details** page of your registry.
      iii. Click the Config Bundle Secret, for example, **example-registry-config-bundle-xncls**.
      iv. Confirm that you are running a custom Clair configuration by looking for the **clair-config.yaml** file under the **Data** section of the **Details** page of your secret.
      v. If you have a **clair-config.yaml** file, click **Actions → Edit Secret**. If you do not, see "Running a custom Clair configuration with a managed Clair database".
      vi. Update your **clair-config.yaml** file to include the **log_level: debug** configuration variable. For example:

      ```
      log_level: debug
      ```

      vii. Click **Save**.

viii. You can check the status of your Clair deployment by clicking **Workloads → Pods**. The **clair-app** pod should report **1/1** under the **Ready** category.

ix. You can confirm that Clair is returning debugging information by clicking the **clair-app** pod that is ready → **Logs**.

6.6.1.5. Checking Clair configuration

Check your Clair **config.yaml** file to ensure that there are no misconfigurations or inconsistencies that could lead to issues. For more information, see Clair configuration overview .

6.6.1.6. Inspect image metadata

In some cases, you might receive an **Unsupported** message. This might indicate that the scanner is unable to extract the necessary metadata from the image. Check if the image metadata is properly formatted and accessible.

Additional resources

For more information, see Troubleshooting Clair.