Red Hat Quay 3.13

Builders and image automation

Builders and image automation

Last Updated: 2024-11-25

Red Hat Quay 3.13 Builders and image automation

Builders and image automation

Legal Notice

Abstract

Understand builders and their role in automating image builds.

Table of Contents

PREFACE

The following guide shows you how to configure the Red Hat Quay *builds* feature on both bare metal and virtual machines.

CHAPTER 1. RED HAT QUAY BUILDS OVERVIEW

Red Hat Quay builds , or just *builds*, are a feature that enable the automation of container image builds. The *builds* feature uses worker nodes to build images from Dockerfiles or other build specifications. These builds can be triggered manually or automatically via webhooks from repositories like GitHub, allowing users to integrate continuous integration (CI) and continuous delivery (CD) pipelines into their workflow.

The *builds* feature is supported on Red Hat Quay on OpenShift Container Platform and Kubernetes clusters. For Operator-based deployments and Kubernetes clusters, *builds* are created by using a *build manager* that coordinates and handles the build jobs. *Builds* support building Dockerfile on both bare metal platforms and on virtualized platforms with *virtual builders*. This versatility allows organizations to adapt to existing infrastructure while leveraging Red Hat Quay's container image build capabilities.

The key features of *Red Hat Quay builds* feature include:

- Automated builds triggered by code commits or version control events
- Support for Docker and Podman container images
- Fine-grained control over build environments and resources
- Integration with Kubernetes and OpenShift Container Platform for scalable builds
- Compatibility with bare metal and virtualized infrastructure

NOTE

Running *builds* directly in a container on bare metal platforms does not have the same isolation as when using virtual machines, however, it still provides good protection.

Builds are highly complex, and administrators are encouraged to review the Build automation architecture guide before continuing.

1.1. BUILDING CONTAINER IMAGES

Building container images involves creating a blueprint for a containerized application. Blueprints rely on base images from other public repositories that define how the application should be installed and configured.

Red Hat Quay supports the ability to build Docker and Podman container images. This functionality is valuable for developers and organizations who rely on container and container orchestration.

1.1.1. Build contexts

When building an image with Docker or Podman, a directory is specified to become the *build context*. This is true for both manual Builds and Build triggers, because the Build that is created by is not different than running **docker build** or **podman build** on your local machine.

Build contexts are always specified in the *subdirectory* from the Build setup, and fallback to the root of the Build source if a directory is not specified.

When a build is triggered, Build workers clone the Git repository to the worker machine, and then enter the Build context before conducting a Build.

For Builds based on **.tar** archives, Build workers extract the archive and enter the Build context. For example:

Extracted Build archive

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

Imagine that the *Extracted Build archive* is the directory structure got a Github repository called **example.** If no subdirectory is specified in the Build trigger setup, or when manually starting the Build, the Build operates in the example directory.

If a subdirectory is specified in the Build trigger setup, for example, **subdir**, only the Dockerfile within it is visible to the Build. This means that you cannot use the **ADD** command in the Dockerfile to add **file**, because it is outside of the Build context.

Unlike Docker Hub, the Dockerfile is part of the Build context on As a result, it must not appear in the **.dockerignore** file.

CHAPTER 2. CONFIGURING THE OPENSHIFT CONTAINER PLATFORM TLS COMPONENT FOR BUILDS

The **tls** component of the **QuayRegistry** custom resource definition (CRD) allows you to control whether SSL/TLS are managed by the Red Hat Quay Operator, or self managed. In its current state, Red Hat Quay does not support the *builds* feature, or the *builder* workers, when the **tls** component is managed by the Red Hat Quay Operator.

When setting the **tls** component to **unmanaged**, you must supply your own **ssl.cert** and **ssl.key** files. Additionally, if you want your cluster to support *builders*, or the worker nodes that are responsible for building images, you must add both the **Quay** route and the **builder** route name to the SAN list in the certificate. Alternatively, however, you could use a wildcard.

The following procedure shows you how to add the *builder* route.

Prerequisites

- You have set the **tls** component to **unmanaged** and uploaded custom SSL/TLS certificates to the Red Hat Quay Operator. For more information, see SSL and TLS for Red Hat Quay.

Procedure

- In the configuration file that defines your SSL/TLS certificate parameters, for example, **openssl.cnf**, add the following information to the certificate's Subject Alternative Name (SAN) field. For example:

```
# ...
[alt_names]
<quayregistry-name>-quay-builder-<namespace>.<domain-name>:443
# ...
```

For example:

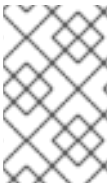
```
# ...
[alt_names]
example-registry-quay-builder-quay-enterprise.apps.cluster-new.gcp.quaydev.org:443
# ...
```

CHAPTER 3. BARE METAL BUILDS WITH RED HAT QUAY ON OPENSHIFT CONTAINER PLATFORM

The procedures in this section explain how to create an environment for *bare metal builds* for Red Hat Quay on OpenShift Container Platform.

3.1. CONFIGURING BARE METAL BUILDS FOR RED HAT QUAY ON OPENSHIFT CONTAINER PLATFORM

Use the following procedure to configure *bare metal builds* for Red Hat Quay on OpenShift Container Platform.



NOTE

If you are using the Red Hat Quay Operator on OpenShift Container Platform with a managed **route** component in your **QuayRegistry** CRD, see "Red Hat Quay on OpenShift Container Platform *builds* limitations with self-managed *routes*".

Prerequisites

- You have an OpenShift Container Platform cluster provisioned with the Red Hat Quay Operator running.
- You have set the **tls** component to **unmanaged** and uploaded custom SSL/TLS certificates to the Red Hat Quay Operator. For more information, see SSL and TLS for Red Hat Quay.
- You are logged into OpenShift Container Platform as a cluster administrator.

Procedure

1. Enter the following command to create a project where Builds will be run, for example, **bare-metal-builder**:

```
$ oc new-project bare-metal-builder
```

2. Create a new **ServiceAccount** in the the **bare-metal-builder** namespace by entering the following command:

```
$ oc create sa -n bare-metal-builder quay-builder
```

3. Enter the following command to grant a user the **edit** role within the **bare-metal-builder** namespace:

```
$ oc policy add-role-to-user -n bare-metal-builder edit system:serviceaccount:bare-metal-builder:quay-builder
```

4. Enter the following command to retrieve a token associated with the **quay-builder** service account in the **bare-metal-builder** namespace. This token is used to authenticate and interact with the OpenShift Container Platform cluster's API server.

   a. If your OpenShift Container Platform cluster is version 4.11+, enter the following command:

```
oc create token quay-builder  -n bare-metal-builder --duration 24h
```

b. If your OpenShift Container Platform cluster is earlier than version 4.11, for example, version 4.10, enter the following command:

```
$ oc sa get-token -n bare-metal-builder quay-builder
```

5. Identify the URL for the OpenShift Container Platform cluster's API server. This can be found in the OpenShift Container Platform web console.

6. Identify a worker node label to be used when scheduling *build jobs*. Because *build pods* must run on bare metal worker nodes, typically these are identified with specific labels.
Check with your cluster administrator to determine exactly which node label should be used.

7. Obtain the Kube API Server's certificate authority (CA) to add to Red Hat Quay's extra certificates.

a. On OpenShift Container Platform versions 4.15+, enter the following commands to obtain the name of the secret containing the CA:

```
$ oc extract cm/kube-root-ca.crt -n openshift-apiserver
```

```
$ mv ca.crt build_cluster.crt
```

b. On OpenShift Container Platform versions earlier than 4.15, for example, 4.14, enter the following command:

```
$ oc get sa openshift-apiserver-sa --namespace=openshift-apiserver -o json | jq
'.secrets[] | select(.name | contains("openshift-apiserver-sa-token"))'.name
```

c. Obtain the **ca.crt** key value from the secret in the OpenShift Container Platform Web Console. The value begins with **"-----BEGIN CERTIFICATE-----"`**.

d. Import the CA to Red Hat Quay. Ensure that the name of this file matches the **K8S_API_TLS_CA** field used in Step 9.

8. Create the following **SecurityContextConstraints** resource for the **ServiceAccount**:

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: quay-builder
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities: null
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: RunAsAny
seccompProfiles:
- '*'
supplementalGroups:
  type: RunAsAny
volumes:
```

```
- '*'
allowHostDirVolumePlugin: true
allowHostIPC: true
allowHostNetwork: true
allowHostPID: true
allowHostPorts: true
allowPrivilegeEscalation: true
allowPrivilegedContainer: true
allowedCapabilities:
- '*'
allowedUnsafeSysctls:
- '*'
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: quay-builder-scc
  namespace: bare-metal-builder
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - quay-builder
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: quay-builder-scc
  namespace: bare-metal-builder
subjects:
- kind: ServiceAccount
  name: quay-builder
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: quay-builder-scc
```

9. Update the **config.yaml** file of your Red Hat Quay on OpenShift Container Platform deployment to include an appropriate *bare metal builds* configuration by using the OpenShift Container Platform web console.

   a. Click **Operators → Installed Operators → Red Hat Quay → Quay Registry**.

   b. Click the name of your registry, for example, **example-registry**.

   c. Under **Config Bundle Secret**, click the name of your configuration bundle, for example, **extra-ca-certificate-config-bundle-secret**.

   d. Click **Actions → Edit Secret**.

e. Add the following information to your Red Hat Quay **config.yaml** file, replacing each value with information that is relevant to your specific installation:

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- <superusername>
FEATURE_USER_CREATION: false
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_BUILD_SUPPORT: True
BUILDMAN_HOSTNAME: ${BUILDMAN_HOSTNAME}:443 (1)
BUILD_MANAGER:
- ephemeral
- ALLOWED_WORKER_COUNT: 10
  ORCHESTRATOR_PREFIX: buildman/production/
    ORCHESTRATOR:
      REDIS_HOST: <sample_redis_hostname> (2)
      REDIS_PASSWORD: ""
      REDIS_SSL: false
      REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
  EXECUTORS:
  - EXECUTOR: kubernetes
    BUILDER_NAMESPACE: <sample_builder_namespace> (3)
    K8S_API_SERVER: <sample_k8s_api_server> (4)
    K8S_API_TLS_CA: <sample_crt_file> (5)
    VOLUME_SIZE: 8G
    KUBERNETES_DISTRIBUTION: openshift
    CONTAINER_MEMORY_LIMITS: 1G (6)
    CONTAINER_CPU_LIMITS: 300m (7)
    CONTAINER_MEMORY_REQUEST: 1G (8)
    CONTAINER_CPU_REQUEST: 300m (9)
    NODE_SELECTOR_LABEL_KEY: beta.kubernetes.io/instance-type
    NODE_SELECTOR_LABEL_VALUE: n1-standard-4
    CONTAINER_RUNTIME: podman
    SERVICE_ACCOUNT_NAME: <sample_service_account_name>
    SERVICE_ACCOUNT_TOKEN: <sample_account_token> (10)
    QUAY_USERNAME: <quay_username>
    QUAY_PASSWORD: <quay_password>
    WORKER_IMAGE: <registry>/quay-quay-builder
    WORKER_TAG: <some_tag>
    BUILDER_VM_CONTAINER_IMAGE: quay.io/quay/quay-builder-qemu-
fedoracoreos:latest
    SETUP_TIME: 180
    MINIMUM_RETRY_THRESHOLD: 0
    SSH_AUTHORIZED_KEYS: (11)
    - <ssh-rsa 12345 someuser@email.com>
    - <ssh-rsa 67890 someuser2@email.com>
    HTTP_PROXY: <http://10.0.0.1:80>
    HTTPS_PROXY: <http://10.0.0.1:80>
    NO_PROXY: <hostname.example.com>
```

(1) Obtained by running the following command: **$ oc get route quayregistry-quay-builder -n ${QUAY_PROJECT} -o jsonpath='{.spec.host}'**.

2. The hostname for your Redis service.
3. Set to match the name of your *bare metal builds* namespace. This example used **bare-metal-builder**.
4. The **K8S_API_SERVER** is obtained by running **$ oc cluster-info**.
5. You must manually create and add your custom CA cert, for example, **K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build-cluster.crt**.
6. Defaults to **5120Mi** if left unspecified.
7. Defaults to **1000m** if left unspecified.
8. Defaults to **3968Mi** if left unspecified.
9. Defaults to **500m** if left unspecified.
10. Obtained when running **$ oc create sa**.
11. Allows public SSH keys to be added to the build environment for remote troubleshooting access. This key, or keys, should correspond to the private key that an admin or developer will use to SSH into the build worker for debugging purposes. This key can be obtained by establishing an SSH connection to the remote host using a specific SSH key and port. For example: **$ ssh -i /path/to/ssh/key/set/in/ssh_authorized_keys -p 9999 core@localhost**.

10. Restart your Red Hat Quay registry to enable the *builds* feature.

3.1.1. Red Hat Quay on OpenShift Container Platform *builds* limitations with self-managed *routes*

The following limitations apply when you are using the Red Hat Quay Operator on OpenShift Container Platform with a managed **route** component:

- Currently, OpenShift Container Platform *routes* are only able to serve traffic to a single port. Additional steps are required to set up Red Hat Quay Builds.
- Ensure that your **kubectl** or **oc** CLI tool is configured to work with the cluster where the Red Hat Quay Operator is installed and that your **QuayRegistry** exists; the **QuayRegistry** does not have to be on the same bare metal cluster where *builders* run.
- Ensure that HTTP/2 ingress is enabled on the OpenShift cluster by following these steps.
- The Red Hat Quay Operator creates a **Route** resource that directs gRPC traffic to the Build manager server running inside of the existing **Quay** pod, or pods. If you want to use a custom hostname, or a subdomain like **<builder-registry.example.com>**, ensure that you create a CNAME record with your DNS provider that points to the **status.ingress[0].host** of the create **Route** resource. For example:

```
$ kubectl get -n <namespace> route <quayregistry-name>-quay-builder -o jsonpath=
{.status.ingress[0].host}
```

- Using the OpenShift Container Platform UI or CLI, update the **Secret** referenced by **spec.configBundleSecret** of the **QuayRegistry** with the *build* cluster CA certificate. Name the

key **extra_ca_cert_build_cluster.cert**. Update the **config.yaml** file entry with the correct values referenced in the *build* configuration that you created when you configured Red Hat Quay *builds*, and add the **BUILDMAN_HOSTNAME** CONFIGURATION FIELD:

```
BUILDMAN_HOSTNAME: <build-manager-hostname> 1
BUILD_MANAGER:
- ephemeral
- ALLOWED_WORKER_COUNT: 1
  ORCHESTRATOR_PREFIX: buildman/production/
  JOB_REGISTRATION_TIMEOUT: 600
  ORCHESTRATOR:
    REDIS_HOST: <quay_redis_host
    REDIS_PASSWORD: <quay_redis_password>
    REDIS_SSL: true
    REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
  EXECUTORS:
  - EXECUTOR: kubernetes
    BUILDER_NAMESPACE: builder
    ...
```

1 The externally accessible server hostname which the *build jobs* use to communicate back to the *build manager*. Default is the same as **SERVER_HOSTNAME**. For an OpenShift **route** resource, it is either **status.ingress[0].host** or the CNAME entry if using a custom hostname. **BUILDMAN_HOSTNAME** must include the port number, for example, **somehost:443** for an OpenShift Container Platform **route** resource, as the gRPC client used to communicate with the *build manager* does not infer any port if omitted.

CHAPTER 4. VIRTUAL BUILDS WITH RED HAT QUAY ON OPENSHIFT CONTAINER PLATFORM

The procedures in this section explain how to create an environment for *bare metal builds* for Red Hat Quay on OpenShift Container Platform.

Virtual builds can be run on virtualized machines with Red Hat Quay on OpenShift Container Platform. With this method, the *build manager* first creates the **Job Object** resource. Then, the **Job Object** creates a pod using the **quay-builder-image**. The **quay-builder-image** contains the **quay-builder** binary and the Podman service. The created pod runs as **unprivileged**. The **quay-builder** binary then builds the image while communicating status and retrieving build information from the *build manager*.

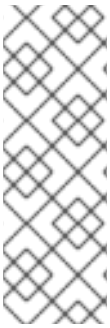
4.1. VIRTUAL BUILDS LIMITATIONS

The following limitations apply to the *virtual builds* feature:

- Running *virtual builds* with Red Hat Quay on OpenShift Container Platform in an unprivileged context might cause some commands that were working under the previous build strategy to fail. Attempts to change the build strategy could potentially cause performance issues and reliability with the build.
- Running *virtual builds* directly in a container does not have the same isolation as using virtual machines. Changing the build environment might also cause builds that were previously working to fail.

4.2. CONFIGURING VIRTUAL BUILDS FOR RED HAT QUAY ON OPENSHIFT CONTAINER PLATFORM

The procedures in this section explain how to create an environment for *virtual builds* for Red Hat Quay on OpenShift Container Platform.



NOTE

- If you are using Amazon Web Service (AWS) S3 storage, you must modify your storage bucket in the AWS console, prior to running builders. See "Modifying your AWS S3 storage bucket" in the following section for the required parameters.
- If you are using a Google Cloud Platform (GCP) object bucket, you must configure cross-origin resource sharing (CORS) to enable *virtual builds*.

Prerequisites

- You have an OpenShift Container Platform cluster provisioned with the Red Hat Quay Operator running.
- You have set the **tls** component to **unmanaged** and uploaded custom SSL/TLS certificates to the Red Hat Quay Operator. For more information, see SSL and TLS for Red Hat Quay.
- You have configured the OpenShift Container Platform TLS component for builds.
- You are logged into OpenShift Container Platform as a cluster administrator.

Procedure

1. Create a new project where your virtual builders will be run, for example, **virtual-builders**, by running the following command:

```
$ oc new-project virtual-builders
```

2. Create a **ServiceAccount** in the project that will be used to run *builds* by entering the following command:

```
$ oc create sa -n virtual-builders quay-builder
```

Example output

```
serviceaccount/quay-builder created
```

3. Provide the created service account with editing permissions so that it can run a *build*:

```
$ oc adm policy -n virtual-builders add-role-to-user edit system:serviceaccount:virtual-builders:quay-builder
```

Example output

```
clusterrole.rbac.authorization.k8s.io/edit added: "system:serviceaccount:virtual-builders:quay-builder"
```

4. Grant the *builder* worker **anyuid scc** permissions by entering the following command. This requires cluster administrator privileges, which is required because *builders* must run as the Podman user for unprivileged or rootless builds to work.

```
$ oc adm policy -n virtual-builders add-scc-to-user anyuid -z quay-builder
```

Example output

```
clusterrole.rbac.authorization.k8s.io/system:openshift:scc:anyuid added: "quay-builder"
```

5. Obtain the token for the *builder* service account by entering the following command:

```
$ oc create token quay-builder -n virtual-builders
```

NOTE

When the token expires you will need to request a new token. Optionally, you can also add a custom expiration. For example, specify **--duration 20160m** to retain the token for two weeks.

Example output

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ltTHZ0dGZMYjhIWnYxZTQzN2dJVEJxcDJscldSdEUtYWsifQ...
```

6. Determine the *builder* route by entering the following command:

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

Example output

```
NAME: example-registry-quay-builder
HOST/PORT: example-registry-quay-builder-quay-enterprise.apps.stevsmit-cluster-new.gcp.quaydev.org
PATH:
SERVICES: example-registry-quay-app
PORT: grpc
TERMINATION: passthrough/Redirect
WILDCARD: None
```

7. Generate a self-signed SSL/TLS certificate with the **.crt** extension by entering the following command:

```
$ oc extract cm/kube-root-ca.crt -n openshift-apiserver
```

Example output

```
ca.crt
```

8. Rename the **ca.crt** file to **build-cluster.crt** by entering the following command:

```
$ mv ca.crt build-cluster.crt
```

9. Update the **config.yaml** file of your Red Hat Quay on OpenShift Container Platform deployment to include an appropriate *virtual builds* configuration by using the OpenShift Container Platform web console.

   a. Click **Operators → Installed Operators → Red Hat Quay → Quay Registry**.

   b. Click the name of your registry, for example, **example-registry**.

   c. Under **Config Bundle Secret**, click the name of your configuration bundle, for example, **extra-ca-certificate-config-bundle-secret**.

   d. Click **Actions → Edit Secret**.

   e. Add an appropriate *virtual builds* configuration using the following as a reference:

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- <superusername>
FEATURE_USER_CREATION: false
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_BUILD_SUPPORT: True
BUILDMAN_HOSTNAME: <sample_build_route> 1
BUILD_MANAGER:
  - ephemeral
  - ALLOWED_WORKER_COUNT: 1
```

```
ORCHESTRATOR_PREFIX: buildman/production/
JOB_REGISTRATION_TIMEOUT: 3600 (2)
ORCHESTRATOR:
  REDIS_HOST: <sample_redis_hostname> (3)
  REDIS_PASSWORD: ""
  REDIS_SSL: false
  REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
EXECUTORS:
  - EXECUTOR: kubernetesPodman
    NAME: openshift
    BUILDER_NAMESPACE: <sample_builder_namespace> (4)
    SETUP_TIME: 180
    MINIMUM_RETRY_THRESHOLD: 0
    BUILDER_CONTAINER_IMAGE: quay.io/projectquay/quay-builder:{producty}
    # Kubernetes resource options
    K8S_API_SERVER: <sample_k8s_api_server> (5)
    K8S_API_TLS_CA: <sample_crt_file> (6)
    VOLUME_SIZE: 8G
    KUBERNETES_DISTRIBUTION: openshift
    CONTAINER_MEMORY_LIMITS: 1G (7)
    CONTAINER_CPU_LIMITS: 300m (8)
    CONTAINER_MEMORY_REQUEST: 1G (9)
    CONTAINER_CPU_REQUEST: 300m (10)
    NODE_SELECTOR_LABEL_KEY: ""
    NODE_SELECTOR_LABEL_VALUE: ""
    SERVICE_ACCOUNT_NAME: <sample_service_account_name>
    SERVICE_ACCOUNT_TOKEN: <sample_account_token> (11)
    HTTP_PROXY: <http://10.0.0.1:80>
    HTTPS_PROXY: <http://10.0.0.1:80>
    NO_PROXY: <hostname.example.com>
```

1. The build route is obtained by running **$ oc get route -n** with the namespace of your Red Hat Quay on OpenShift Container Platform deployment. A port must be provided at the end of the route, and it should use the following format: **[quayregistry-cr-name]-quay-builder-[ocp-namespace].[ocp-domain-name]:443**.

2. If the **JOB_REGISTRATION_TIMEOUT** parameter is set too low, you might receive the following error: **failed to register job to build manager: rpc error: code = Unauthenticated desc = Invalid build token: Signature has expired**. This parameter should be set to at least **240**.

3. If your Redis host has a password or SSL/TLS certificates, you must update this field accordingly.

4. Set to match the name of your *virtual builds* namespace. This example used **virtual-builders**.

5. The **K8S_API_SERVER** is obtained by running **$ oc cluster-info**.

6. You must manually create and add your custom CA cert, for example, **K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build-cluster.crt**.

7. Defaults to **5120Mi** if left unspecified.

8.

For *virtual builds*, you must ensure that there are enough resources in your cluster. Defaults to **1000m** if left unspecified.

9 Defaults to **3968Mi** if left unspecified.

10 Defaults to **500m** if left unspecified.

11 Obtained when running **$ oc create sa**.

Example *virtual builds* configuration

```
FEATURE_USER_INITIALIZE: true
BROWSER_API_CALLS_XHR_ONLY: false
SUPER_USERS:
- quayadmin
FEATURE_USER_CREATION: false
FEATURE_QUOTA_MANAGEMENT: true
FEATURE_BUILD_SUPPORT: True
BUILDMAN_HOSTNAME: example-registry-quay-builder-quay-
enterprise.apps.docs.quayteam.org:443
BUILD_MANAGER:
  - ephemeral
  - ALLOWED_WORKER_COUNT: 1
    ORCHESTRATOR_PREFIX: buildman/production/
    JOB_REGISTRATION_TIMEOUT: 3600
    ORCHESTRATOR:
      REDIS_HOST: example-registry-quay-redis
      REDIS_PASSWORD: ""
      REDIS_SSL: false
      REDIS_SKIP_KEYSPACE_EVENT_SETUP: false
    EXECUTORS:
      - EXECUTOR: kubernetesPodman
        NAME: openshift
        BUILDER_NAMESPACE: virtual-builders
        SETUP_TIME: 180
        MINIMUM_RETRY_THRESHOLD: 0
        BUILDER_CONTAINER_IMAGE: quay.io/projectquay/quay-builder:{producty}
        # Kubernetes resource options
        K8S_API_SERVER: api.docs.quayteam.org:6443
        K8S_API_TLS_CA: /conf/stack/extra_ca_certs/build-cluster.crt
        VOLUME_SIZE: 8G
        KUBERNETES_DISTRIBUTION: openshift
        CONTAINER_MEMORY_LIMITS: 1G
        CONTAINER_CPU_LIMITS: 300m
        CONTAINER_MEMORY_REQUEST: 1G
        CONTAINER_CPU_REQUEST: 300m
        NODE_SELECTOR_LABEL_KEY: ""
        NODE_SELECTOR_LABEL_VALUE: ""
        SERVICE_ACCOUNT_NAME: quay-builder
        SERVICE_ACCOUNT_TOKEN:
"eyJhbGciOiJSUzI1NiIsImtpZCI6IldfQUJkaDVmb3ltTHZ0dGZMYjhIWnYxZTQzN2dJVEJxc
DJscldSdEUtYWsifQ"
        HTTP_PROXY: <http://10.0.0.1:80>
        HTTPS_PROXY: <http://10.0.0.1:80>
        NO_PROXY: <hostname.example.com>
```

f. Click **Save** on the **Edit Secret** page.

10. Restart your Red Hat Quay on OpenShift Container Platform registry with the new configuration.

4.2.1. Modifying your AWS S3 storage bucket

If you are using AWS S3 storage, you must change your storage bucket in the AWS console prior to starting a *build*.

Procedure

1. Log in to your AWS console at s3.console.aws.com.
2. In the search bar, search for **S3** and then click **S3**.
3. Click the name of your bucket, for example, **myawsbucket**.
4. Click the **Permissions** tab.
5. Under **Cross-origin resource sharing (CORS)** include the following parameters:

```
[
    {
        "AllowedHeaders": [
            "Authorization"
        ],
        "AllowedMethods": [
            "GET"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [],
        "MaxAgeSeconds": 3000
    },
    {
        "AllowedHeaders": [
            "Content-Type",
            "x-amz-acl",
            "origin"
        ],
        "AllowedMethods": [
            "PUT"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [],
        "MaxAgeSeconds": 3000
    }
]
```

4.2.2. Modifying your Google Cloud Platform object bucket

NOTE

Currently, modifying your Google Cloud Platform object bucket is not supported on IBM Power and IBM Z.

Use the following procedure to configure cross-origin resource sharing (CORS) for virtual builders. Without CORS configuration, uploading a build Dockerfile fails.

Procedure

1. Use the following reference to create a JSON file for your specific CORS needs. For example:

```
$ cat gcp_cors.json
```

Example output

```
[
   {
     "origin": ["*"],
     "method": ["GET"],
     "responseHeader": ["Authorization"],
     "maxAgeSeconds": 3600
   },
   {
     "origin": ["*"],
     "method": ["PUT"],
     "responseHeader": [
            "Content-Type",
            "x-goog-acl",
            "origin"],
     "maxAgeSeconds": 3600
   }
]
```

2. Enter the following command to update your GCP storage bucket:

```
$ gcloud storage buckets update gs://<bucket_name> --cors-file=./gcp_cors.json
```

Example output

```
Updating
  Completed 1
```

3. You can display the updated CORS configuration of your GCP bucket by running the following command:

```
$ gcloud storage buckets describe gs://<bucket_name>  --format="default(cors)"
```

Example output

```
cors:
- maxAgeSeconds: 3600
  method:
```

```
  - GET
  origin:
  - '*'
  responseHeader:
  - Authorization
- maxAgeSeconds: 3600
  method:
  - PUT
  origin:
  - '*'
  responseHeader:
  - Content-Type
  - x-goog-acl
  - origin
```

CHAPTER 5. STARTING A NEW BUILD

After you have enabled the Red Hat Quay *builds* feature by configuring your deployment, you can start a new build by invoking a *build trigger* or by uploading a Dockerfile.

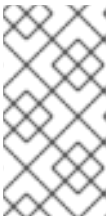
Use the following procedure to start a new *build* by uploading a Dockerfile. For information about creating a *build trigger*, see "Build triggers".

Prerequisites

- You have navigated to the **Builds** page of your repository.
- You have configured your environment to use the *build* feature.

Procedure

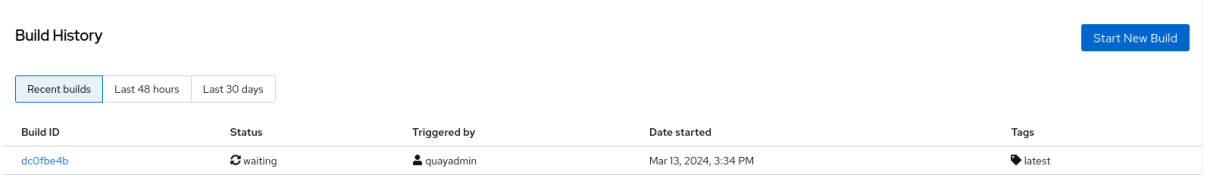
1. On the **Builds** page, click **Start New Build**.
2. When prompted, click **Upload Dockerfile** to upload a Dockerfile or an archive that contains a Dockerfile at the root directory.
3. Click **Start Build**.



NOTE

- Currently, users cannot specify the Docker build context when manually starting a build.
- Currently, BitBucket is unsupported on the Red Hat Quay v2 UI.

4. You are redirected to the *build*, which can be viewed in real-time. Wait for the Dockerfile *build* to be completed and pushed.
5. Optional. you can click **Download Logs** to download the logs, or **Copy Logs** to copy the logs.
6. Click the back button to return to the **Repository Builds** page, where you can view the *build history*.



7. You can check the status of your *build* by clicking the commit in the **Build History** page, or by running the following command:

```
$ oc get pods -n virtual-builders
```

Example output

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

8. After the *build* has completed, the **oc get pods -n virtual-builders** command returns no resources:

```
$ oc get pods -n virtual-builders
```

Example output

```
No resources found in virtual-builders namespace.
```

CHAPTER 6. BUILD TRIGGERS

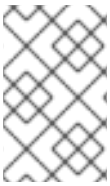
Build triggers are automated mechanisms that start a container image build when specific conditions are met, such as changes to source code, updates to dependencies, or creating a webhook call. These triggers help automate the image-building process and ensure that the container images are always up-to-date without manual intervention.

The following sections cover content related to creating a build trigger, tag naming conventions, how to skip a source control-triggered build, starting a *build*, or manually triggering a *build*.

6.1. CREATING A BUILD TRIGGER

The following procedure sets up a *custom Git trigger*. A custom Git trigger is a generic way for any Git server to act as a *build trigger*. It relies solely on SSH keys and webhook endpoints. Creating a custom Git trigger is similar to the creation of any other trigger, with the exception of the following:

These steps can be replicated to create a *build trigger* using Github, Gitlab, or Bitbucket, however, you must configure the credentials for these services in your **config.yaml** file.



NOTE

- If you want to use Github to create a *build trigger*, you must configure Github to be used with Red Hat Quay by creating an OAuth application. For more information, see "Creating an OAuth application Github".

Prerequisites

- For Red Hat Quay on OpenShift Container Platform deployments, you have configured your OpenShift Container Platform environment for either bare metal builds or virtual builds.

Procedure

1. Log in to your Red Hat Quay registry.
2. In the navigation pane, click **Repositories**.
3. Click **Create Repository**.
4. Click the **Builds** tab.
5. On the **Builds** page, click **Create Build Trigger**.
6. Select the desired platform, for example, **Github**, **Bitbucket**, **Gitlab**, or use a custom Git repository. For this example, click **Custom Git Repository Push**
7. Enter a custom Git repository name, for example, **git@github.com:<username>/<repo>.git**. Then, click **Next**.
8. When prompted, configure the tagging options by selecting one of, or both of, the following options:
    - **Tag manifest with the branch or tag name** When selecting this option, the built manifest the name of the branch or tag for the git commit are tagged.

- **Add latest tag if on default branch** When selecting this option, the built manifest with latest if the build occurred on the default branch for the repository are tagged. Optionally, you can add a custom tagging template. There are multiple tag templates that you can enter here, including using short SHA IDs, timestamps, author names, committer, and branch names from the commit as tags. For more information, see "Tag naming for build triggers".

  After you have configured tagging, click **Next**.

9. When prompted, select the location of the Dockerfile to be built when the trigger is invoked. If the Dockerfile is located at the root of the git repository and named Dockerfile, enter **/Dockerfile** as the Dockerfile path. Then, click **Next**.

10. When prompted, select the context for the Docker build. If the Dockerfile is located at the root of the Git repository, enter / as the build context directory. Then, click **Next**.

11. Optional. Choose an optional robot account. This allows you to pull a private base image during the build process. If you know that a private base image is not used, you can skip this step.

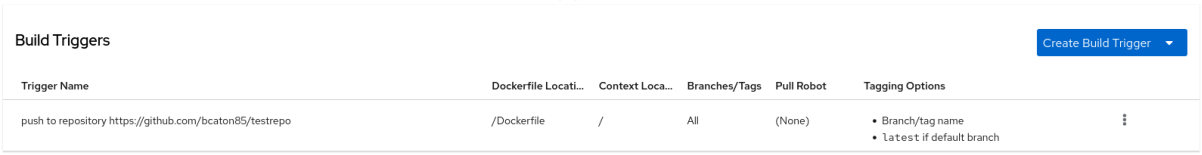
12. Click **Next**. Check for any verification warnings. If necessary, fix the issues before clicking **Finish**.

13. You are alerted that the trigger has been successfully activated. Note that using this trigger requires the following actions:

    - You must give the following public key read access to the git repository.

    - You must set your repository to **POST** to the following URL to trigger a build.
      Save the SSH Public Key, then click **Return to <organization_name>/<repository_name>**.
      You are redirected to the **Builds** page of your repository.

14. On the **Builds** page, you now have a *build trigger*. For example:



After you have created a custom Git trigger, additional steps are required. Continue on to "Setting up a custom Git trigger".

If you are setting up a *build trigger* for Github, Gitlab, or Bitbucket, continue on to "Manually triggering a build".

6.1.1. Setting up a custom Git trigger

After you have created a *custom Git trigger*, two additional steps are required:

1. You must provide read access to the SSH public key that is generated when creating the trigger.

2. You must setup a webhook that POSTs to the Red Hat Quay endpoint to trigger the build.

These steps are only required if you are using a *custom Git trigger*.

6.1.1.1. Obtaining build trigger credentials

The SSH public key and Webhook Endpoint URL are available on the Red Hat Quay UI.

Prerequisites

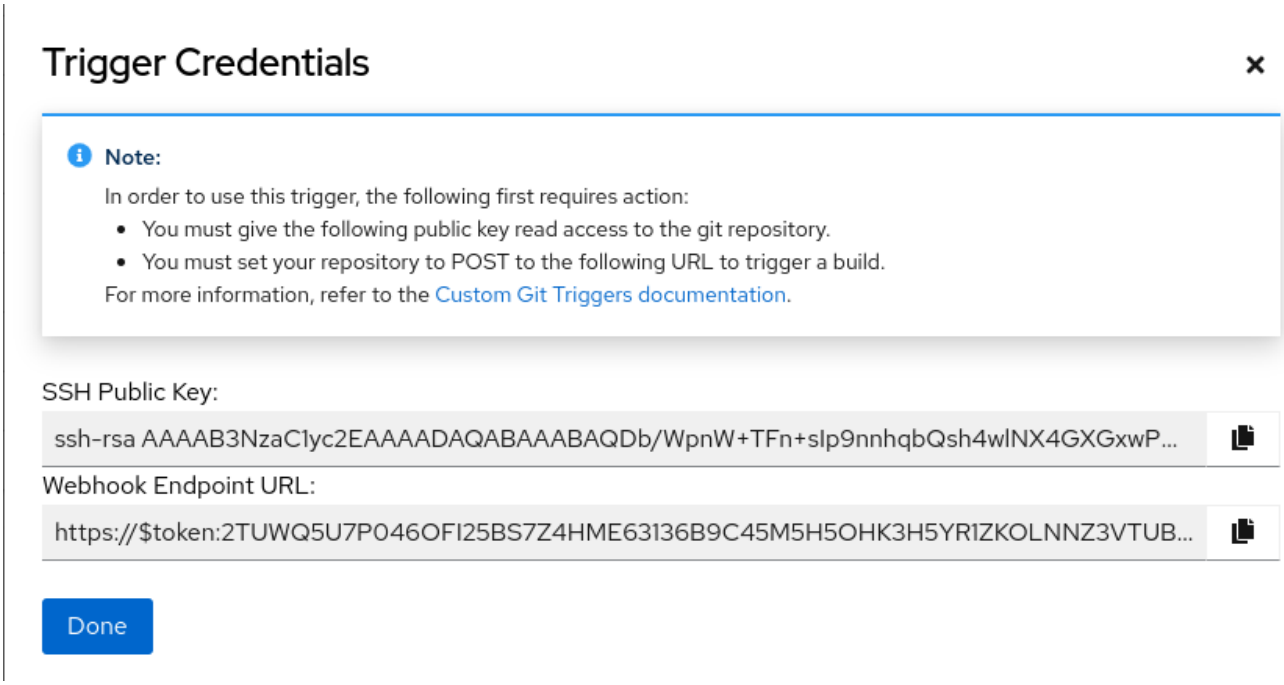
- You have created a *custom Git trigger*.

Procedure

1. On the **Builds** page of your repository, click the menu kebab for your *custom Git trigger*.
2. Click **View Credentials**.
3. Save the SSH Public Key and Webhook Endpoint URL.

The key and the URL are available by selecting **View Credentials** from the **Settings**, or *gear* icon.

View and modify tags from your repository



6.1.1.1.1. SSH public key access

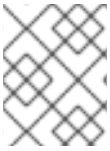
Depending on the Git server configuration, there are multiple ways to install the SSH public key that generates for a custom Git trigger.

For example, documentation for Getting Git on a Server describes a describes how to set up a Git server on a Linux-based machine with a focus on managing repositories and access control through SSH. In this procedure, a small server is set up to add the keys to the **$HOME/.ssh/authorize_keys** folder, which provides access for *builders* to clone the repository.

For any Git repository management software that is not officially supported, there is usually a location to input the key that is often labeled as **Deploy Keys**.

6.1.1.1.2. Webhook

To automatically trigger a build, you must **POST** a **.json** payload to the webhook URL using the following format:



NOTE

This request requires a **Content-Type** header containing **application/json** in order to be valid.

Example webhook

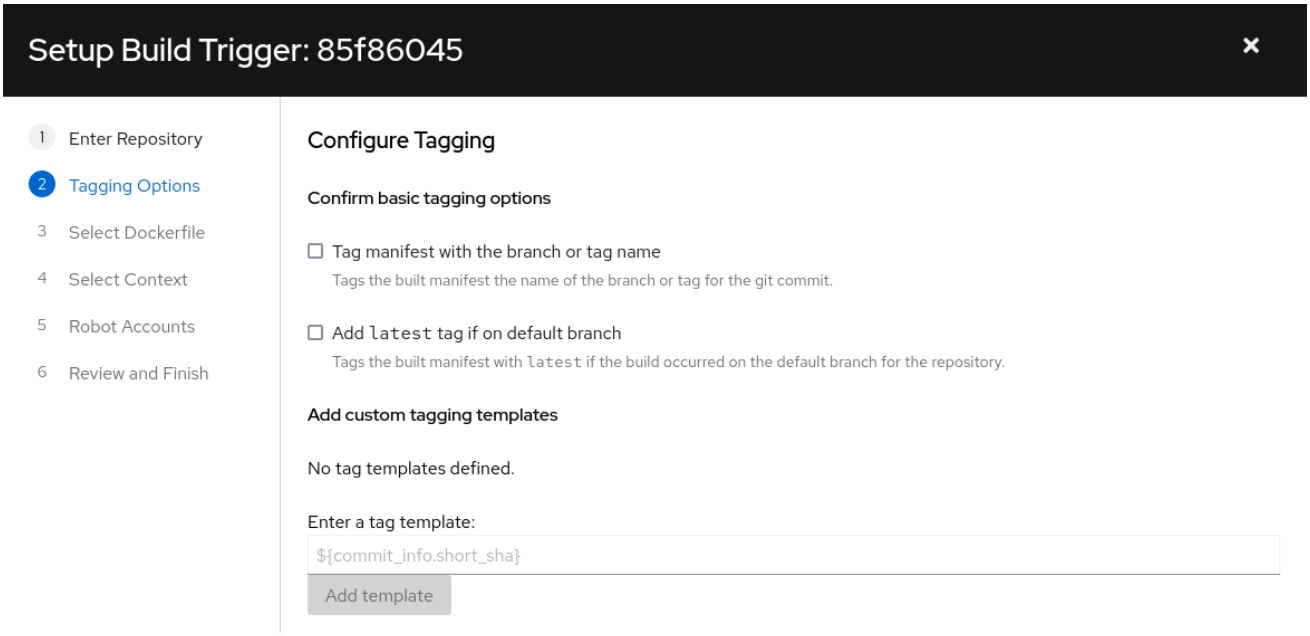
```
{
  "commit": "1c002dd",                                  // required
  "ref": "refs/heads/master",                           // required
  "default_branch": "master",                           // required
  "commit_info": {                                      // optional
    "url": "gitsoftware.com/repository/commits/1234567",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

This can typically be accomplished with a **post-receive** Git hook, however it does depend on your server setup.

6.1.2. Tag naming for build triggers

Custom tags are available for use in Red Hat Quay.

One option is to include any string of characters assigned as a tag for each built image. Alternatively, you can use the following tag templates on the **Configure Tagging** section of the build trigger to tag images with information from each commit:



- **${commit}**: Full SHA of the issued commit
- **${parsed_ref.branch}**: Branch information (if available)
- **${parsed_ref.tag}**: Tag information (if available)
- **${parsed_ref.remote}**: The remote name
- **${commit_info.date}**: Date when the commit was issued
- **${commit_info.author.username}**: Username of the author of the commit
- **${commit_info.short_sha}**: First 7 characters of the commit SHA
- **${committer.properties.username}**: Username of the committer

This list is not complete, but does contain the most useful options for tagging purposes. You can find the complete tag template schema on this page.

For more information, see Set up custom tag templates in build triggers for Red Hat Quay and Quay.io .

6.1.3. Skipping a source control-triggered build

To specify that a commit should be ignored by the build system, add the text **[skip build]** or **[build skip]** anywhere in your commit message.

6.2. MANUALLY TRIGGERING A BUILD

Builds can be triggered manually by using the following procedure.

Procedure

1. On the **Builds** page, **Start new build**
2. When prompted, select **Invoke Build Trigger**.
3. Click **Run Trigger Now** to manually start the process.

4. Enter a commit ID from which to initiate the build, for example, **1c002dd**.
   After the build starts, you can see the *build ID* on the **Repository Builds** page.

5. You can check the status of your *build* by clicking the commit in the **Build History** page, or by running the following command:

```
$ oc get pods -n virtual-builders
```

   Example output

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

6. After the *build* has completed, the **oc get pods -n virtual-builders** command returns no resources:

```
$ oc get pods -n virtual-builders
```

   Example output

```
No resources found in virtual-builders namespace.
```

CHAPTER 7. CREATING AN OAUTH APPLICATION IN GITHUB

The following sections describe how to authorize Red Hat Quay to integrate with GitHub by creating an OAuth application. This allows Red Hat Quay to access GitHub repositories on behalf of a user.

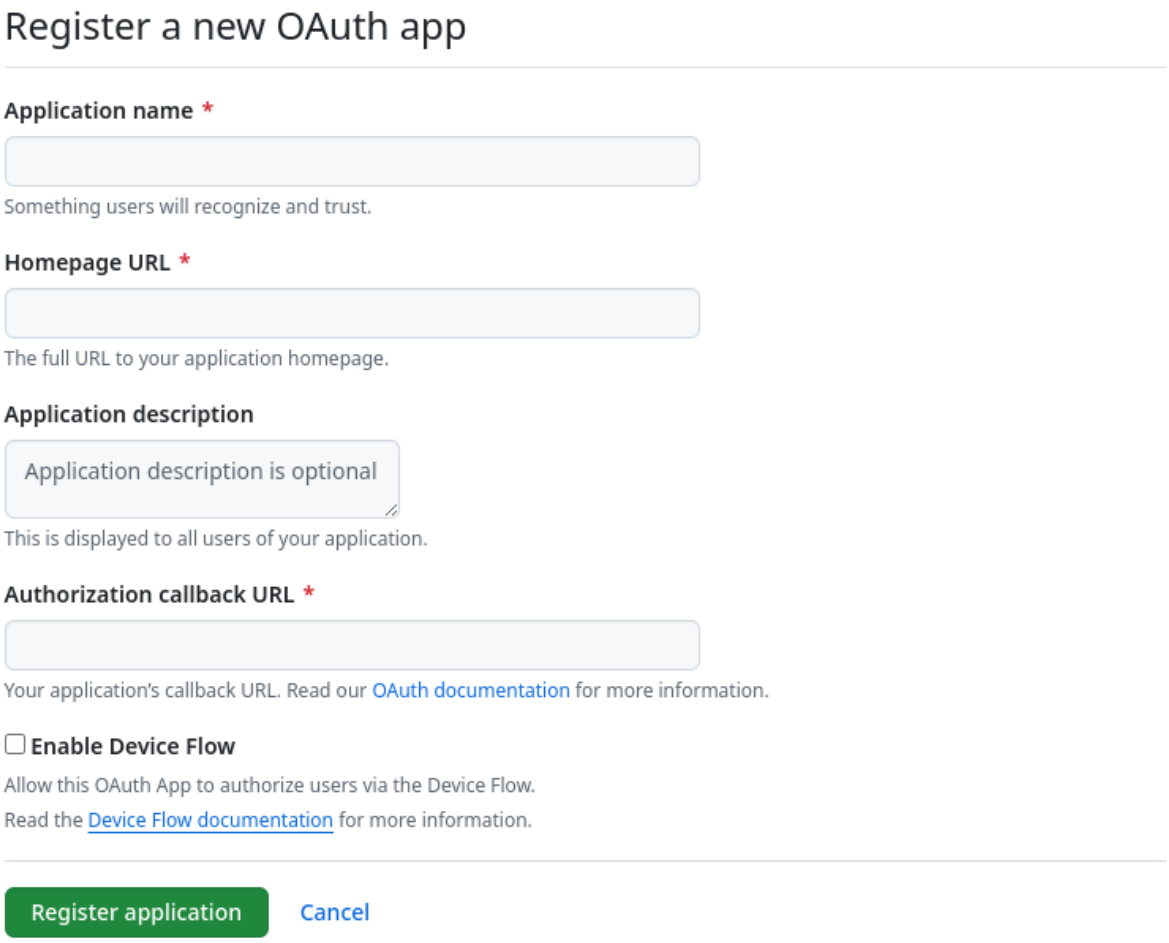
OAuth integration with GitHub is primarily used to allow features like automated builds, where Red Hat Quay can be enabled to monitor specific GitHub repositories for changes like commits or pull requests, and trigger contain image builds when those changes are made.

7.1. CREATE NEW GITHUB APPLICATION

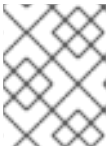
Use the following procedure to create an OAuth application in Github.

Procedure

1. Log into GitHub Enterprise.
2. In the navigation pane, select your username → **Your organizations**.
3. In the navigation pane, select **Applications** → **Developer Settings**.
4. In the navigation pane, click **OAuth Apps** → **New OAuth App**. You are navigated to the following page:



5. Enter a name for the application in the **Application name** textbox.
6. In the **Homepage URL** textbox, enter your Red Hat Quay URL.



NOTE

If you are using public GitHub, the Homepage URL entered must be accessible by your users. It can still be an internal URL.

7. In the **Authorization callback URL**, enter https://<RED_HAT_QUAY_URL>/oauth2/github/callback.
8. Click **Register application** to save your settings.
9. When the new application's summary is shown, record the Client ID and the Client Secret shown for the new application.

CHAPTER 8. TROUBLESHOOTING BUILDS

The *builder* instances started by the *build manager* are ephemeral. This means that they will either get shut down by Red Hat Quay on timeouts or failure, or garbage collected by the control plane (EC2/K8s). In order to obtain the *builds* logs, you must do so while the *builds* are running.

8.1. DEBUG CONFIG FLAG

The **DEBUG** flag can be set to **true** in order to prevent the *builder* instances from getting cleaned up after completion or failure. For example:

```
EXECUTORS:
  - EXECUTOR: ec2
    DEBUG: true
    ...
  - EXECUTOR: kubernetes
    DEBUG: true
    ...
```

When set to **true**, the debug feature prevents the *build nodes* from shutting down after the **quay-builder** service is done or fails. It also prevents the *build manager* from cleaning up the instances by terminating EC2 instances or deleting Kubernetes jobs. This allows debugging *builder node* issues.

Debugging should not be set in a production cycle. The lifetime service still exists; for example, the instance still shuts down after approximately two hours. When this happens, EC2 instances are terminated and Kubernetes jobs are completed.

Enabling debug also affects the **ALLOWED_WORKER_COUNT** because the unterminated instances and jobs still count toward the total number of running workers. As a result, the existing *builder workers* must be manually deleted if **ALLOWED_WORKER_COUNT** is reached to be able to schedule new *builds*.

8.2. TROUBLESHOOTING OPENSHIFT CONTAINER PLATFORM AND KUBERNETES BUILDS

Use the following procedure to troubleshooting OpenShift Container Platform Kubernetes Builds.

Procedure

1. Create a port forwarding tunnel between your local machine and a pod running with either an OpenShift Container Platform cluster or a Kubernetes cluster by entering the following command:

```
$ oc port-forward <builder_pod> 9999:2222
```

2. Establish an SSH connection to the remote host using a specified SSH key and port, for example:

```
$ ssh -i /path/to/ssh/key/set/in/ssh_authorized_keys -p 9999 core@localhost
```

3. Obtain the **quay-builder** service logs by entering the following commands:

```
$ systemctl status quay-builder
```

```
$ journalctl -f -u quay-builder
```