Use a private registry for container images

Redis Enterprise Software, its Kubernetes operator, and the Service Rigger are all distributed as separate container images. Your Kubernetes deployment will pull these images as needed. You can control where these images are pulled from within the operator deployment and also via the Redis Enterprise custom resources.

In general, images for deployments that do not have a registry domain name (e.g., gcr.io or localhost:5000) are pulled from the default registry associated with the Kubernetes cluster. A plain reference to redislabs/redis will likely pull from DockerHub (except on OpenShift where it pulls from Red Hat).

For security reasons (e.g., in air-gapped environments), you may want to pull the images from a public registry once and then push them to a private registry under your control.

**Warning** - It is very important that the images you are pushing to the private registry have the same exact version tag as the original images.

Furthermore, because [Docker rate limits public pulls](https://docs.docker.com/engine/security/docker-rate-limit/), you may want to consider pulling images from a private registry to avoid deployment failures when you hit your DockerHub rate limit.

The information below will help you track and configure where your deployments pull container images.

**Note:** IMPORTANT

- Each version of the Redis Enterprise operator is mapped to a specific version of Redis Enterprise Software. The semantic versions always match (e.g., 6.0.8), although the specific release numbers may be different (e.g., 6.0.8-1 is the operator version for RS 6.0.8-28).
- A specific operator version only supports a specific Redis Enterprise version. Other combinations of operator and Redis Enterprise versions are not supported.

**Find container sources**

Every pod in your deployed application has a source registry. Any image not prefixed by a registry domain name (e.g., “gcr.io”) will pull from the default registry for the Kubernetes cluster (i.e., DockerHub). You can use the commands below to discover the pull sources for the images on your cluster.

To list all the images used by your cluster:

```
kubectl get pods --all-namespaces -o jsonpath="{..image}" | tr -s '[:space:]' ' ' | uniq -c
```

To specifically determine the pull source for the Redis Enterprise operator itself, run the following command:

```
kubectl get pods --all-namespaces -o jsonpath="{..image}" | tr -s '[:space:]' ' ' | uniq -c | grep redislabs
```

You can limit this command to specific namespaces by replacing the `--all-namespaces` parameter with a set of `-n {namespace}` parameters, where each `{namespace}` is a specific namespace of interest on your cluster.

**Create a private container registry**
You can set up a private container registry in a couple of ways:

- On-premise via Docker registry, Red Hat Quay, or other providers
- Cloud provider based registries (e.g., Azure Container Registry, Google Container Registry, etc.).

Once you have set up a private container registry, you will identify the container registry using:

- A domain name
- A port (optional)
- A repository path (optional)

**Push images to a private container registry**

Important images for a Redis Enterprise Software deployment include:

- Redis Enterprise Software
- Bootstrapping a Redis Enterprise cluster node (in the operator image)
- The Service Rigger
- The Redis Enterprise Software operator

You will need to push all these images to your private container registry. In general, to push the images you must:

1. **Pull** the various images locally for the Redis Enterprise Software, the Service Rigger, and the operator.
2. **Tag** the local images with the private container registry, repository, and version tag.
3. **Push** the newly tagged images.

The example below shows the commands for pushing the images for Redis Enterprise Software and its operator to a private container registry:

```
PRIVATE_REPO=...your_repo...
RS_VERSION=6.0.8-28
OPERATOR_VERSION=6.0.8-1

docker pull redislabs/redis:${RS_VERSION}
docker pull redislabs/operator:${OPERATOR_VERSION}
docker pull redislabs/k8s-controller:${OPERATOR_VERSION}
docker tag redislabs/redis:${RS_VERSION} ${PRIVATE_REPO}/redislabs/redis:${RS_VERSION}
docker tag redislabs/operator:${OPERATOR_VERSION} ${PRIVATE_REPO}/redislabs/operator:${OPERATOR_VERSION}
docker tag redislabs/k8s-controller:${OPERATOR_VERSION} ${PRIVATE_REPO}/redislabs/k8s-controller:${OPERATOR_VERSION}
docker push ${PRIVATE_REPO}/redislabs/redis:${RS_VERSION}
docker push ${PRIVATE_REPO}/redislabs/operator:${OPERATOR_VERSION}
docker push ${PRIVATE_REPO}/redislabs/k8s-controller:${OPERATOR_VERSION}
```

**Configure deployments to use a private container registry**

Once you push your images to your private container registry, you need to configure your deployments to use that registry for Redis Enterprise Software and operator deployments. The operator container image is configured directly by the operator deployment bundle. The Redis Enterprise cluster pod (RS and bootstrapper) and Service Rigger images are configured in the Redis Enterprise custom resource.

Depending on your Kubernetes platform, your private container registry may require authentication. If you do need authentication,
add a pull secret to your namespace. Then you’ll need to configure Kubernetes and the operator to use the pull secret. The two following sections have examples of adding the imagePullSecrets to the operator deployment and pullSecrets to the cluster custom resource.

**Specify the operator image source**

The operator bundle contains the operator deployment and the reference to the operator image (redislabs/operator). To use a private container registry, you must change this image reference in your operator deployment file before you deploy the operator. If you apply this change to modify an existing operator deployment, the operator’s pod will restart.

In the operator deployment file, 'containers:image' should point to the same repository and tag you used when pushing to the private container registry:

```
${PRIVATE_REPO}/redislabs/operator:${OPERATOR_VERSION}
```

The example below specifies a 6.0.8-1 operator image in a Google Container Registry:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-enterprise-operator
spec:
  replicas: 1
  selector:
    matchLabels:
      name: redis-enterprise-operator
  template:
    metadata:
      labels:
        name: redis-enterprise-operator
    spec:
      serviceAccountName: redis-enterprise-operator
      containers:
        - name: redis-enterprise-operator
          image: gcr.io/yourproject/redislabs/operator:6.0.8-1
```

If your container registry requires a pull secret, configure imagePullSecrets on the operator deployment:

```
spec:
  template:
    spec:
      imagePullSecrets:
        - name: regcred
```

**Specify the Redis Enterprise cluster images source**

A Redis Enterprise cluster managed by the operator consists of three container images:

- `redislabs/redis`: the Redis Enterprise Software container image
- `redislabs/operator`: the bootstrapper is packaged within the operator container image
- `redislabs/k8s-controller`: the Service Rigger container image

By default, a new Redis Enterprise cluster is created using the container images listed above. These container images are pulled from the K8s cluster’s default container registry.
To pull the Redis Enterprise container images from a private container registry, you must specify them in the Redis Enterprise cluster custom resource.

Add the following sections to the spec section of your RedisEnterpriseCluster resource file:

- **redisEnterpriseImageSpec**: controls the Redis Enterprise Software container image. The version should match the RS version associated with the operator version.
- **bootstrapperImageSpec**: controls the bootstrapper container image. The version must match the operator version.
- **redisEnterpriseServicesRiggerImageSpec**: controls the Service Rigger container image. The version must match the operator version.

The REC custom resource example below pulls all three container images from a GCR private registry:

```yaml
apiVersion: app.redislabs.com/v1
kind: RedisEnterpriseCluster
metadata:
  name: rec
spec:
  nodes: 3
  redisEnterpriseImageSpec:
    imagePullPolicy: IfNotPresent
    repository: gcr.io/yourproject/redislabs/redis
    versionTag: 6.0.8-28
  bootstrapperImageSpec:
    imagePullPolicy: IfNotPresent
    repository: gcr.io/yourproject/redislabs/operator
    versionTag: 6.0.8-1
  redisEnterpriseServicesRiggerImageSpec:
    imagePullPolicy: IfNotPresent
    repository: gcr.io/yourproject/redislabs/k8s-controller
    versionTag: 6.0.8-1
```

If your private container registry requires pull secrets, you must add pullSecrets to the spec section:

```yaml
apiVersion: app.redislabs.com/v1
kind: RedisEnterpriseCluster
metadata:
  name: rec
spec:
  nodes: 3
  pullSecrets:
    - name: regcred
  redisEnterpriseImageSpec:
    imagePullPolicy: IfNotPresent
    repository: gcr.io/yourproject/redislabs/redis
    versionTag: 6.0.8-28
  bootstrapperImageSpec:
    imagePullPolicy: IfNotPresent
    repository: gcr.io/yourproject/redislabs/operator
    versionTag: 6.0.8-1
  redisEnterpriseServicesRiggerImageSpec:
    imagePullPolicy: IfNotPresent
    repository: gcr.io/yourproject/redislabs/k8s-controller
    versionTag: 6.0.8-1
```
Rate limiting with DockerHub

Docker has rate limits for image pulls. Anonymous users are allowed a certain number of pulls every 6 hours. For authenticated users, the limit is larger. These rate limits may affect your Kubernetes cluster in a number of ways:

- The cluster nodes will likely be treated as a single anonymous user.
- The number of pulls during a deployment might exceed the rate limit for other deployment dependencies, including the operator, Redis Enterprise Software, or other non-Redis pods.
- Pull failures may prevent your deployment from downloading the required images in a timely manner. Delays here can affect the stability of deployments used by the Redis Enterprise operator.

For these reasons, you should seriously consider where your images are pulled from to avoid failures caused by rate limiting. The easiest solution is to push the required images to a private container registry under your control.

Updated: February 2, 2022