builds for Red Hat OpenShift 1.0

Authentication

Understanding authentication at runtime
builds for Red Hat OpenShift 1.0 Authentication

Understanding authentication at runtime
Abstract

This document provides information about authentication at runtime.
## Table of Contents

CHAPTER 1. UNDERSTANDING AUTHENTICATION AT RUNTIME .......................... 3
  1.1. BUILD SECRET ANNOTATION .................................................. 3
  1.2. AUTHENTICATION TO GIT REPOSITORIES ................................. 3
      1.2.1. Basic authentication ................................................. 3
      1.2.2. SSH authentication .................................................. 4
      1.2.3. Usage of Git secret .................................................. 4
  1.3. AUTHENTICATION TO CONTAINER REGISTRIES .......................... 5
  1.4. ROLE-BASED ACCESS CONTROL ............................................. 6
builds for Red Hat OpenShift 1.0 Authentication
CHAPTER 1. UNDERSTANDING AUTHENTICATION AT RUNTIME

When building images, you might need to define authentication in the following scenarios:

- Authenticating to a container registry
- Pulling source code from Git

The authentication is done through the definition of secrets in which the required sensitive data is stored.

1.1. BUILD SECRET ANNOTATION

You can add an annotation `build.shipwright.io/referenced.secret: "true"` to a build secret. Based on this annotation, the build controller takes a reconcile action when an event, such as create, update, or delete triggers for the build secret. The following example shows the usage of an annotation with a secret:

```
apiVersion: v1
data:
  .dockerconfigjson: <pull_secret> 1
kind: Secret
metadata:
  annotations:
    build.shipwright.io/referenced.secret: "true" 2
  name: secret-docker
  type: kubernetes.io/dockerconfigjson
```

1. Base64-encoded pull secret.
2. The value of the `build.shipwright.io/referenced.secret` annotation is set to `true`.

This annotation filters secrets which are not referenced in a build instance. For example, if a secret does not have this annotation, the build controller does not reconcile even if the event is triggered for the secret. Reconciling on triggering of events allows the build controller to re-trigger validations on the build configuration, helping you to understand if a dependency is missing.

1.2. AUTHENTICATION TO GIT REPOSITORIES

You can define the following types of authentication for a Git repository:

- Basic authentication
- Secure Shell (SSH) authentication

You can also configure Git secrets with both types of authentication in your `Build` CR.

1.2.1. Basic authentication

With basic authentication, you must configure the user name and password of the Git repository. The following example shows the usage of basic authentication for Git:
1.2.2. SSH authentication

With SSH authentication, you must configure the Tekton annotations to specify the hostname of the Git repository provider for use. For example, `github.com` for GitHub or `gitlab.com` for GitLab.

The following example shows the usage of SSH authentication for Git:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: secret-git-ssh-auth
  annotations:
    build.shipwright.io/referenced.secret: "true"
    build.shipwright.io/secret-auth: "true"
  type: kubernetes.io/ssh-auth

data:
  ssh-privatekey: |
    # Insert ssh private key, base64 encoded
```

1. The type of the Kubernetes secret.

2. Base64 encoding of the SSH key used to authenticate into Git. You can generate this key by using the `base64 ~/.ssh/id_rsa.pub` command, where `~/.ssh/id_rsa.pub` denotes the default location of the key that is generally used to authenticate to Git.

1.2.3. Usage of Git secret

After creating a secret in the relevant namespace, you can reference it in your Build custom resource (CR). You can configure a Git secret with both types of authentication.

The following example shows the usage of a Git secret with SSH authentication type:

```yaml
apiVersion: shipwright.io/v1beta1
kind: Build
metadata:
  name: buildah-golang-build
spec:
  source:
```

1. The type of the Kubernetes secret.

2. The field to store your user name and password in clear text.
The following example shows the usage of a Git secret with basic authentication type:

```
apiVersion: shipwright.io/v1beta1
git:
  url: git@gitlab.com:userjohn/newtaxi.git
  cloneSecret: secret-git-ssh-auth
```

The following example shows the usage of a Git secret with basic authentication type:

```
apiVersion: shipwright.io/v1beta1
git:
  url: git@gitlab.com:userjohn/newtaxi.git
  cloneSecret: secret-git-ssh-auth
```

1.3. AUTHENTICATION TO CONTAINER REGISTRIES

To push images to a private container registry, you must define a secret in the respective namespace and then reference it in your Build custom resource (CR).

Procedure

1. Run the following command to generate a secret:

   ```
   $ oc --namespace <namespace> create secret docker-registry <container_registry_secret_name> \
       --docker-server=<registry_host> \
       --docker-username=<username> \
       --docker-password=<password> \
       --docker-email=<email_address>
   ```

   1. The `<registry_host>` value denotes the URL in this format `https://<registry_server>/<registry_host>`.
   2. The `<username>` value is the user ID.
   3. The `<password>` value can be your container registry password or an access token.

2. Run the following command to annotate the secret:

   ```
   $ oc --namespace <namespace> annotate secrets <container_registry_secret_name> build.shipwright.io/referenced.secret='true'
   ```

3. Set the value of the `spec.output.pushSecret` field to the secret name in your Build CR:

   ```
   apiVersion: shipwright.io/v1beta1
git:
    url: git@gitlab.com:userjohn/newtaxi.git
    cloneSecret: secret-git-ssh-auth
   
   # ...
   ```
1.4. ROLE-BASED ACCESS CONTROL

The release deployment YAML file includes two cluster-wide roles for using Builds objects. The following roles are installed by default:

- **shipwright-build-aggregate-view**: Grants you read access to the Builds resources, such as `BuildStrategy`, `ClusterBuildStrategy`, `Build`, and `BuildRun`. This role is aggregated to the Kubernetes `view` role.

- **shipwright-build-aggregate-edit**: Grants you write access to the Builds resources that are configured at namespace level. The build resources include `BuildStrategy`, `Build`, and `BuildRun`. Read access is granted to all `ClusterBuildStrategy` resources. This role is aggregated to the Kubernetes `edit` and `admin` roles.

Only cluster administrators have write access to the `ClusterBuildStrategy` resources. You can change this setting by creating a separate Kubernetes `ClusterRole` role with these permissions and binding the role to appropriate users.