Red Hat Trusted Application Pipeline 1.0

Managing compliance with Enterprise Contract

Learn how Enterprise Contract enables you to better verify and govern compliance of the code you promote. Additionally, customize the sample policies to fit your corporate standards.

Last Updated: 2024-07-01

Red Hat Trusted Application Pipeline 1.0 Managing compliance with Enterprise Contract

Learn how Enterprise Contract enables you to better verify and govern compliance of the code you promote. Additionally, customize the sample policies to fit your corporate standards.

Legal Notice

Abstract

This document provides information about how to manage and maintain software supply chain security by defining and enforcing policies for building and testing container images.

Table of Contents

PREFACE

Enterprise Contract is a policy-driven workflow tool for maintaining software supply chain security by defining and enforcing policies for building and testing container images. A secure CI/CD workflow should include artifact verification to detect problems early. It's the job of Enterprise Contract to validate that a container image is signed and attested by a known and trusted build system.

CHAPTER 1. ENTERPRISE CONTRACT FOR RED HAT TRUSTED APPLICATION PIPELINE

The more complex a software supply chain becomes, the more critical it is to employ reliable checks and best practices to guarantee software artifact integrity and source code dependability. Artifacts such as your image containers. This is where Red Hat Enterprise Contract enters your Red Hat Trusted Application Pipeline build and deploy experience.

Enterprise Contract is a policy-driven workflow tool for maintaining software supply chain security by defining and enforcing policies for building and testing container images. For a build system that creates Supply-chain Levels for Software Artifacts (SLSA) provenance attestations, such as Tekton with Tekton Chains and GitHub Actions with the SLSA GitHub Generator, checking the signatures and confirming that the contents of the attestations actually match what is expected is a critical part of verifying and maintaining the integrity of your software supply chain. A secure CI/CD workflow should include artifact verification to detect problems early. It's the job of Enterprise Contract to validate that a container image is signed and attested by a known and trusted build system.

The general steps for validating a signed and attested container image are as follows:

1. Create or copy a container image with Red Hat Trusted Application Pipeline.
2. Generate a signing key with Cosign.
3. Sign the container image with Cosign.
4. Attest the image with Cosign.
5. Verify your signed and attested container image with the Enterprise Contract CLI.

But what does it mean to *sign* and *attest to the provenance of* a software artifact like a container image? Why do it? And how?

Signed software artifacts like container images are at a significantly lower risk of several attack vectors than unsigned artifacts. When a container image is signed, various cryptographic techniques bind the image to a specific entity or organization. The result is a digital signature that verifies the authenticity of the image so that you can trace it back to its creator—that entity or organization—and also verify that the image wasn't altered or tampered with after it was signed. For more information about software supply chain threats, see Supply chain threats.

Enterprise Contract uses the industry standard Sigstore Cosign as a resource library to validate your container images. With Trusted Artifact Signer, Red Hat's supported version of the Sigstore framework, you can use your own on-prem instance of Sigstore's services to sign and attest your container images with the Cosign CLI. For more information about Trusted Artifact Signer, see Red Hat Trusted Artifact Signer.

As for software artifact *attestation*, it can't happen without provenance. *Provenance* is the verifiable information about software artifacts like container images that describes where, when, and how that artifact was produced. The attestation itself is an authenticated statement, in the form of metadata, that proves that an artifact is intact and trustworthy. Enterprise Contract uses that attestation to cryptographically verify that the build was not tampered with, and to check the build against any set of policies, such as SLSA requirements. For more information about SLSA, see About SLSA.

When you push your code from either the RHTAP development namespace to the stage namespace, or from the stage namespace to the production namespace, Enterprise Contract automatically runs its validation checks to make sure your container image was signed and attested by known and trusted build systems. When your image passes the Enterprise Contract check, you can merge your code changes to

complete your promotion from one environment to the next. For more information about deploying your application to a different namespace, see Trusted Application Pipeline Software Template. For more inforamtion about where RHTAP saves your deployment manifests, see the RHTAP GitOps repository and its YAML files.

Additional resources

For more information about signing and attesting a container image, see Signing a container image.

CHAPTER 2. INSTALLING THE ENTERPRISE CONTRACT COMMAND LINE

Prerequisites

- A working Red Hat Trusted Artifact Signer installation on Red Hat OpenShift Container Platform version 4.13 or higher.
- A workstation with the **cosign** and **oc** binary files installed.
- Access to the OpenShift web console.

Procedure

1. Download the **ec** binary file from the OpenShift cluster.
   a. Log in to the OpenShift web console. From the home page, click the **?** icon, select **Command line tools**, go to the **ec** download section, then click the link for your platform.
   b. Open a terminal on your workstation, decompress the binary **.gz** file, then set the execute bit:

   #### Example

   ```
   gunzip ec-amd64.gz
   chmod +x ec-amd64
   ```

2. Move and rename the binary to a location within your **$PATH** environment:

   #### Example

   ```
   sudo mv ec-amd64 /usr/local/bin/ec
   ```

Verification

- Run the **ec version** command. The result should be the version of the Enterprise Contract CLI that you just installed.

CHAPTER 3. CREATING A POLICY

An Enterprise Contract *policy* is a rule or set of rules and Enterprise Contract-specific annotations. Enterprise Contract can perform several types of policy checks, including checking all of policy rules required for Red Hat products. Enterprise Contract uses the general purpose policy engine called Open Policy Agent, or OPA. OPA defines its policy rules in their own language, called Rego. This means that the policy rules from OPA that are in an Enterprise Contract policy are also defined in Rego.

Procedure

1. Create a Rego file to define a new policy rule, as in the following example:

```
echo 'package zero_to_hero

import future.keywords.contains
import future.keywords.if
import future.keywords.in


# METADATA 1
# title: Builder ID
# description: Verify the SLSA Provenance has the builder.id set to
#   the expected value.
# custom:
#   short_name: builder_id 2
#   failure_msg: The builder ID %q is not the expected %q
#   solution: >-
#     Ensure the correct build system was used to build the container
#     image.
deny contains result if {
 some attestation in input.attestations 3
 attestation.statement.predicateType == "https://slsa.dev/provenance/v0.2"

 expected := "https://localhost/dummy-id"
 got := attestation.statement.predicate.builder.id

 expected != got

 result := {
  "code": "zero_to_hero.builder_id",
  "msg": sprintf("The builder ID %q is not expected, %q", [got, expected])
 }
}
' > rules.rego
```

1 The **METADATA** comment block—the first 10 lines of code, which are all preceded by hashtags (#)--is how rego specifies rules annotations so that Enterprise Contract can include those annotation details its successes and violations report. For more information about rego metadata and annotations, see Metadata. For more information about the annotations that Enterprise Contract policy rules must contain, see Rule annotations.

2 This single policy rule verifies that the **builder.id** in your new policy rule matches the **builder.id** in your Supply-chain Levels for Software Artifacts, or *SLSA*, provenance.

3 **input.attestations** is a rego object that contains all of the information about your container image, its signature, and its attestations. See Policy Input for more information about

image, its signature, and its attestations. See Policy input for more information about where and how Enterprise Contract defines **input.attestations** contents.

TIP

You can save the **input.attestations** object to a JSON file so that you can borrow from it when you specify new policy rules. To save **input.attestations** as a JSON file, run a command that's similar to the following example:

```
ec validate image --public-key cosign.pub --image "$REPOSITORY:latest" --policy
policy.yaml \
   --show-successes --info --output yaml
```

2. Create a policy configuration to use your new policy rule, as in the following example:

```
echo "
---
sources:
  - policy:
      - $(pwd)/rules.rego
" > policy.yaml
```

3. Use your new policy to validate your container image, and to display additional information in the successes and violations report, as in the following example:

```
ec validate image --public-key cosign.pub --image "$REPOSITORY:latest" --policy
policy.yaml \
   --show-successes --info --output yaml
```

Verification

- Check the Successes and violations report to make sure that your new rule is in the **successes** list.

Additional resources

- For a set of useful Enterprise Contract policy rules, see the ec-policies GitHub repository.
- For more information about OPA and Rego, see OPA's Policy Language content.
- For more information about SLSA provenance, see SLSA Provenance.

3.1. CONFIGURING A POLICY

You can configure an Enterprise Contract policy with an inline JSON or YAML string. This policy, sometimes called a *config* or a *contract*, specifies where Enterprise Contract should find the rules and data to use to apply the policies you want to enforce. You can also include or exclude a single rule or a particular package of rules.

Procedure

1. Configure your policy in the command line as a JSON or YAML string, as in the following example:

```
ec validate image --policy '{
    "configuration": {
        "include": ["@minimal"]
    },
    "sources": [
        {
            "policy": ["oci::quay.io/enterprise-contract/ec-release-policy:latest"],
            "data": ["git::https://github.com/enterprise-contract/ec-policies//example/data"]
        }
    ]
}' ...
```

2. (Optional) Exclude a particular package of rules from your Enterprise Contract policy, as in the following example:

```
exclude:
- attestation_task_bundle
- slsa_build_scripted_build
```

This command includes every rule from every package except for the rules in the specified packages.

3. (Optional) Exclude a single rule, as in the following example:

```
exclude:
- attestation_task_bundle.unacceptable_task_bundle
```

This commands includes every rule from the **attestation_task_bundle** package except for the **unacceptable_task_bundle** rule.

4. (Optional) Include rules from only a particular package, as in the following example:

```
include:
- test
- java
```

This command includes only the rules from the specified packages.

5. (Optional) Include only some rules from a particular package. This means that you can specify both **include** and **exclude** to select only the rules you want your Enterprise Contract policy to include, as in the following example:

```
include:
- "*" 1
- attestation_task_bundle.unacceptable_task_bundle
exclude:
- attestation_task_bundle.*
```

1. The asterisk (*) acts as a wildcard to match any package. Note that it does not match partial names, which means that, for example, you can't specify "s*" to match every package that starts with "s".

These commands specify that you want to include *only* the **unacceptable_task_bundle** rule from the **attestation_task_bundle** package, and exclude all the other rules in that package.

6. (Optional) Exclude certain checks so that Enterprise Contract can validate your container image even if those checks fail or don't complete, as in the following example:

```
exclude:
- test:get-clair-scan
- test:clamav-scan
```

This command specifies that, if either of the identified checks fails or doesn't complete, Enterprise Contract can still finish to validate your container image.

7. (Optional) Modify the defaults for rules in a package by running either the **config.policy.include** command or the **config.policy.exclude** command, along with a list of strings.
Your list of strings should include one of the following:

- "package name" - Choose from the packages in the Available rule collections list.
- "rule name" - Specify a rule name by entering the name of the package and the rule code, separated by a dot (.), as in this example: **attestation_type.unknown_att_type**. You can find rule codes under "Attestation type" here.
- "package name:term" - Some policy rules process a list of items. When you add "term" to the "package name" string, you can exclude or include a particular item from that list. This works similarly to "package name," except that it applies only to policy rules in the package that match that term. For example, if you run the test package, you can choose to ignore a given test case but include all the others.
- "rule name:term" - This is similar to "package name:term" except that, instead of including or excluding an *item* from a package, you can include ot exclude a particular package *policy rule*.
- "@collection name" - Add this to your string to specify a predefined collection of rules. TMake sure you prefix the collection name with the @ symbol. Choose from the available rule collections here.

Additional resources

For a comprehensive list of release policy details, see Release Policy.

CHAPTER 4. SIGNING A CONTAINER IMAGE

Prerequisites

- Access to the OpenShift web console.
- A working Red Hat Trusted Artifact Signer installation running on OpenShift version 4.13 or later.
- A workstation with the **ec**, **cosign**, and **oc** binary files installed.

Procedure

1. Log in to your OpenShift cluster:

   #### Syntax

   oc login --token=TOKEN --server=SERVER_URL_AND_PORT

   #### Example

   oc login --token=sha256~ZvFDBvoIYAbVECixS4-WmkN4RfnNd8Neh3y1WuiFPXC --server=https://example.com:6443

   

   #### NOTE

   To find your command line login token and URL, log in to the OpenShift web console. Click your user name, then click **Copy login command**. If prompted, enter your user name and password again, then click **Display Token**.

2. Log in to Trusted Artifact Signer. Make sure to configure your Trusted Artifact Signer shell environment to sign and verify container images; for example:

   ```
   cd sigstore-ocp
   source tas-env-variables.sh
   ```

   You also have the option to set the environment variables manually; for example:

   ```
   export OPENSHIFT_APPS_SUBDOMAIN=apps.$(oc get dns cluster -o jsonpath='{
   .spec.baseDomain }')
   export OIDC_AUTHENTICATION_REALM=sigstore
   export FULCIO_URL=https://fulcio.$OPENSHIFT_APPS_SUBDOMAIN
   export OIDC_ISSUER_URL=https://keycloak-keycloak-
   system.$OPENSHIFT_APPS_SUBDOMAIN/auth/realms/$OIDC_AUTHENTICATION_REALM

   export REKOR_URL=https://rekor.$OPENSHIFT_APPS_SUBDOMAIN
   export TUF_URL=https://tuf.$OPENSHIFT_APPS_SUBDOMAIN
   ```

   #### Example

   ```
   $ source ./tas-env-vars.sh
   ```

3. Log out of your OpenShift cluster by running this command: **oc logout**.

4. Identify the container image you want to sign and attest; for example: **IMAGE=quay.io/lucarval/rhtas-test@sha256:6b95efc134c2af3d45472c0a2f88e6085433df058cc210abb2bb061ac4d74359**.

5. Indicate to RHTAP that you want to sign and attest your container image with Trusted Artifact Signer instead of the public Sigstore deployment. Enter this command: **cosign initialize --mirror=$TUF_URL --root=$TUF_URL/root.json**.

6. Sign your container image with the following command:

```
cosign sign -y --fulcio-url=$FULCIO_URL --rekor-url=$REKOR_URL \
    --oidc-issuer=$OIDC_ISSUER_URL $IMAGE
```

7. When prompted, log in to the Keycloak instance that RHTAP installed when you installed Trusted Artifact Signer. This is so Keycloak can authenticate you.

Next steps

Your image is now signed. Now you can:

1. Create a SLSA provenance attestation and associate it with your container image.

2. Validate your container image with the Red Hat Enterprise Contract.

Additional resources

- For more information about Red Hat Trusted Artifact Signer, and especially about how it works with RHTAP and Enterprise Contract, see Red Hat Trusted Artifact Signer Deployment.

- For more information about Keycloak, see Keycloak.org.

4.1. GENERATING A SIGNING KEY TO SIGN AND ATTEST A CONTAINER IMAGE

You must have a signing key before you can sign and attest a container image.

Prerequisites

- A workstation with the **cosign** binary files installed.

Procedure

1. In your CLI, run this command: **cosign generate-key-pair**.

2. When prompted, enter a new password for the key-pair. Make sure your password is memorable and strong.

Verification

- You should now have two new files in your working directory: a **cosign.pub** file and a **cosign.key** file.
    - The **cosign.pub** file contains your public signing key. You can share this key with any collaborator who needs to validate the container image.

- The **cosign.key** file is your private key for signing content. Only the person responsible for signing and attesting images should have access to the **cosign.key** file.

4.2. VALIDATING CONTAINER IMAGE SIGNATURES WITH ENTERPRISE CONTRACT AND TRUSTED ARTIFACT SIGNER

When you install the Red Hat Trusted Artifact Signer service RHTAS, you can use the **ec** binary file to validate the attestation and signature of container images that use the RHTAS service's keyless signing framework. For more information about installing RHTAS, see Installing Red Hat Trusted Artifact Signer using the Operator Lifecycle Manager.

Prerequisites

- A working RHTAS installation running on OpenShift version 4.13 or later.
- Access to the OpenShift web console.
- A workstation with the **cosign** and **oc** binary files installed.

Procedure

1. Download the **ec** binary file from the OpenShift cluster:
   a. Log in to the OpenShift web console. From the home page, click the **?** icon in the upper right, then select **Command Line Tools**.
   b. From the **ec download** section, click the link for your platform.
   c. Open a terminal, decompress the **.gz** file, and set the execute bit on the **ec** binary file:

      Examples for Linux and macOS

      - **$ gunzip ec-amd64.gz**
      - **$ chmod +x ec-amd64**

   d. Move the **ec** binary file to a directory within your **$PATH** environment:

      Example

      $ sudo mv ec-amd64 /usr/local/bin/ec

TIP

Run the **ec validate image --help** command to see all the image validation command options.

2. Configure your shell environment for container image signing and verification.
   a. Open a terminal and run the **tas-env-variables.sh** script from the **sigstore-ocp** directory:

      Example

```
cd sigstore-ocp
source tas-env-variables.sh
```

b. (Optional) Set the environment variables manually:

Example

```
export OPENSHIFT_APPS_SUBDOMAIN=apps.$(oc get dns cluster -o jsonpath='{
.spec.baseDomain }')
export OIDC_AUTHENTICATION_REALM=sigstore
export FULCIO_URL=https://fulcio.$OPENSHIFT_APPS_SUBDOMAIN
export OIDC_ISSUER_URL=https://keycloak-keycloak-
system.$OPENSHIFT_APPS_SUBDOMAIN/auth/realms/$OIDC_AUTHENTICATION_RE
ALM
export REKOR_URL=https://rekor.$OPENSHIFT_APPS_SUBDOMAIN
export TUF_URL=https://tuf.$OPENSHIFT_APPS_SUBDOMAIN
```

Example

$ source ./tas-env-vars.sh

3. Initialize The Update Framework (TUF) system:

Example

cosign initialize --mirror=$TUF_URL --root=$TUF_URL/root.json

4. Sign your container image:

Syntax

cosign sign -y --fulcio-url=$FULCIO_URL --rekor-url=$REKOR_URL --oidc-issuer=$OIDC_ISSUER_URL IMAGE_NAME

Example

cosign sign -y --fulcio-url=$FULCIO_URL --rekor-url=$REKOR_URL --oidc-issuer=$OIDC_ISSUER_URL example-hello-world@sha256:2788a47fd0ef1ece30898c1e608050ea71036d3329b9772dbb3d1f69313f745c

In the web browser that opens, sign the container image with an email address.

5. Create a **predicate.json** file:

Example

```
{
  "builder": {
    "id": "https://localhost/dummy-id"
  },
  "buildType": "https://example.com/tekton-pipeline",
  "invocation": {},
  "buildConfig": {},
  "metadata": {
    "completeness": {
      "parameters": false,
      "environment": false,
      "materials": false
    },
    "reproducible": false
```

```
  },
  "materials": []
}
```

6. Associate the **predicate.json** file with your container image:

   Syntax

```
cosign attest -y --predicate ./predicate.json \
--type slsaprovenance IMAGE_NAME:TAG
```

   Example

```
$ cosign attest -y --predicate ./predicate.json \
--type slsaprovenance example.io/hello-world:latest
```

7. Verify that the container image has at least one attestation and signature:

   Syntax

   cosign tree IMAGE_NAME:TAG

   Example

```
$ cosign tree example.io/hello-world:latest
  Supply Chain Security Related artifacts for an image: example.io/hello-
world@sha256:7de5fa822a9d1e507c36565ee0cf50c08faa64505461c844a3ce3944d23efa35
└──  Attestations for an image tag: example.io/hello-world:sha256-
7de5fa822a9d1e507c36565ee0cf50c08faa64505461c844a3ce3944d23efa35.att
   └──
sha256:40d94d96a6d3ab3d94b429881e1b470ae9a3cac55a3ec874051bdecd9da06c2e
└──  Signatures for an image tag: example.io/hello-world:sha256-
7de5fa822a9d1e507c36565ee0cf50c08faa64505461c844a3ce3944d23efa35.sig
   └──
sha256:f321712507l5d4538aec33adc40fac2343f5092631d4fc2457e2116a489387b7
```

8. Verify the container image with Enterprise Contract:

   Syntax

```
ec validate image --image IMAGE_NAME:TAG \
--certificate-identity-regexp 'SIGNER_EMAIL_ADDR' \
--certificate-oidc-issuer-regexp 'keycloak-keycloak-system' \
--output yaml --show-successes
```

   Example

```
$ ec validate image --image example.io/hello-world:latest \
--certificate-identity 'jdoe@example.com' \
--certificate-oidc-issuer 'keycloak-keycloak-system' \
--output yaml --show-successes

success: true
successes:
```

```
  - metadata:
      code: builtin.attestation.signature_check
    msg: Pass
  - metadata:
      code: builtin.attestation.syntax_check
    msg: Pass
  - metadata:
      code: builtin.image.signature_check
    msg: Pass
ec-version: v0.1.2427-499ef12
effective-time: "2024-01-21T19:57:51.338191Z"
key: ""
policy: {}
success: true
```

Enterprise Contract generates a pass/fail report with details about any security violations. When you add the **--info** flag, the report includes more details and possible solutions for any violations.

Additional resources

- For more information about RHTAS, see Product Documentation for Red Hat Trusted Artifact Signer.
- For more information about TUF, see The Update Framework.
- For more information about signing and verifying container images with Cosign, see ADD LATER.
- For more information about SLSA provenance predicate specification, see SLSA Provenance.

CHAPTER 5. ATTESTING AND VALIDATING A CONTAINER IMAGE

Before Enterprise Contract can validate your signed container image, you must first create SLSA provenance and associate it with your container image. Provenance is the verifiable information about software artifacts, including where, when, and how a given software "link" in a supply chain was produced. For more information about Supply-chain Levels for Software Artifacts (SLSA) provenance, see SLSA Provenance.

Prerequisites

- A signed container image.
- Access to the OpenShift web console.
- A working Red Hat Trusted Artifact Signer installation running on OpenShift version 4.13 or later.
- A workstation with the **cosign** and **oc** binary files installed.

Procedure

1. Create a SLSA provenance **predicate.json** file; for example:

```
echo '{
 "builder": {
   "id": "https://localhost/dummy-id"
 },
 "buildType": "https://localhost/dummy-type",
 "invocation": {},
 "buildConfig": {},
 "metadata": {
   "buildStartedOn": "2023-09-25T16:26:44Z",
   "buildFinishedOn": "2023-09-25T16:28:59Z",
   "completeness": {
     "parameters": false,
     "environment": false,
     "materials": false
   },
   "reproducible": false
 },
 "materials": []
}
' > predicate.json
```

2. Sign and attest the **predicate.json** file you just created; for example:

```
cosign attest -y --fulcio-url=$FULCIO_URL \
  --rekor-url=$REKOR_URL \
  --oidc-issuer=$OIDC_ISSUER_URL \
  --predicate predicate.json \
  --type slsaprovenance $IMAGE
```

Keycloak opens to automatically authenticate you based on your login when you signed your container image.

3. Verify the signature and attestation with Enterprise Contract, for example:

```
ec validate image --image $IMAGE \
  --certificate-identity-regexp '.*' \
  --certificate-oidc-issuer-regexp '.*' \
  --output yaml --show-successes
```

IMPORTANT

Be as specific as possible when you run the **ec validate image** command so that each signature matches the expected **identity**.

Verification

- When Enterprise Contract has validated your container image, a detailed report of all Enterprise Contract verifications and signatures opens.

Additional resources

- For more information about Red Hat Trusted Artifact Signer, and especially about how it works with RHTAP and Enterprise Contract, see Red Hat Trusted Artifact Signer Deployment.
- For more information about SLSA, see slsa.dev.
- For more information about Keycloak, see keycloak.org.

5.1. VERIFYING JSON AND YAML DEFINITIONS

When you're considering all of your options for securing your software supply chain, you might choose to apply policies to task or pipeline definitions. For example, maybe you want to use Enterprise Contract to check which tasks are performed in a given pipeline. Perhaps you want to make some pipeline tasks mandatory, allow only certain tasks to be performed, or any other convention you want to enforce before tasks and pipelines run.

Prerequisites

- You've installed the Enterprise Contract CLI. For install instructions, refer to the Installing Enterprise Contract CLI guide. You can find the required files in the **ec-cli** GitHub repository.

Procedure

To verify that an Enterprise Contract JSON or YAML definition is valid, perform the following steps:

- In the Enterprise Contract CLI, enter the **ec validate input** command.