# Doc analysis

| **Jira link** | [RHDEVDOCS-4233](https://issues.redhat.com/browse/RHDEVDOCS-4233) |
| --- | --- |
| **Jira title** | Document guidance on auth strategies for pipelines (git, image registry) |
| **SME** | [Vincent Demeester](mailto:vdemeest@redhat.com) |
| **Tester/QE** |  |
| **Process to follow** | This Jira does not require us to follow the Change Management process |
| **Consideration** | This document does not include the “[Using VolumeMount(s)](https://docs.openshift-pipelines.org/pipeline/auth.html#volumemounts)” approach since it is not recommended for the users to follow. |

Queries

1. How do we factor [the existing doc](https://docs.openshift.com/container-platform/4.12/cicd/pipelines/authenticating-pipelines-using-git-secret.html) with this new topic?
   1. **Vincent**: That doc is only for git-secret.. we may want to either make it "wider" or create multiple pages (one for git, one for oci registries and one for "generic" secret)

**Making the existing doc wider**

1. Two sections on the page. The page title will be - Authenticating Pipelines.

# Section 1 - Authenticating pipelines using git secret

* 1. Section 2 - Authenticating pipelines using secrets

# **Authenticating pipelines using git Secrets**

This document describes the recommended approaches to handle authentication while executing task runs and pipeline runs in GitLab, GitHub, or Docker using git secrets.

There are two recommended approaches to authenticate using secrets:

* Approach 1: Uses Secrets and Workspaces
* Approach 2: Uses ServiceAccount

## **Using Secrets and Workspaces in Tekton Pipelines**

Secrets are Kubernetes resources that are used to store sensitive information such as passwords, access keys, and certificates. Workspaces are temporary directories that are mounted into a Task’s container to provide a workspace for the Task’s execution. Combing the use of Secrets and Workspaces provides a secure and flexible approach to managing credentials in Tektop pipelines.

The following examples illustrate how to use Secrets and Workspaces.

### **Example 1: git-clone Task**

This approach involves creating a task called `git-clone`, which clones a git repository using SSH authentication. The following are the steps to use Secrets and Workspace in Tekton Pipelines:

1. Create a Task called `git-clone` that clones a git repository using SSH authentication.
2. Define workspaces, describe the process to create a secret, and bind it to the workspace.

apiVersion: v1

kind: Secret

metadata:

name: my-github-ssh-credentials

type: Opaque

data:

id\_ed25519: *# […]*

known\_hosts: *# […]*

*# config: # […] # optional*

**Note:** To create a secret using `kubectl` run, kubectl create secret generic my-github-ssh-credentials \

--from-file**=**id\_ed25519**=**/path/to/.ssh/id\_ed25519 \

--from-file**=**known\_hosts**=**/path/to/.ssh/known\_hosts

secret/my-github-ssh-credentials created

1. Define task parameters for the repository URL, revision, and the git-init image that the Task uses.
2. Define results that the Task produces, for example, the commit SHA and the repository URL.
3. Define the steps that the Task executes, for example, copying the content of the ssh secret to the user’s home directory and running the git-init binary to clone the repository.
4. Create a secret with the ssh authentication content, for example, private key and known hosts.
5. Start the Task with the required parameters and workspace bindings. To start the Task run:

tkn task start git-clone \

--param url**=**git@github.com:vdemeester/buildkit-tekton \

--workspace name**=**output,emptyDir**=**"" \

--workspace name**=**ssh-directory,secret**=**my-github-ssh-credentials \

--use-param-defaults --showlog

**Example: Tekton Task for cloning a Git repository using an SSH key for authentication:**

apiVersion: tekton.dev/v1beta1

kind: Task

metadata:

name: git-clone

spec:

workspaces:

- name: output

description: The git repo will be cloned onto the volume backing this Workspace.

- name: ssh-directory *# (1)*

description: | # *(2)*

A .ssh directory with private key, known\_hosts, config, etc. Copied to

the user's home before git commands are executed. Used to authenticate

with the git remote when performing the clone. Binding a Secret to this

Workspace is strongly recommended over other volume types

params:

- name: url

description: Repository URL to clone from.

type: string

- name: revision

description: Revision to checkout. (branch, tag, sha, ref, etc...)

type: string

default: ""

- name: gitInitImage

description: The image providing the git-init binary that this Task runs.

type: string

default: "gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/git-init:v0.37.0"

results:

- name: commit

description: The precise commit SHA that was fetched by this Task.

- name: url

description: The precise URL that was fetched by this Task.

steps:

- name: clone

image: "$(params.gitInitImage)"

script: | *(3)*

#!/usr/bin/env sh

set -eu

# This is necessary for recent version of git

git config --global --add safe.directory '\*'

# (2)

cp -R "$(workspaces.ssh-directory.path)" "${HOME}"/.ssh

chmod 700 "${HOME}"/.ssh

chmod -R 400 "${HOME}"/.ssh/\*

CHECKOUT\_DIR="$(workspaces.output.path)/"

/ko-app/git-init \

-url="$(params.url)" \

-revision="$(params.revision)" \

-path="${CHECKOUT\_DIR}"

cd "${CHECKOUT\_DIR}"

RESULT\_SHA="$(git rev-parse HEAD)"

EXIT\_CODE="$?"

if [ "${EXIT\_CODE}" != 0 ] ; then

exit "${EXIT\_CODE}"

fi

printf "%s" "${RESULT\_SHA}" > "$(results.commit.path)"

printf "%s" "$(params.url)" > "$(results.url.path)"

1. The workspaces that the Task needs to run, for example, ssh-directory.
2. The description outlines the process for creating the secret.
3. The script copies the content of the secret (in the form of a folder) to ${HOME}/.ssh, which is the standard folder where `ssh` searches for credentials.

## 

### **Example 2: An existing docker configuration file**

This approach uses an existing docker configuration file inside a Task. The Task definition is the same as that of the “A git-clone task” approach, but Skopeo is used to replicate an image to a personal repository. This approach can be adapted to other tools such as Podman, Buildah, or Docker, as long as they are capable of interpreting a docker client configuration file.

Following are the steps involved in using a Docker configuration file inside a Tekton pipeline task.

1. Define a Tekton Task in your Kubernetes cluster with a reference to Skopeo image that copies a docker image to a specified repository.
2. In the Task definition, create a workspace called `dockerconfig` that holds a `config.json` file.
3. Define a secret in Kubernetes containing the `config.json` file, which contains the Docker authentication information.
4. Use the `kubectl create secret` command to create the secret, specifying the path to the `config.json` file on your local machine.
5. Start the TeKton Task using the `tkn task start` command, specifying the name of the Task, the workspace name (`dockerconfig`), and the secret name (`regcred`).
6. The Task executes the Skopeo container, which reads the `DOCKER\_CONFIG` environment variable to locate the `config.json` file containing the authentication information for accessing the Docker registry.
7. The Skopeo container then copies the specified Docker image to the specified repository.

**Example:** An existing docker configuration file inside a Task**:**

apiVersion: tekton.dev/v1beta1

kind: Task

metadata:

name: skopeo-copy

spec:

workspaces:

- name: dockerconfig *# (1)*

description: Includes a docker `config.json`

steps:

- name: clone

image: quay.io/skopeo/stable:v1.8.0

env:

- name: DOCKER\_CONFIG

value: $(workspaces.dockerconfig.path) *# (2)*

script: |

#!/usr/bin/env sh

set -eu

skopeo copy docker://docker.io/library/ubuntu:latest docker://docker.io/vdemeester/ubuntu-copy:latest

1. The name of the workspace that contains the config.json file. For a secret, this represents a key named config.json.
2. The DOCKER\_CONFIG environment variable points to the `dockerconfig` workspace path `skopeo` to get the authentication information.

### 

### **Example 3: The git-clone Task with optional workspaces**

This approach uses the optional workspaces feature. With optional workspaces, a Workspace might be bound to a Pod or Container (also known as, mounties as a volume) or it might not be present.

To make sure that the task takes this optional property into account the `git-clone` task is adapted to include an optional workspace. The `apiversion`, `kind`, `metadata`, `params`, `results`, and `steps` of the `git-clone` task are defined as in the standard task, but two workspaces are defined, one of which is optional. The `name` and `description` of the optional workspace are defined and the script is updated to conditionally copy the content of the workspace to the user’s home directory only if the workspace is bound.

The advantage of using optional workspaces is that it makes the task more flexible for example if a public Git repository is cloned, there s no need to bind a secret to a workspace.

The following steps illustrate how to use the optional workspace feature in the git-clone Task:

1. Create a Task called `git-clone-optional-ws` that clones a git repository using SSH authentication, with an optional workspace.
2. Define workspaces for the `git-ssh` secret and the optional workspace `my-optional-workspace`.
3. Define task parameters for the repository URL, revision, and the git-init image that the Task uses.
4. Define results that the Task produces, for example, the commit SHA and the repository URL.
5. Define the steps that the Task executes, including a condition to check if the optional workspace is bound before copying the content of the workspace to the user's home directory.

apiVersion: tekton.dev/v1beta1

kind: Task

metadata:

name: git-clone-optional-ws

spec:

workspaces:

- name: git-ssh

description: Workspace containing the git-ssh secret

- name: my-optional-workspace

description: Optional workspace that may be bound to a Pod or Container

optional: true

params:

- name: url

description: The URL of the Git repository to clone

type: string

default: ""

- name: revision

description: The Git revision to check out

type: string

default: "master"

- name: image

description: The image to use for the git-init container

type: string

default: "alpine/git:latest"

results:

- name: commit-sha

description: The commit SHA of the checked out revision

- name: repo-url

description: The URL of the cloned Git repository

steps:

- name: clone

image: $(params.image)

workingDir: /workspace/source

env:

- name: GIT\_SSH\_COMMAND

value: ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -i /workspace/git-ssh/ssh-privatekey

command:

- sh

args:

- -c

- |

git clone $(params.url) .

git checkout $(params.revision)

echo $(git rev-parse HEAD) > $(results.commit-sha.path)

echo $(params.url) > $(results.repo-url.path)

- name: optional-workspace

image: alpine:latest

workingDir: /workspace

command:

- sh

args:

- -c

- |

if [ -d "/workspace/my-optional-workspace" ]; then

cp -r /workspace/my-optional-workspace/\* $HOME/

fi

1. Create a secret with the ssh authentication content, for example, private key and known hosts.
2. Start the Task with the required parameters and workspace bindings. You can optionally bind the `my-optional-workspace` workspace to the Pod or Container where the Task runs.

apiVersion: tekton.dev/v1beta1

kind: TaskRun

metadata:

name: my-git-clone-taskrun

spec:

taskRef:

name: my-git-clone-task

params:

- name: url

value: https://github.com/my-org/my-repo.git

- name: revision

value: main

workspaces:

- name: my-workspace

persistentVolumeClaim:

claimName: my-pvc

- name: my-optional-workspace

volumeClaim:

claimName: my-optional-pvc

podTemplate:

spec:

containers:

- name: my-container

image: my-image

volumeMounts:

- name: my-workspace

mountPath: /workspace

- name: my-optional-workspace

mountPath: /optional-workspace

Example: A git clone task is modified to incorporate the optional Workspace feature.

apiVersion: tekton.dev/v1beta1

kind: Task

metadata:

name: git-clone

spec:

workspaces:

- name: output

description: The git repo will be cloned onto the volume backing this Workspace.

- name: ssh-directory *# (1)*

description: | # (2)

A .ssh directory with private key, known\_hosts, config, etc. Copied to

the user's home before git commands are executed. Used to authenticate

with the git remote when performing the clone. Binding a Secret to this

Workspace is strongly recommended over other volume types

params:

- name: url

description: Repository URL to clone from.

type: string

- name: revision

description: Revision to checkout. (branch, tag, sha, ref, etc...)

type: string

default: ""

- name: gitInitImage

description: The image providing the git-init binary that this Task runs.

type: string

default: "gcr.io/tekton-releases/github.com/tektoncd/pipeline/cmd/git-init:v0.37.0"

results:

- name: commit

description: The precise commit SHA that was fetched by this Task.

- name: url

description: The precise URL that was fetched by this Task.

steps:

- name: clone

image: "$(params.gitInitImage)"

script: | # (3)

#!/usr/bin/env sh

set -eu

# This is necessary for recent version of git

git config --global --add safe.directory '\*'

# (2)

if [ "$(workspaces.ssh-directory.bound)" = "true" ] ; then

cp -R "$(workspaces.ssh-directory.path)" "${HOME}"/.ssh

chmod 700 "${HOME}"/.ssh

chmod -R 400 "${HOME}"/.ssh/\*

fi

CHECKOUT\_DIR="$(workspaces.output.path)/"

/ko-app/git-init \

-url="$(params.url)" \

-revision="$(params.revision)" \

-path="${CHECKOUT\_DIR}"

cd "${CHECKOUT\_DIR}"

RESULT\_SHA="$(git rev-parse HEAD)"

EXIT\_CODE="$?"

if [ "${EXIT\_CODE}" != 0 ] ; then

exit "${EXIT\_CODE}"

fi

printf "%s" "${RESULT\_SHA}" > "$(results.commit.path)"

printf "%s" "$(params.url)" > "$(results.url.path)"

1. The name of the workspace, for example, ssh-directory.
2. The description outlines the process for creating the secret.
3. The script that conditionally copies the content of the secret (in the form of a folder) to ${HOME}/.ssh, which is the standard folder where `ssh` searches for credentials. If the `Workspace` is bound, the script performs the copy operation.

### **git-credentials**

In this approach the `git-clone` task includes an optional workspace for `git-credentials`. The script is updated to conditionally copy the contents of the git-credentials` workspace to the user’s home directory if the workspace is bound.

if [ "$(workspaces.basic\_auth.bound)" = "true" ] ; then

cp "$(workspaces.basic\_auth.path)/.git-credentials" "${HOME}/.git-credentials"

cp "$(workspaces.basic\_auth.path)/.gitconfig" "${HOME}/.gitconfig"

chmod 400 "${HOME}/.git-credentials"

chmod 400 "${HOME}/.gitconfig"

fi

## `ServiceAccount`

This approach involves the following steps:

1. Annotating specific secrets using either the GIT or Docker annotation
2. Linking the annotated `Secrets` to `ServiceAccounts`
3. Specifying the `ServiceAccounts` on *PipelineRun* or *TaskRuns*

### Annotations

Tekton offers two types of authentication for secrets, namely GIT and Docker.

* `tekton.dev/git-{}` - Used to mark the annotated secret as a kid secret.
* `tekton.dev/docker-{}` - Used to mark the annotated secret as a docker secret.

Additionally, Tekton supports different types of secrets for each authentication type.

For `git` it supports the following types of secrets:

* kubernetes.io/basic-auth : basic authentication
* kubernetes.io/ssh-auth : ssh based authentication (keys, …)

For `docker` it supports the following types of secrets:

* kubernetes.io/basic-auth : basic authentications
* kubernetes.io/dockercfg : serialized ~/.dockercfg file kubernetes.io/dockerconfigjson : serialized ~/.docker/config.json file

### 

### Linking the annotated `Secrets` to `ServiceAccounts`

To Link a secret to a ServiceAccount, you can reference the Secret by the name in the `**secrets**` field of the ServiceAccount. For example

apiVersion: v1

kind: ServiceAccount

metadata:

name: build-bot

secrets:

- name: regcred

- name: a-git-auth-secret

### Specifying the `ServiceAccounts` on PipelineRun or TaskRuns

To specify a ServiceAccount for a PipelineRun or a TaskRun, you can use the `**serviceAccountName**` field or the `tkn commands.

**Example displaying the use of the `serviceAccountName` field**

*# For a TaskRun*

apiVersion: tekton.dev/v1beta1

kind: TaskRun

metadata:

name: build-with-basic-auth

spec:

serviceAccountName: build-bot

taskRef:

name: demo-task

*# ...*

---

*# For a PipelineRun*

apiVersion: tekton.dev/v1beta1

kind: PipelineRun

metadata:

name: demo-pipeline

namespace: default

spec:

serviceAccountName: build-bot

pipelineRef:

name: demo-pipeline

*# […]*

**Example displaying the use of the `tkn` commands**

---

*# For a TaskRun*

$ tkn task start demo-task --serviceaccount build-bot

*# For a PipelineRun*

$ tkn pipeline start demo-pipeline --serviceaccount build-bot --param *# […]*

—