Red Hat OpenShift Pipelines 1.12 Pipelines as Code

Configuring and using Pipelines as Code
Abstract

This document provides information about configuring and using Pipelines as Code, a subsystem of OpenShift Pipelines that enables defining pipeline templates as part of Git source code repositories.
Table of Contents

CHAPTER 1. ABOUT PIPELINES AS CODE ......................................................... 3
  1.1. KEY FEATURES 3

CHAPTER 2. INSTALLING AND CONFIGURING PIPELINES AS CODE .................. 4
  2.1. INSTALLING PIPELINES AS CODE ON AN OPENShift CONTAINER PLATFORM 4
  2.2. INSTALLING PIPELINES AS CODE CLI 5
  2.3. CUSTOMIZING PIPELINES AS CODE CONFIGURATION 5
  2.4. ADDITIONAL RESOURCES 7

CHAPTER 3. USING PIPELINES AS CODE WITH A GIT REPOSITORY HOSTING SERVICE PROVIDER .... 9
  3.1. USING PIPELINES AS CODE WITH A GITHUB APP 9
    3.1.1. Configuring a GitHub App 9
    3.1.2. Configuring Pipelines as Code to access a GitHub App 10
    3.1.3. Creating a GitHub App in administrator perspective 11
    3.1.4. Scoping the GitHub token to additional repositories 12
  3.2. USING PIPELINES AS CODE WITH GITHUB WEBHOOK 14
  3.3. USING PIPELINES AS CODE WITH GITLAB 18
  3.4. USING PIPELINES AS CODE WITH BITBUCKET CLOUD 21
  3.5. USING PIPELINES AS CODE WITH BITBUCKET SERVER 25
  3.6. INTERFACING PIPELINES AS CODE WITH CUSTOM CERTIFICATES 27
  3.7. USING PRIVATE REPOSITORIES WITH PIPELINES AS CODE 27

CHAPTER 4. USING THE REPOSITORY CUSTOM RESOURCE ..................................... 29
  4.1. CREATING THE REPOSITORY CUSTOM RESOURCE 29
  4.2. SETTING CONCURRENCY LIMITS 29
  4.3. CHANGING THE SOURCE BRANCH FOR THE PIPELINE DEFINITION 30
  4.4. CUSTOM PARAMETER EXPANSION 30

CHAPTER 5. USING THE PIPELINES AS CODE RESOLVER .................................... 33
  5.1. ABOUT THE PIPELINES AS CODE RESOLVER 33
  5.2. USING REMOTE TASK ANNOTATIONS WITH PIPELINES AS CODE 33
  5.3. USING REMOTE PIPELINE ANNOTATIONS WITH PIPELINES AS CODE 35

CHAPTER 6. MANAGING PIPELINE RUNS .......................................................... 36
  6.1. CREATING A PIPELINE RUN USING PIPELINES AS CODE 36
  6.2. RUNNING A PIPELINE RUN USING PIPELINES AS CODE 39
  6.3. MONITORING PIPELINE RUN STATUS USING PIPELINES AS CODE 41
  6.4. CLEANING UP PIPELINE RUN USING PIPELINES AS CODE 42
  6.5. USING INCOMING WEBHOOK WITH PIPELINES AS CODE 43
  6.6. ADDITIONAL RESOURCES 44

CHAPTER 7. PIPELINES AS CODE COMMAND REFERENCE ...................................... 45
  7.1. PIPELINES AS CODE COMMAND REFERENCE 45
    7.1.1. Basic syntax 45
    7.1.2. Global options 45
    7.1.3. Utility commands 45
      7.1.3.1. bootstrap 45
      7.1.3.2. repository 46
      7.1.3.3. generate 46
      7.1.3.4. resolve 47
    7.2. CONFIGURING PIPELINES AS CODE LOGGING 47
    7.3. SPLITTING PIPELINES AS CODE LOGS BY NAMESPACE 50
    7.4. ADDITIONAL RESOURCES 50
CHAPTER 1. ABOUT PIPELINES AS CODE

With Pipelines as Code, cluster administrators and users with the required privileges can define pipeline templates as part of source code Git repositories. When triggered by a source code push or a pull request for the configured Git repository, Pipelines as Code runs the pipeline and reports the status.

1.1. KEY FEATURES

Pipelines as Code supports the following features:

- Pull request status and control on the platform hosting the Git repository.
- GitHub Checks API to set the status of a pipeline run, including rechecks.
- GitHub pull request and commit events.
- Pull request actions in comments, such as /retest.
- Git events filtering and a separate pipeline for each event.
- Automatic task resolution in OpenShift Pipelines, including local tasks, Tekton Hub, and remote URLs.
- Retrieval of configurations using GitHub blobs and objects API.
- Access Control List (ACL) over a GitHub organization or using a Prow style OWNER file.
- The tkn pac CLI plugin for managing bootstrapping and Pipelines as Code repositories.
- Support for GitHub App, GitHub Webhook, Bitbucket Server, and Bitbucket Cloud.
CHAPTER 2. INSTALLING AND CONFIGURING PIPELINES AS CODE

You can install Pipelines as Code as a part of Red Hat OpenShift Pipelines installation.

2.1. INSTALLING PIPELINES AS CODE ON AN OPENSSHIFT CONTAINER PLATFORM

Pipelines as Code is installed in the openshift-pipelines namespace when you install the Red Hat OpenShift Pipelines Operator. For more details, see Installing OpenShift Pipelines in the Additional resources section.

To disable the default installation of Pipelines as Code with the Operator, set the value of the enable parameter to false in the TektonConfig custom resource.

```yaml
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  platforms:
    openshift:
      pipelinesAsCode:
        enable: false
        settings:
          application-name: Pipelines as Code CI
          auto-configure-new-github-repo: "false"
          bitbucket-cloud-check-source-ip: "true"
          hub-catalog-name: tekton
          hub-url: https://api.hub.tekton.dev/v1
          remote-tasks: "true"
          secret-auto-create: "true"

# ...
```

Optionally, you can run the following command:

```
$ oc patch tektonconfig config --type="merge" -p '{"spec": {"platforms": {"openshift":{
          "pipelinesAsCode": {"enable": false}}}}}'
```

To enable the default installation of Pipelines as Code with the Red Hat OpenShift Pipelines Operator, set the value of the enable parameter to true in the TektonConfig custom resource:

```yaml
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  platforms:
    openshift:
      pipelinesAsCode:
        enable: true
        settings:
          application-name: Pipelines as Code CI
```
2.2. INSTALLING PIPELINES AS CODE CLI

Cluster administrators can use the `tkn pac` and `opc` CLI tools on local machines or as containers for testing. The `tkn pac` and `opc` CLI tools are installed automatically when you install the `tkn` CLI for Red Hat OpenShift Pipelines.

You can install the `tkn pac` and `opc` version 1.12.0 binaries for the supported platforms:

- Linux (x86_64, amd64)
- Linux on IBM zSystems and IBM® LinuxONE (s390x)
- Linux on IBM Power (ppc64le)
- Linux on ARM (aarch64, arm64)
- macOS
- Windows

2.3. CUSTOMIZING PIPELINES AS CODE CONFIGURATION

To customize Pipelines as Code, cluster administrators can configure the following parameters in the `TektonConfig` custom resource, in the `pipelinesAsCode.settings` spec:

Table 2.1. Customizing Pipelines as Code configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>application-name</td>
<td>The name of the application. For example, the name displayed in the GitHub Checks labels.</td>
<td>&quot;Pipelines as Code CI&quot;</td>
</tr>
<tr>
<td>secret-auto-create</td>
<td>Indicates whether or not a secret should be automatically created using the token generated in the GitHub application. This secret can then be used with private repositories.</td>
<td>enabled</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>remote-tasks</td>
<td>When enabled, allows remote tasks from pipeline run annotations.</td>
<td>enabled</td>
</tr>
<tr>
<td>hub-url</td>
<td>The base URL for the Tekton Hub API.</td>
<td><a href="https://hub.tekton.dev/">https://hub.tekton.dev/</a></td>
</tr>
<tr>
<td>hub-catalog-name</td>
<td>The Tekton Hub catalog name.</td>
<td>tekton</td>
</tr>
<tr>
<td>tekton-dashboard-url</td>
<td>The URL of the Tekton Hub dashboard. Pipelines as Code uses this URL to generate a <strong>PipelineRun</strong> URL on the Tekton Hub dashboard.</td>
<td>NA</td>
</tr>
<tr>
<td>bitbucket-cloud-check-source-ip</td>
<td>Indicates whether to secure the service requests by querying IP ranges for a public Bitbucket. Changing the parameter’s default value might result into a security issue.</td>
<td>enabled</td>
</tr>
<tr>
<td>bitbucket-cloud-additional-source-ip</td>
<td>Indicates whether to provide an additional set of IP ranges or networks, which are separated by commas.</td>
<td>NA</td>
</tr>
<tr>
<td>max-keep-run-upper-limit</td>
<td>A maximum limit for the <strong>max-keep-run</strong> value for a pipeline run.</td>
<td>NA</td>
</tr>
<tr>
<td>default-max-keep-runs</td>
<td>A default limit for the <strong>max-keep-run</strong> value for a pipeline run. If defined, the value is applied to all pipeline runs that do not have a <strong>max-keep-run</strong> annotation.</td>
<td>NA</td>
</tr>
<tr>
<td>auto-configure-new-github-repo</td>
<td>Configures new GitHub repositories automatically. Pipelines as Code sets up a namespace and creates a custom resource for your repository. This parameter is only supported with GitHub applications.</td>
<td>disabled</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>auto-configure-repo-namespace-template</td>
<td>Configures a template to automatically generate the namespace for your new repository, if <code>auto-configure-new-github-repo</code> is enabled.</td>
<td><code>{repo_name}-pipelines</code></td>
</tr>
<tr>
<td>error-log-snippet</td>
<td>Enables or disables the view of a log snippet for the failed tasks, with an error in a pipeline. You can disable this parameter in the case of data leakage from your pipeline.</td>
<td><code>true</code></td>
</tr>
<tr>
<td>error-detection-from-container-logs</td>
<td>Enables or disables the inspection of container logs to detect error message and expose them as annotations on the pull request. This setting applies only if you are using the GitHub app.</td>
<td><code>true</code></td>
</tr>
<tr>
<td>error-detection-max-number-of-lines</td>
<td>The maximum number of lines inspected in the container logs to search for error messages. Set to <code>-1</code> to inspect an unlimited number of lines.</td>
<td><code>50</code></td>
</tr>
<tr>
<td>secret-github-app-token-scoped</td>
<td>If set to <code>true</code>, the GitHub access token that Pipelines as Code generates using the GitHub app is scoped only to the repository from which Pipelines as Code fetches the pipeline definition. If set to <code>false</code>, you can use both the TektonConfig custom resource and the Repository custom resource to scope the token to additional repositories.</td>
<td><code>true</code></td>
</tr>
<tr>
<td>secret-github-app-scope-extra-repos</td>
<td>Additional repositories for scoping the generated GitHub access token.</td>
<td></td>
</tr>
</tbody>
</table>

2.4. ADDITIONAL RESOURCES

- Installing OpenShift Pipelines
- Installing tkn
- Red Hat OpenShift Pipelines release notes
CHAPTER 3. USING PIPELINES AS CODE WITH A GIT REPOSITORY HOSTING SERVICE PROVIDER

After installing Pipelines as Code, cluster administrators can configure a Git repository hosting service provider. Currently, the following services are supported:

- GitHub App
- GitHub Webhook
- GitLab
- Bitbucket Server
- Bitbucket Cloud

**NOTE**
GitHub App is the recommended service for using with Pipelines as Code.

3.1. USING PIPELINES AS CODE WITH A GITHUB APP

GitHub Apps act as a point of integration with Red Hat OpenShift Pipelines and bring the advantage of Git-based workflows to OpenShift Pipelines. Cluster administrators can configure a single GitHub App for all cluster users. For GitHub Apps to work with Pipelines as Code, ensure that the webhook of the GitHub App points to the Pipelines as Code event listener route (or ingress endpoint) that listens for GitHub events.

**NOTE**
When importing an application using `Import from Git` and the Git repository has a `.tekton` directory, you can configure `pipelines-as-code` for your application.

3.1.1. Configuring a GitHub App

Cluster administrators can create a GitHub App by running the following command:

```
$ tkn pac bootstrap github-app
```

If the `tkn pac` CLI plugin is not installed, you can create the GitHub App manually.

**Procedure**

To create and configure a GitHub App manually for Pipelines as Code, perform the following steps:

1. Sign in to your GitHub account.
2. Go to `Settings → Developer settings → GitHub Apps`, and click `New GitHub App`.
3. Provide the following information in the GitHub App form:
   - **GitHub Application Name**: OpenShift Pipelines
   - **Homepage URL**: OpenShift Console URL
3.1.2. Configuring Pipelines as Code to access a GitHub App

To configure Pipelines as Code to access the newly created GitHub App, execute the following command:

```bash
$ oc -n openshift-pipelines create secret generic pipelines-as-code-secret \
  --from-literal github-private-key="$(cat <PATH_PRIVATE_KEY>)"
```

- **Webhook URL**: The Pipelines as Code route or ingress URL. You can find it by running the following command:
  ```bash
  $ echo https://$(oc get route -n openshift-pipelines pipelines-as-code-controller -o jsonpath='{.spec.host}')
  ```

- **Webhook secret**: An arbitrary secret. You can generate a secret by running the following command:
  ```bash
  $ openssl rand -hex 20
  ```

4. Select the following **Repository permissions**:
   - **Checks**: Read & Write
   - **Contents**: Read & Write
   - **Issues**: Read & Write
   - **Metadata**: Read-only
   - **Pull request**: Read & Write

5. Select the following **Organization permissions**:
   - **Members**: Readonly
   - **Plan**: Readonly

6. Select the following **User permissions**:
   - Check run
   - Issue comment
   - Pull request
   - Push

7. Click **Create GitHub App**

8. On the **Details** page of the newly created GitHub App, note the **App ID** displayed at the top.

9. In the **Private keys** section, click **Generate Private key** to automatically generate and download a private key for the GitHub app. Securely store the private key for future reference and usage.

10. Install the created App on a repository that you want to use with Pipelines as Code.
The path to the private key you downloaded while configuring the GitHub App.

The **App ID** of the GitHub App.

The webhook secret provided when you created the GitHub App.

---

**NOTE**

Pipelines as Code works automatically with GitHub Enterprise by detecting the header set from GitHub Enterprise and using it for the GitHub Enterprise API authorization URL.

### 3.1.3. Creating a GitHub App in administrator perspective

As a cluster administrator, you can configure your GitHub App with the OpenShift Container Platform cluster to use Pipelines as Code. This configuration allows you to execute a set of tasks required for build deployment.

#### Prerequisites

You have installed the Red Hat OpenShift Pipelines `pipelines-1.12` operator from the Operator Hub.

#### Procedure

1. In the administrator perspective, navigate to **Pipelines** using the navigation pane.

2. Click **Setup GitHub App** on the **Pipelines** page.

3. Enter your GitHub App name. For example, `pipelines-ci-clusternam-testui`.

4. Click **Setup**.

5. Enter your Git password when prompted in the browser.

6. Click **Create GitHub App for <username>** where `<username>` is your GitHub user name.

#### Verification

After successful creation of the GitHub App, the OpenShift Container Platform web console opens and displays the details about the application.
The details of the GitHub App are saved as a secret in the `openShift-pipelines` namespace.

To view details such as name, link, and secret associated with the GitHub applications, navigate to **Pipelines** and click **View GitHub App**.

### 3.1.4. Scoping the GitHub token to additional repositories

Pipelines as Code uses the GitHub app to generate a GitHub access token. Pipelines as Code uses this token to retrieve the pipeline payload from the repository and to enable the CI/CD processes to interact with GitHub repositories.

By default, the access token is scoped only to the repository from which Pipelines as Code retrieves the pipeline definition. In some cases, you might want the token to have access to additional repositories. For example, there might be a CI repository where the `.tekton/pr.yaml` file and source payload are located, but the build process defined in `pr.yaml` fetches tasks from a separate private CD repository.

You can extend the scope of the GitHub token in two ways:

- **Global configuration**: You can extend the GitHub token to a list of repositories in different namespaces. You must have administrative permissions to set this configuration.

- **Repository level configuration**: You can extend the GitHub token to a list of repositories that exist in the same namespace as the original repository. You do not need administrative permissions to set this configuration.

**Procedure**

1. In the **TektonConfig** custom resource (CR), in the `pipelinesAsCode.settings` spec, set the `secret-github-app-token-scoped` parameter to `false`. This setting enables scoping the GitHub token to private and public repositories listed in the global and repository level configuration.

2. To set global configuration for scoping the GitHub token, in the **TektonConfig** CR, in the `pipelinesAsCode.settings` spec, specify the additional repositories in the `secret-github-app-scope-extra-repos` parameter, as in the following example:
3. To set repository level configuration for scoping the GitHub token, specify the additional repositories in the `github_app_token_scope_repos` parameter of the `Repository` CR, as in the following example:

```yaml
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: test
namespace: test-repo
spec:
  url: "https://github.com/linda/project"
settings:
  github_app_token_scope_repos:
  - "owner/project"
  - "owner1/project1"
```

In this example, the `Repository` custom resource is associated with the `linda/project` repository in the `test-repo` namespace. The scope of the generated GitHub token is extended to the `owner/project` and `owner1/project1` repositories, as well as the `linda/project` repository. These repositories must exist under the `test-repo` namespace.

**NOTE**

The additional repositories can be public or private, but must reside in the same namespace as the repository with which the `Repository` resource is associated.

If any of the repositories do not exist in the namespace, the scoping of the GitHub token fails with an error message:

```
failed to scope GitHub token as repo owner1/project1 does not exist in namespace test-repo
```

**Result**

The generated GitHub token enables access to the additional repositories that you configured in the global and repository level configuration, as well as the original repository where the Pipelines as Code payload files are located.

If you provide both global configuration and repository level configuration, the token is scoped to all the repositories from both configurations, as in the following example.
TektonConfig custom resource

```yaml
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  platforms:
    openshift:
      pipelinesAsCode:
        enable: true
      settings:
        secret-github-app-token-scoped: false
        secret-github-app-scope-extra-repos: "owner2/project2, owner3/project3"
```

Repository custom resource

```yaml
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: test
namespace: test-repo
spec:
  url: "https://github.com/linda/project"
  settings:
    github_app_token_scope_repos:
      - "owner/project"
      - "owner1/project1"
```

The GitHub token is scoped to the owner/project, owner1/project1, owner2/project2, owner3/project3, and linda/project repositories.

3.2. USING PIPELINES AS CODE WITH GITHUB WEBHOOK

Use Pipelines as Code with GitHub Webhook on your repository if you cannot create a GitHub App. However, using Pipelines as Code with GitHub Webhook does not give you access to the GitHub Check Runs API. The status of the tasks is added as comments on the pull request and is unavailable under the Checks tab.

**NOTE**

Pipelines as Code with GitHub Webhook does not support GitOps comments such as /retest and /ok-to-test. To restart the continuous integration (CI), create a new commit to the repository. For example, to create a new commit without any changes, you can use the following command:

```
$ git --amend -a --no-edit && git push --force-with-lease <origin> <branchname>
```

**Prerequisites**

- Ensure that Pipelines as Code is installed on the cluster.
- For authorization, create a personal access token on GitHub.
To generate a secure and fine-grained token, restrict its scope to a specific repository and grant the following permissions:

Table 3.1. Permissions for fine-grained tokens

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administration</td>
<td>Read-only</td>
</tr>
<tr>
<td>Metadata</td>
<td>Read-only</td>
</tr>
<tr>
<td>Content</td>
<td>Read-only</td>
</tr>
<tr>
<td>Commit statuses</td>
<td>Read and Write</td>
</tr>
<tr>
<td>Pull request</td>
<td>Read and Write</td>
</tr>
<tr>
<td>Webhooks</td>
<td>Read and Write</td>
</tr>
</tbody>
</table>

To use classic tokens, set the scope as `public_repo` for public repositories and `repo` for private repositories. In addition, provide a short token expiration period and note the token in an alternate location.

**NOTE**

If you want to configure the webhook using the `tkn pac` CLI, add the `admin:repo_hook` scope.

Procedure

1. Configure the webhook and create a Repository custom resource (CR).

   - **To configure a webhook and create a Repository CR automatically using the `tkn pac` CLI tool**, use the following command:

     ```bash
     $ tkn pac create repo
     ```

Sample interactive output

```text
? Enter the Git repository url (default: https://github.com/owner/repo):
? Please enter the namespace where the pipeline should run (default: repo-pipelines):
! Namespace repo-pipelines is not found
? Would you like me to create the namespace repo-pipelines? Yes
✓ Repository owner-repo has been created in repo-pipelines namespace
✓ Setting up GitHub Webhook for Repository https://github.com/owner/repo
   I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
? Do you want me to use it? Yes
? Please enter the secret to configure the webhook for payload validation (default: sJNwdmTifHTs): sJNwdmTifHTs
ℹ You now need to create a GitHub personal access token, please checkout the docs at https://docs.github.com/en/authentication/keeping-your-account-and-data-
```
To configure a webhook and create a Repository CR manually, perform the following steps:

i. On your OpenShift cluster, extract the public URL of the Pipelines as Code controller.

```
$ echo https://$(oc get route -n openshift-pipelines pipelines-as-code-controller -o jsonpath='{.spec.host}')
```

ii. On your GitHub repository or organization, perform the following steps:

A. Go to Settings -> Webhooks and click Add webhook.

B. Set the Payload URL to the Pipelines as Code controller public URL.

C. Select the content type as application/json.

D. Add a webhook secret and note it in an alternate location. With openssl installed on your local machine, generate a random secret.

```
$ openssl rand -hex 20
```

E. Click Let me select individual events and select these events: Commit comments, Issue comments, Pull request, and Pushes.

F. Click Add webhook.

iii. On your OpenShift cluster, create a Secret object with the personal access token and webhook secret.

```
$ oc -n target-namespace create secret generic github-webhook-config 
  --from-literal provider.token="<GITHUB_PERSONAL_ACCESS_TOKEN>" 
  --from-literal webhook.secret="<WEBHOOK_SECRET>"
```

iv. Create a Repository CR.

**Example: Repository CR**

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
  url: "https://github.com/owner/repo"
git_provider:
```
NOTE

Pipelines as Code assumes that the OpenShift Secret object and the Repository CR are in the same namespace.

2. Optional: For an existing Repository CR, add multiple GitHub Webhook secrets or provide a substitute for a deleted secret.

   a. Add a webhook using the tkn pac CLI tool.

      Example: Additional webhook using the tkn pac CLI

      ```
      $ tkn pac webhook add -n repo-pipelines
      ```

      Sample interactive output

      ✔ Setting up GitHub Webhook for Repository https://github.com/owner/repo
      I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
      ? Do you want me to use it? Yes
      ? Please enter the secret to configure the webhook for payload validation (default: AeHdHTJVfAeH):  AeHdHTJVfAeH
      ✔ Webhook has been created on repository owner/repo
      Secret owner-repo has been updated with webhook secert in the repo-pipelines namespace.

   b. Update the webhook.secret key in the existing OpenShift Secret object.

3. Optional: For an existing Repository CR, update the personal access token.

   • Update the personal access token using the tkn pac CLI tool.

      Example: Updating personal access token using the tkn pac CLI

      ```
      $ tkn pac webhook update-token -n repo-pipelines
      ```

      Sample interactive output

      ? Please enter your personal access token:  ****************************************
      Secret owner-repo has been updated with new personal access token in the repo-pipelines namespace.

   • Alternatively, update the personal access token by modifying the Repository CR.

      i. Find the name of the secret in the Repository CR.
Use the `oc patch` command to update the values of the `$NEW_TOKEN` in the `$target_namespace` namespace.

```bash
$ oc -n $target_namespace patch secret github-webhook-config -p "{"data":{"provider.token": "$(echo -n $NEW_TOKEN|base64 -w0)"}}"
```

Additional resources

- GitHub Webhook documentation on GitHub
- GitHub Check Runs documentation on GitHub
- Creating a personal access token on GitHub
- Classic tokens with pre-filled permissions

### 3.3. USING PIPELINES AS CODE WITH GITLAB

If your organization or project uses GitLab as the preferred platform, you can use Pipelines as Code for your repository with a webhook on GitLab.

**Prerequisites**

- Ensure that Pipelines as Code is installed on the cluster.
- For authorization, generate a personal access token as the manager of the project or organization on GitLab.

**NOTE**

- If you want to configure the webhook using the `tkn pac` CLI, add the `admin:repo_hook` scope to the token.

- Using a token scoped for a specific project cannot provide API access to a merge request (MR) sent from a forked repository. In such cases, Pipelines as Code displays the result of a pipeline as a comment on the MR.

**Procedure**

1. Configure the webhook and create a **Repository** custom resource (CR).
To configure a webhook and create a Repository CR automatically using the tkn pac CLI tool, use the following command:

```bash
$ tkn pac create repo
```

Sample interactive output

```
? Enter the Git repository url (default: https://gitlab.com/owner/repo):
? Please enter the namespace where the pipeline should run (default: repo-pipelines):
! Namespace repo-pipelines is not found
? Would you like me to create the namespace repo-pipelines? Yes
✓ Repository repositories-project has been created in repo-pipelines namespace
✓ Setting up GitLab Webhook for Repository https://gitlab.com/owner/repo
? Please enter the project ID for the repository you want to be configured, project ID refers to an unique ID (e.g. 34405323) shown at the top of your GitLab project: 17103
  I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
? Do you want me to use it? Yes
? Please enter the secret to configure the webhook for payload validation (default: IFjHIEcaGFIF): IFjHIEcaGFIF
ℹ You now need to create a GitLab personal access token with `api` scope
ℹ Go to this URL to generate one https://gitlab.com/-/profile/personal_access_tokens, see https://is.gd/rOEo9B for documentation
? Please enter the GitLab access token: *************************
? Please enter your GitLab API URL:: https://gitlab.com
✓ Webhook has been created on your repository
   Webhook Secret repositories-project has been created in the repo-pipelines namespace.
   Repository CR repositories-project has been updated with webhook secret in the repo-pipelines namespace.
   Directory .tekton has been created.
   ✓ A basic template has been created in /home/Go/src/gitlab.com/repositories/project/.tekton/pipelinerun.yaml, feel free to customize it.
```

To configure a webhook and create a Repository CR manually, perform the following steps:

i. On your OpenShift cluster, extract the public URL of the Pipelines as Code controller.

```bash
$ echo https://$(oc get route -n openshift-pipelines pipelines-as-code-controller -o jsonpath='{.spec.host}')
```

ii. On your GitLab project, perform the following steps:

   A. Use the left sidebar to go to Settings -> Webhooks.

   B. Set the URL to the Pipelines as Code controller public URL.

   C. Add a webhook secret and note it in an alternate location. With openssl installed on your local machine, generate a random secret.

```bash
$ openssl rand -hex 20
```
D. Click **Let me select individual events** and select these events: **Commit comments**, **Issue comments**, **Pull request**, and **Pushes**.

E. Click **Save changes**.

iii. On your OpenShift cluster, create a **Secret** object with the personal access token and webhook secret.

```bash
$ oc -n target-namespace create secret generic gitlab-webhook-config
   --from-literal provider.token="<GITLAB_PERSONAL_ACCESS_TOKEN>"
   --from-literal webhook.secret="<WEBHOOK_SECRET>
```

iv. Create a **Repository** CR.

**Example: Repository CR**

```yaml
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
  url: "https://gitlab.com/owner/repo" # The repository URL
git_provider:
  #url: "https://gitlab.example.com/"
  secret:
    name: "gitlab-webhook-config"
    key: "provider.token" # Set this if you have a different key in your secret
webhook_secret:
  name: "gitlab-webhook-config"
  key: "webhook.secret" # Set this if you have a different key for your secret
```

1. If you are using a private instance of GitLab and not GitLab.com, uncomment this field and set it to the URL of your GitLab API. The GitLab API is the same host as the repository. For example, if the repository is

   `https://gitlab.example.com/owner/repo`, the API URL is

   `https://gitlab.example.com/`.

**NOTE**

- Pipelines as Code assumes that the OpenShift **Secret** object and the **Repository** CR are in the same namespace.

2. Optional: For an existing **Repository** CR, add multiple GitLab Webhook secrets or provide a substitute for a deleted secret.

   a. Add a webhook using the **tkn pac** CLI tool.

   **Example: Adding additional webhook using the **tkn pac** CLI**

   ```bash
   $ tkn pac webhook add -n repo-pipelines
   ```

   **Sample interactive output**
Update the webhook.secret key in the existing OpenShift Secret object.

3. Optional: For an existing Repository CR, update the personal access token.
   - Update the personal access token using the tkn pac CLI tool.
     
     **Example: Updating personal access token using the tkn pac CLI**
     
     $ tkn pac webhook update-token -n repo-pipelines
     
     **Sample interactive output**
     
     ? Please enter your personal access token: ****************************
     
     Secret owner-repo has been updated with new personal access token in the repo-pipelines namespace.
     
   - Alternatively, update the personal access token by modifying the Repository CR.
     
     i. Find the name of the secret in the Repository CR.
     
     ```
     ... 
     spec: 
     git_provider: 
     secret: 
     name: "gitlab-webhook-config" 
     ...
     ```
     
     ii. Use the oc patch command to update the values of the $NEW_TOKEN in the $target_namespace namespace.
     
     ```
     $ oc -n $target_namespace patch secret gitlab-webhook-config -p 
     "{"data": 
     {"provider.token": "$(echo -n $NEW_TOKEN|base64 -w0)"}, 
     "provider.token": "$(echo -n $NEW_TOKEN|base64 -w0)"}}"
     ```

Additional resources

- GitLab Webhook documentation on GitLab

**3.4. USING PIPELINES AS CODE WITH BITBUCKET CLOUD**

If your organization or project uses Bitbucket Cloud as the preferred platform, you can use Pipelines as Code for your repository with a webhook on Bitbucket Cloud.

**Prerequisites**
• Ensure that Pipelines as Code is installed on the cluster.

• Create an app password on Bitbucket Cloud.
  • Check the following boxes to add appropriate permissions to the token:
    o Account: Email, Read
    o Workspace membership: Read, Write
    o Projects: Read, Write
    o Issues: Read, Write
    o Pull requests: Read, Write

NOTE
  ■ If you want to configure the webhook using the tkn pac CLI, add the Webhooks: Read and Write permission to the token.
  ■ Once generated, save a copy of the password or token in an alternate location.

Procedure

1. Configure the webhook and create a Repository CR.
   • To configure a webhook and create a Repository CR automatically using the tkn pac CLI tool, use the following command:

```bash
$ tkn pac create repo
```

Sample interactive output

? Enter the Git repository url (default: https://bitbucket.org/workspace/repo):
? Please enter the namespace where the pipeline should run (default: repo-pipelines):
! Namespace repo-pipelines is not found
? Would you like me to create the namespace repo-pipelines? Yes
✓ Repository workspace-repo has been created in repo-pipelines namespace
✓ Setting up Bitbucket Webhook for Repository https://bitbucket.org/workspace/repo
? Please enter your bitbucket cloud username: <username>
ℹ You now need to create a Bitbucket Cloud app password, please checkout the docs at https://is.gd/fqMHiJ for the required permissions
? Please enter the Bitbucket Cloud app password: .................................
I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
? Do you want me to use it? Yes
✓ Webhook has been created on repository workspace/repo
✓ Webhook Secret workspace-repo has been created in the repo-pipelines namespace.
✓ Repository CR workspace-repo has been updated with webhook secret in the repo-pipelines namespace
ℹ Directory .tekton has been created.
✓ A basic template has been created in /home/Go/src/bitbucket/repo/.tekton/pipelinerun.yaml, feel free to customize it.
To configure a webhook and create a Repository CR manually, perform the following steps:

i. On your OpenShift cluster, extract the public URL of the Pipelines as Code controller.

$ echo https://$(oc get route -n openshift-pipelines pipelines-as-code-controller -o jsonpath='{.spec.host}')</p>

ii. On Bitbucket Cloud, perform the following steps:

A. Use the left navigation pane of your Bitbucket Cloud repository to go to Repository settings → Webhooks and click Add webhook.

B. Set a Title. For example, "Pipelines as Code".

C. Set the URL to the Pipelines as Code controller public URL.

D. Select these events: Repository: Push, Pull Request: Created, Pull Request: Updated, and Pull Request: Comment created.

E. Click Save.

iii. On your OpenShift cluster, create a Secret object with the app password in the target namespace.

$ oc -n target-namespace create secret generic bitbucket-cloud-token --from-literal provider.token="<BITBUCKET_APP_PASSWORD>"</p>

iv. Create a Repository CR.

Example: Repository CR

```yaml
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
namespace: target-namespace
spec:
  url: "https://bitbucket.com/workspace/repo"
  branch: "main"
  git_provider:
    user: "><BITBUCKET_USERNAME>" 1
    secret:
      name: "bitbucket-cloud-token" 2
      key: "provider.token" # Set this if you have a different key in your secret
```

1 You can only reference a user by the ACCOUNT_ID in an owner file.

2 Pipelines as Code assumes that the secret referred in the git_provider.secret spec and the Repository CR is in the same namespace.
NOTE
- The `tkn pac create` and `tkn pac bootstrap` commands are not supported on Bitbucket Cloud.
- Bitbucket Cloud does not support webhook secrets. To secure the payload and prevent hijacking of the CI, Pipelines as Code fetches the list of Bitbucket Cloud IP addresses and ensures that the webhook receptions come only from those IP addresses.
  - To disable the default behavior, set the `bitbucket-cloud-check-source-ip` parameter to `false` in the `TektonConfig` custom resource, in the `pipelinesAsCode.settings` spec.
  - To allow additional safe IP addresses or networks, add them as comma separated values to the `bitbucket-cloud-additional-source-ip` parameter in the `TektonConfig` custom resource, in the `pipelinesAsCode.settings` spec.

2. Optional: For an existing Repository CR, add multiple Bitbucket Cloud Webhook secrets or provide a substitute for a deleted secret.
   a. Add a webhook using the `tkn pac` CLI tool.

      **Example: Adding additional webhook using the `tkn pac` CLI**

      ```sh
      $ tkn pac webhook add -n repo-pipelines
      
      Sample interactive output
      
      ✓ Setting up Bitbucket Webhook for Repository https://bitbucket.org/workspace/repo
      ? Please enter your bitbucket cloud username: <username>
      I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
      ? Do you want me to use it? Yes
      ✓ Webhook has been created on repository workspace/repo
      Secret workspace-repo has been updated with webhook secret in the repo-pipelines namespace.
      
      NOTE
      Use the `[-n <namespace>]` option with the `tkn pac webhook add` command only when the Repository CR exists in a namespace other than the default namespace.

   b. Update the `webhook.secret` key in the existing OpenShift Secret object.

3. Optional: For an existing Repository CR, update the personal access token.
   - Update the personal access token using the `tkn pac` CLI tool.

      **Example: Updating personal access token using the `tkn pac` CLI**

      ```sh
      $ tkn pac webhook update-token -n repo-pipelines
      ```
Sample interactive output

? Please enter your personal access token: *****************************
Secret owner-repo has been updated with new personal access token in the repo-pipelines namespace.

NOTE

Use the [-n <namespace>] option with the tkn pac webhook update-token command only when the Repository CR exists in a namespace other than the default namespace.

- Alternatively, update the personal access token by modifying the Repository CR.
  
  i. Find the name of the secret in the Repository CR.

  ```
  ...
  spec:
    git_provider:
      user: "<BITBUCKET_USERNAME>"
      secret:
        name: "bitbucket-cloud-token"
        key: "provider.token"
  ...
  ```

  ii. Use the oc patch command to update the values of the $password in the $target_namespace namespace.

  ```
  $ oc -n $target_namespace patch secret bitbucket-cloud-token -p "{"data":{"provider.token": "$(echo -n $NEW_TOKEN|base64 -w0)"}}"
  ```

Additional resources

- Creating app password on Bitbucket Cloud
- Introducing Altassian Account ID and Nicknames

3.5. USING PIPELINES AS CODE WITH BITBUCKET SERVER

If your organization or project uses Bitbucket Server as the preferred platform, you can use Pipelines as Code for your repository with a webhook on Bitbucket Server.

Prerequisites

- Ensure that Pipelines as Code is installed on the cluster.
- Generate a personal access token as the manager of the project on Bitbucket Server, and save a copy of it in an alternate location.
NOTE

- The token must have the `PROJECT_ADMIN` and `REPOSITORY_ADMIN` permissions.
- The token must have access to forked repositories in pull requests.

Procedure

1. On your OpenShift cluster, extract the public URL of the Pipelines as Code controller.

   ```bash
   $ echo https://$(oc get route -n openshift-pipelines pipelines-as-code-controller -o jsonpath='{.spec.host}')
   ```

2. On Bitbucket Server, perform the following steps:
   
a. Use the left navigation pane of your Bitbucket Data Center repository to go to Repository settings –> Webhooks and click Add webhook.
   
b. Set a Title. For example, "Pipelines as Code".
   
c. Set the URL to the Pipelines as Code controller public URL.
   
d. Add a webhook secret and save a copy of it in an alternate location. If you have `openssl` installed on your local machine, generate a random secret using the following command:

   ```bash
   $ openssl rand -hex 20
   ```

   e. Select the following events:

   - Repository: Push
   - Repository: Modified
   - Pull Request: Opened
   - Pull Request: Source branch updated
   - Pull Request: Comment added

   f. Click Save.

3. On your OpenShift cluster, create a `Secret` object with the app password in the target namespace.

   ```bash
   $ oc -n target-namespace create secret generic bitbucket-server-webhook-config \
   --from-literal provider.token="<PERSONAL_TOKEN>" \
   --from-literal webhook.secret="<WEBHOOK_SECRET>"
   ```

4. Create a `Repository` CR.

   **Example: Repository CR**

   ```yaml
   apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
   kind: Repository
   metadata:
   ```
name: my-repo
namespace: target-namespace
spec:
  url: "https://bitbucket.com/workspace/repo"
git_provider:
  url: "https://bitbucket.server.api.url/rest"
  user: "<BITBUCKET_USERNAME>"
  name: "bitbucket-server-webhook-config"
  key: "provider.token"  # Set this if you have a different key in your secret
webhook_secret:
  name: "bitbucket-server-webhook-config"
  key: "webhook.secret"  # Set this if you have a different key for your secret

1. Ensure that you have the right Bitbucket Server API URL without the /api/v1.0 suffix. Usually, the default install has a /rest suffix.

2. You can only reference a user by the ACCOUNT_ID in an owner file.

3. Pipelines as Code assumes that the secret referred in the git_provider.secret spec and the Repository CR is in the same namespace.

NOTE

The tnk pac create and tnk pac bootstrap commands are not supported on Bitbucket Server.

Additional resources

- Creating personal tokens on Bitbucket Server
- Creating webhooks on Bitbucket server

3.6. INTERFACING PIPELINES AS CODE WITH CUSTOM CERTIFICATES

To configure Pipelines as Code with a Git repository that is accessible with a privately signed or custom certificate, you can expose the certificate to Pipelines as Code.

Procedure

- If you have installed Pipelines as Code using the Red Hat OpenShift Pipelines Operator, you can add your custom certificate to the cluster using the Proxy object. The Operator exposes the certificate in all Red Hat OpenShift Pipelines components and workloads, including Pipelines as Code.

Additional resources

- Enabling the cluster-wide proxy

3.7. USING PRIVATE REPOSITORIES WITH PIPELINES AS CODE
Pipelines as Code supports private repositories by creating or updating a secret in the target namespace with the user token. The **git-clone** task from Tekton Hub uses the user token to clone private repositories.

Whenever Pipelines as Code creates a new pipeline run in the target namespace, it creates or updates a secret with the `pac-gitauth-<REPOSITORY_OWNER>-<REPOSITORY_NAME>-<RANDOM_STRING>` format.

You must reference the secret with the **basic-auth** workspace in your pipeline run and pipeline definitions, which is then passed on to the **git-clone** task.

```yaml
... workspace:   - name: basic-auth secret:     secretName: "{{ git_auth_secret }}"
...```

In the pipeline, you can reference the **basic-auth** workspace for the **git-clone** task to reuse:

```yaml
... workspaces:   - name basic-auth params:     - name: repo_url     - name: revision
... tasks:     workspaces:   - name: basic-auth     workspace: basic-auth
... tasks:     - name: git-clone-from-catalog taskRef:       name: git-clone     params:         - name: url         value: $(params.repo_url)         - name: revision         value: $(params.revision)
...```

The **git-clone** task picks up the **basic-auth** workspace and uses it to clone the private repository.

You can modify this configuration by setting the **secret-auto-create** parameter to either a **false** or **true** value, as required, in the **TektonConfig** custom resource, in the **pipelinesAsCode.settings** spec.
CHAPTER 4. USING THE REPOSITORY CUSTOM RESOURCE

The Repository custom resource (CR) has the following primary functions:

- Inform Pipelines as Code about processing an event from a URL.
- Inform Pipelines as Code about the namespace for the pipeline runs.
- Reference an API secret, username, or an API URL necessary for Git provider platforms when using webhook methods.
- Provide the last pipeline run status for a repository.

4.1. CREATING THE REPOSITORY CUSTOM RESOURCE

You can use the `tkn pac` CLI or other alternative methods to create a Repository custom resource (CR) inside the target namespace. For example:

```
cat <<EOF|kubectl create -n my-pipeline-ci -f-
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: project-repository
spec:
  url: "https://github.com/<repository>/<project>"
EOF
```

`my-pipeline-ci` is the target namespace.

Whenever there is an event coming from the URL such as `https://github.com/<repository>/<project>`, Pipelines as Code matches it and then starts checking out the content of the `<repository>/<project>` repository for pipeline run to match the content in the .tekton/ directory.

NOTE

- You must create the Repository CR in the same namespace where pipelines associated with the source code repository will be executed; it cannot target a different namespace.
- If multiple Repository CRs match the same event, Pipelines as Code processes only the oldest one. If you need to match a specific namespace, add the `pipelinesascode.tekton.dev/target-namespace: "<mynamespace>"` annotation. Such explicit targeting prevents a malicious actor from executing a pipeline run in a namespace to which they do not have access.

4.2. SETTING CONCURRENCY LIMITS

You can use the `concurrency_limit` spec in the Repository custom resource definition (CRD) to define the maximum number of pipeline runs running simultaneously for a repository.
If there are multiple pipeline runs matching an event, the pipeline runs that match the event start in an alphabetical order.

For example, if you have three pipeline runs in the `.tekton` directory and you create a pull request with a `concurrency_limit` of 1 in the repository configuration, then all the pipeline runs are executed in an alphabetical order. At any given time, only one pipeline run is in the running state while the rest are queued.

### 4.3. CHANGING THE SOURCE BRANCH FOR THE PIPELINE DEFINITION

By default, when processing a push event or a pull request event, Pipelines as Code fetches the pipeline definition from the branch that triggered the event. You can use the `pipelinerun_provenance` setting in the `Repository` custom resource definition (CRD) to fetch the definition from the default branch configured on the Git repository provider, such as `main`, `master`, or `trunk`.

```yaml
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
  # ...
  concurrency_limit: <number>
# ...
```

**NOTE**

You can use this setting as a security precaution. With the default behaviour, Pipelines as Code uses the pipeline definition in the submitted pull request. With the `default-branch` setting, the pipeline definition must be merged into the default branch before it is run. This requirement ensures maximum possible verification of any changes during merge review.

### 4.4. CUSTOM PARAMETER EXPANSION

You can use Pipelines as Code to expand a custom parameter within your `PipelineRun` resource by using the `params` field. You can specify a value for the custom parameter inside the template of the `Repository` custom resource (CR). The specified value replaces the custom parameter in your pipeline run.

You can use custom parameters in the following scenarios:

- To define a URL parameter, such as a registry URL that varies based on a push or a pull request.
To define a parameter, such as an account UUID that an administrator can manage without necessitating changes to the PipelineRun execution in the Git repository.

**NOTE**

Use the custom parameter expansion feature only when you cannot use the Tekton PipelineRun parameters because Tekton parameters are defined in a Pipeline resource and customized alongside it inside a Git repository. However, custom parameters are defined and customized where the Repository CR is located. So, you cannot manage your CI/CD pipeline from a single point.

The following example shows a custom parameter named `company` in the Repository CR:

```yaml
... 
spec: 
  params: 
    - name: company 
      value: "ABC Company"
... 
```

The value `ABC Company` replaces the parameter name `company` in your pipeline run and in the remotely fetched tasks.

You can also retrieve the value for a custom parameter from a Kubernetes secret, as shown in the following example:

```yaml
... 
spec: 
  params: 
    - name: company 
      secretRef: 
        name: my-secret 
        key: companyname
... 
```

Pipelines as Code parses and uses custom parameters in the following manner:

- If you have a `value` and a `secretRef` defined, Pipelines as Code uses the `value`.
- If you do not have a `name` in the `params` section, Pipelines as Code does not parse the parameter.
- If you have multiple `params` with the same `name`, Pipelines as Code uses the last parameter.

You can also define a custom parameter and use its expansion only when specified conditions were matched for a CEL filter. The following example shows a CEL filter applicable on a custom parameter named `company` when a pull request event is triggered:

```yaml
... 
spec: 
  params: 
    - name: company 
      value: "ABC Company"
  filter: 
    - name: event
... 
```
NOTE

When you have multiple parameters with the same name and different filters, Pipelines as Code uses the first parameter that matches the filter. So, Pipelines as Code allows you to expand parameters according to different event types. For example, you can combine a push and a pull request event.

```python
value:
pac.event_type == "pull_request"
...
CHAPTER 5. USING THE PIPELINES AS CODE RESOLVER

The Pipelines as Code resolver ensures that a running pipeline run does not conflict with others.

5.1. ABOUT THE PIPELINES AS CODE RESOLVER

To split your pipeline and pipeline run, store the files in the .tekton/ directory or its subdirectories.

If Pipelines as Code observes a pipeline run with a reference to a task or a pipeline in any YAML file located in the .tekton/ directory, Pipelines as Code automatically resolves the referenced task to provide a single pipeline run with an embedded spec in a PipelineRun object.

If Pipelines as Code cannot resolve the referenced tasks in the Pipeline or PipelineSpec definition, the run fails before applying any changes to the cluster. You can see the issue on your Git provider platform and inside the events of the target namespace where the Repository CR is located.

The resolver skips resolving if it observes the following type of tasks:

- A reference to a cluster task.
- A task or pipeline bundle.
- A custom task with an API version that does not have a tekton.dev/ prefix.

The resolver uses such tasks literally, without any transformation.

To test your pipeline run locally before sending it in a pull request, use the tkn pac resolve command.

You can also reference remote pipelines and tasks.

5.2. USING REMOTE TASK ANNOTATIONS WITH PIPELINES AS CODE

Pipelines as Code supports fetching remote tasks or pipelines by using annotations in a pipeline run. If you reference a remote task in a pipeline run, or a pipeline in a PipelineRun or a PipelineSpec object, the Pipelines as Code resolver automatically includes it. If there is any error while fetching the remote tasks or parsing them, Pipelines as Code stops processing the tasks.

To include remote tasks, refer to the following examples of annotation:

Reference remote tasks in Tekton Hub

- Reference a single remote task in Tekton Hub.

```yaml
... 
pipelinesascode.tekton.dev/task: "git-clone"  
... 
```

1 Pipelines as Code includes the latest version of the task from the Tekton Hub.

- Reference multiple remote tasks from Tekton Hub

```yaml
... 
pipelinesascode.tekton.dev/task: "[git-clone, golang-test, tkn]" 
... 
```
- Reference multiple remote tasks from Tekton Hub using the `<NUMBER>` suffix.

  ```yaml
  pipelinesascode.tekton.dev/task: "git-clone"
pipelinesascode.tekton.dev/task-1: "golang-test"
pipelinesascode.tekton.dev/task-2: "tkn" ①
  ...
  ```

  ① By default, Pipelines as Code interprets the string as the latest task to fetch from Tekton Hub.

- Reference a specific version of a remote task from Tekton Hub.

  ```yaml
  pipelinesascode.tekton.dev/task: "[git-clone:0.1]" ①
  ...
  ```

  ① Refers to the **0.1** version of the `git-clone` remote task from Tekton Hub.

**Remote tasks using URLs**

```yaml
pipelinesascode.tekton.dev/task: "<https://remote.url/task.yaml>" ①
...
```

① The public URL to the remote task.

**NOTE**

- If you use GitHub and the remote task URL uses the same host as the Repository custom resource definition (CRD), Pipelines as Code uses the GitHub token and fetches the URL using the GitHub API. For example, if you have a repository URL similar to `https://github.com/<organization>/<repository>` and the remote HTTP URL references a GitHub blob similar to `https://github.com/<organization>/<repository>/blob/<mainbranch>/<path>/<file>`, Pipelines as Code fetches the task definition files from that private repository with the GitHub App token.

  When you work on a public GitHub repository, Pipelines as Code acts similarly for a GitHub raw URL such as `https://raw.githubusercontent.com/<organization>/<repository>/<mainbranch>/<path>/<file>`.

- GitHub App tokens are scoped to the owner or organization where the repository is located. When you use the GitHub webhook method, you can fetch any private or public repository on any organization where the personal token is allowed.

**Reference a task from a YAML file inside your repository**
Relative path to the local file containing the task definition.

5.3. USING REMOTE PIPELINE ANNOTATIONS WITH PIPELINES AS CODE

You can share a pipeline definition across multiple repositories by using the remote pipeline annotation.

NOTE
You can reference only one pipeline definition using the annotation.
CHAPTER 6. MANAGING PIPELINE RUNS

Using Pipelines as Code, you can create pipelines in your code repository and run these pipelines.

6.1. CREATING A PIPELINE RUN USING PIPELINES AS CODE

To run pipelines using Pipelines as Code, you can create pipelines definitions or templates as YAML files in the .tekton/ directory of the repository. You can reference YAML files in other repositories using remote URLs, but pipeline runs are only triggered by events in the repository containing the .tekton/ directory.

The Pipelines as Code resolver bundles the pipeline runs with all tasks as a single pipeline run without external dependencies.

NOTE

- For pipelines, use at least one pipeline run with a spec, or a separated Pipeline object.
- For tasks, embed task spec inside a pipeline, or define it separately as a Task object.

Parameterizing commits and URLs

You can specify the parameters of your commit and URL by using dynamic, expandable variables with the {{<var>}} format. Currently, you can use the following variables:

- {{repo_owner}}: The repository owner.
- {{repo_name}}: The repository name.
- {{repo_url}}: The repository full URL.
- {{revision}}: Full SHA revision of a commit.
- {{sender}}: The username or account id of the sender of the commit.
- {{source_branch}}: The branch name where the event originated.
- {{target_branch}}: The branch name that the event targets. For push events, it’s the same as the source_branch.
- {{pull_request_number}}: The pull or merge request number, defined only for a pull_request event type.
- {{git_auth_secret}}: The secret name that is generated automatically with Git provider’s token for checking out private repos.

Matching an event to a pipeline run

You can match different Git provider events with each pipeline by using special annotations on the pipeline run. If there are multiple pipeline runs matching an event, Pipelines as Code runs them in parallel and posts the results to the Git provider as soon a pipeline run finishes.

Matching a pull event to a pipeline run
You can use the following example to match the `pipeline-pr-main` pipeline with a `pull_request` event that targets the `main` branch:

```yaml
... 
metadata:
  name: pipeline-pr-main
annotations:
  pipelinesascode.tekton.dev/on-target-branch: "[main]" # 1
  pipelinesascode.tekton.dev/on-event: "[pull_request]"
...
```

1 You can specify multiple branches by adding comma-separated entries. For example, "[main, release-nightly]". In addition, you can specify the following:

- Full references to branches such as "refs/heads/main"
- Globs with pattern matching such as "refs/heads/*"
- Tags such as "refs/tags/1.*"

### Matching a push event to a pipeline run

You can use the following example to match the `pipeline-push-on-main` pipeline with a `push` event targeting the `refs/heads/main` branch:

```yaml
... 
metadata:
  name: pipeline-push-on-main
annotations:
  pipelinesascode.tekton.dev/on-target-branch: "[refs/heads/main]" # 1
  pipelinesascode.tekton.dev/on-event: "[push]"
...
```

1 You can specify multiple branches by adding comma-separated entries. For example, "[main, release-nightly]". In addition, you can specify the following:

- Full references to branches such as "refs/heads/main"
- Globs with pattern matching such as "refs/heads/*"
- Tags such as "refs/tags/1.*"

### Advanced event matching

Pipelines as Code supports using Common Expression Language (CEL) based filtering for advanced event matching. If you have the `pipelinesascode.tekton.dev/on-cel-expression` annotation in your pipeline run, Pipelines as Code uses the CEL expression and skips the `on-target-branch` annotation. Compared to the simple `on-target-branch` annotation matching, the CEL expressions allow complex filtering and negation.

To use CEL-based filtering with Pipelines as Code, consider the following examples of annotations:

- To match a `pull_request` event targeting the `main` branch and coming from the `wip` branch:
To run a pipeline only if a path has changed, you can use the `.pathChanged` suffix function with a glob pattern:

```plaintext
... pipelinesascode.tekton.dev/on-cel-expression:
  event == "pull_request" && "docs/\*.md".pathChanged() 1
...```

1 Matches all markdown files in the `docs` directory.

- To match all pull requests starting with the title `[DOWNSTREAM]`:

```plaintext
... pipelinesascode.tekton.dev/on-cel-expression:
  event == "pull_request && event_title.startsWith("[DOWNSTREAM]")
...```

- To run a pipeline on a `pull_request` event, but skip the `experimental` branch:

```plaintext
... pipelinesascode.tekton.dev/on-cel-expression:
  event == "pull_request && target_branch != experimental"
...```

For advanced CEL-based filtering while using Pipelines as Code, you can use the following fields and suffix functions:

- **event**: A `push` or `pull_request` event.
- **target_branch**: The target branch.
- **source_branch**: The branch of origin of a `pull_request` event. For `push` events, it is same as the `target_branch`.
- **event_title**: Matches the title of the event, such as the commit title for a `push` event, and the title of a pull or merge request for a `pull_request` event. Currently, only GitHub, Gitlab, and Bitbucket Cloud are the supported providers.
- **.pathChanged**: A suffix function to a string. The string can be a glob of a path to check if the path has changed. Currently, only GitHub and Gitlab are supported as providers.

**Using the temporary GitHub App token for Github API operations**

You can use the temporary installation token generated by Pipelines as Code from GitHub App to access the GitHub API. The token value is stored in the temporary `{{git_auth_secret}}` dynamic variable generated for private repositories in the `git-provider-token` key.

For example, to add a comment to a pull request, you can use the `github-add-comment` task from Tekton Hub using a Pipelines as Code annotation:
You can then add a task to the `tasks` section or `finally` tasks in the pipeline run definition:

```
[...]
tasks:
- name:
  taskRef:
    name: github-add-comment
  params:
    - name: REQUEST_URL
      value: "{{ repo_url }}/pull/{{ pull_request_number }}"
    - name: COMMENT_OR_FILE
      value: "Pipelines as Code IS GREAT!"
    - name: GITHUB_TOKEN_SECRET_NAME
      value: "{{ git_auth_secret }}"
    - name: GITHUB_TOKEN_SECRET_KEY
      value: "git-provider-token"
```  

1 By using the dynamic variables, you can reuse this snippet template for any pull request from any repository.

**NOTE**

On GitHub Apps, the generated installation token is available for 8 hours and scoped to the repository from where the events originate unless configured differently on the cluster.

Additional resources

- CEL language specification

### 6.2. RUNNING A PIPELINE RUN USING PIPELINES AS CODE

With default configuration, Pipelines as Code runs any pipeline run in the `.tekton/` directory of the default branch of repository, when specified events such as pull request or push occurs on the repository. For example, if a pipeline run on the default branch has the annotation `pipelinesascode.tekton.dev/on-event: "[pull_request]"`, it will run whenever a pull request event occurs.

In the event of a pull request or a merge request, Pipelines as Code also runs pipelines from branches other than the default branch, if the following conditions are met by the author of the pull request:

- The author is the owner of the repository.
- The author is a collaborator on the repository.
- The author is a public member on the organization of the repository.
- The pull request author is listed in an `OWNER` file located in the repository root of the main
branch as defined in the GitHub configuration for the repository. Also, the pull request author is added to either approvers or reviewers section. For example, if an author is listed in the approvers section, then a pull request raised by that author starts the pipeline run.

```yaml
... 
  approvers:
    - approved 
...
```

If the pull request author does not meet the requirements, another user who meets the requirements can comment /ok-to-test on the pull request, and start the pipeline run.

**Pipeline run execution**

A pipeline run always runs in the namespace of the Repository custom resource definition (CRD) associated with the repository that generated the event.

You can observe the execution of your pipeline runs using the tkn pac CLI tool.

- To follow the execution of the last pipeline run, use the following example:
  
  ```bash
  $ tkn pac logs -n <my-pipeline-ci> -L 1
  ```

  `my-pipeline-ci` is the namespace for the Repository CRD.

- To follow the execution of any pipeline run interactively, use the following example:
  
  ```bash
  $ tkn pac logs -n <my-pipeline-ci> 1
  ```

  `my-pipeline-ci` is the namespace for the Repository CRD. If you need to view a pipeline run other than the last one, you can use the tkn pac logs command to select a PipelineRun attached to the repository:

If you have configured Pipelines as Code with a GitHub App, Pipelines as Code posts a URL in the Checks tab of the GitHub App. You can click the URL and follow the pipeline execution.

**Restarting a pipeline run**

You can restart a pipeline run with no events, such as sending a new commit to your branch or raising a pull request. On a GitHub App, go to the Checks tab and click Re-run.

If you target a pull or merge request, use the following comments inside your pull request to restart all or specific pipeline runs:

- The /retest comment restarting all pipeline runs.

- The /retest <pipelinerun-name> comment restarting specific pipeline run.

- The /cancel comment cancelling all pipeline runs.

- The /cancel <pipelinerun-name> comment cancelling specific pipeline run.

The results of the comments are visible under the Checks tab of a GitHub App.
6.3. MONITORING PIPELINE RUN STATUS USING PIPELINES AS CODE

Depending on the context and supported tools, you can monitor the status of a pipeline run in different ways.

Status on GitHub Apps

When a pipeline run finishes, the status is added in the Check tabs with limited information on how long each task of your pipeline took, and the output of the `tkn pipelinerun describe` command.

Log error snippet

When Pipelines as Code detects an error in one of the tasks of a pipeline, a small snippet consisting of the last 3 lines in the task breakdown of the first failed task is displayed.

**NOTE**

Pipelines as Code avoids leaking secrets by looking into the pipeline run and replacing secret values with hidden characters. However, Pipelines as Code cannot hide secrets coming from workspaces and envFrom source.

Annotations for log error snippets

In the `TektonConfig` custom resource, in the `pipelinesAsCode.settings` spec, you can set the `error-detection-from-container-logs` parameter to `true`. In this case, Pipelines as Code detects the errors from the container logs and adds them as annotations on the pull request where the error occurred.

**IMPORTANT**

Adding annotations for log error snippets is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope.

Currently, Pipelines as Code supports only the simple cases where the error looks like `makefile` or `grep` output of the following format:

```
<filename>:<line>:<column>: <error message>
```

You can customize the regular expression used to detect the errors with the `error-detection-simple-regexp` parameter. The regular expression uses named groups to give flexibility on how to specify the matching. The groups needed to match are `filename`, `line`, and `error`. You can view the Pipelines as Code config map for the default regular expression.

**NOTE**

By default, Pipelines as Code scans only the last 50 lines of the container logs. You can increase this value in the `error-detection-max-number-of-lines` field or set `-1` for an unlimited number of lines. However, such configurations may increase the memory usage of the watcher.
Status for webhook

For webhook, when the event is a pull request, the status is added as a comment on the pull or merge request.

Failures

If a namespace is matched to a Repository custom resource definition (CRD), Pipelines as Code emits its failure log messages in the Kubernetes events inside the namespace.

Status associated with Repository CRD

The last 5 status messages for a pipeline run is stored inside the Repository custom resource.

```
$ oc get repo -n <pipelines-as-code-ci>
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>URL</th>
<th>NAMESPACE</th>
<th>SUCCEEDED</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipelines-as-code-ci</td>
<td><a href="https://github.com/openshift-pipelines/pipelines-as-code">https://github.com/openshift-pipelines/pipelines-as-code</a></td>
<td>pipelines-as-code-ci</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Succeeded</td>
</tr>
</tbody>
</table>

Using the tkn pac describe command, you can extract the status of the runs associated with your repository and its metadata.

Notifications

Pipelines as Code does not manage notifications. If you need to have notifications, use the finally feature of pipelines.

Additional resources

- An example task to send Slack messages on success or failure

- An example of a pipeline run with finally tasks triggered on push events

Additional resources

- An example of the git-clone task used for cloning private repositories

6.4. CLEANING UP PIPELINE RUN USING PIPELINES AS CODE

There can be many pipeline runs in a user namespace. By setting the max-keep-runs annotation, you can configure Pipelines as Code to retain a limited number of pipeline runs that matches an event. For example:

```
... pipelinesascode.tekton.dev/max-keep-runs: "<max_number>" 1 ...
```

1 Pipelines as Code starts cleaning up right after it finishes a successful execution, retaining only the maximum number of pipeline runs configured using the annotation.
NOTE

- Pipelines as Code skips cleaning the running pipelines but cleans up the pipeline runs with an unknown status.
- Pipelines as Code skips cleaning a failed pull request.

6.5. USING INCOMING WEBHOOK WITH PIPELINES AS CODE

Using an incoming webhook URL and a shared secret, you can start a pipeline run in a repository.

To use incoming webhooks, specify the following within the `spec` section of the `Repository` custom resource definition (CRD):

- The incoming webhook URL that Pipelines as Code matches.
- The Git provider and the user token. Currently, Pipelines as Code supports `github`, `gitlab`, and `bitbucket-cloud`.

NOTE

When using incoming webhook URLs in the context of GitHub app, you must specify the token.

- The target branches and a secret for the incoming webhook URL.

Example: Repository CRD with incoming webhook

```yaml
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: repo
  namespace: ns
spec:
  url: "https://github.com/owner/repo"
  git_provider:
    type: github
  secret:
    name: "owner-token"
  incoming:
    - targets:
      - main
    secret:
      name: repo-incoming-secret
      type: webhook-url
```

Example: The `repo-incoming-secret` secret for incoming webhook

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: repo-incoming-secret
  namespace: ns
```
To trigger a pipeline run located in the .tekton directory of a Git repository, use the following command:

```
$ curl -X POST 'https://control.pac.url/incoming?secret=very-secure-shared-secret&repository=repo&branch=main&pipelinerun=target_pipelinerun'
```

Pipelines as Code matches the incoming URL and treats it as a push event. However, Pipelines as Code does not report status of the pipeline runs triggered by this command.

To get a report or a notification, add it directly with a finally task to your pipeline. Alternatively, you can inspect the Repository CRD with the tkn pac CLI tool.

### 6.6. ADDITIONAL RESOURCES

- An example of the .tekton/ directory in the Pipelines as Code repository
- Creating applications using the Developer perspective
CHAPTER 7. PIPELINES AS CODE COMMAND REFERENCE

You can use the tkn pac CLI tool to control Pipelines as Code. You can also configure Pipelines as Code logging with the TektonConfig custom resource and use the oc command to view Pipelines as Code logs.

7.1. PIPELINES AS CODE COMMAND REFERENCE

The tkn pac CLI tool offers the following capabilities:

- Bootstrap Pipelines as Code installation and configuration.
- Create a new Pipelines as Code repository.
- List all Pipelines as Code repositories.
- Describe a Pipelines as Code repository and the associated runs.
- Generate a simple pipeline run to get started.
- Resolve a pipeline run as if it was executed by Pipelines as Code.

TIP

You can use the commands corresponding to the capabilities for testing and experimentation, so that you don’t have to make changes to the Git repository containing the application source code.

7.1.1. Basic syntax

```
$ tkn pac [command or options] [arguments]
```

7.1.2. Global options

```
$ tkn pac --help
```

7.1.3. Utility commands

7.1.3.1. bootstrap

Table 7.1. Bootstrapping Pipelines as Code installation and configuration

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tkn pac bootstrap</td>
<td>Installs and configures Pipelines as Code for Git repository hosting service providers, such as GitHub and GitHub Enterprise.</td>
</tr>
<tr>
<td>tkn pac bootstrap --nightly</td>
<td>Installs the nightly build of Pipelines as Code.</td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tkn pac bootstrap --route-url &lt;public_url_to_ingress_spec&gt;</td>
<td>Overrides the OpenShift route URL. By default, <code>tkn pac bootstrap</code> detects the OpenShift route, which is automatically associated with the Pipelines as Code controller service. If you do not have an OpenShift Container Platform cluster, it asks you for the public URL that points to the ingress endpoint.</td>
</tr>
<tr>
<td>tkn pac bootstrap github-app</td>
<td>Create a GitHub application and secrets in the <code>openshift-pipelines</code> namespace.</td>
</tr>
</tbody>
</table>

7.1.3.2. repository

Table 7.2. Managing Pipelines as Code repositories

```
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tkn pac create repository</td>
<td>Creates a new Pipelines as Code repository and a namespace based on the pipeline run template.</td>
</tr>
<tr>
<td>tkn pac list</td>
<td>Lists all the Pipelines as Code repositories and displays the last status of the associated runs.</td>
</tr>
<tr>
<td>tkn pac repo describe</td>
<td>Describes a Pipelines as Code repository and the associated runs.</td>
</tr>
</tbody>
</table>

7.1.3.3. generate

Table 7.3. Generating pipeline runs using Pipelines as Code

```
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tkn pac generate</td>
<td>Generates a simple pipeline run. When executed from the directory containing the source code, it automatically detects current Git information. In addition, it uses basic language detection capability and adds extra tasks depending on the language. For example, if it detects a <code>setup.py</code> file at the repository root, the <code>pylint</code> task is automatically added to the generated pipeline run.</td>
</tr>
</tbody>
</table>
7.1.3.4. resolve

Table 7.4. Resolving and executing pipeline runs using Pipelines as Code

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tkn pac resolve</td>
<td>Executes a pipeline run as if it is owned by the Pipelines as Code on service.</td>
</tr>
<tr>
<td>tkn pac resolve -f .tekton/pull-request.yaml</td>
<td>Displays the status of a live pipeline run that uses the template in .tekton/pull-request.yaml.</td>
</tr>
<tr>
<td>oc apply -f -</td>
<td>Combined with a Kubernetes installation running on your local machine, you can observe the pipeline run without generating a new commit.</td>
</tr>
<tr>
<td>tkn pac resolve -f .tekton/pr.yaml -p revision=main -p repo_name=&lt;repository_name&gt;</td>
<td>Executes a pipeline run by overriding default parameter values derived from the Git repository.</td>
</tr>
<tr>
<td></td>
<td>The -f option can also accept a directory path and apply the tkn pac resolve command on all .yaml or .yml files in that directory. You can also use the -f flag multiple times in the same command.</td>
</tr>
<tr>
<td></td>
<td>You can override the default information gathered from the Git repository by specifying parameter values using the -p option. For example, you can use a Git branch as a revision and a different repository name.</td>
</tr>
</tbody>
</table>

7.2. CONFIGURING PIPELINES AS CODE LOGGING

You can configure Pipelines as Code logging by editing the pac-config-logging config map in the TektonConfig custom resource (CR).

Prerequisites

- You have Pipelines as Code installed on your cluster.

Procedure

1. In the Administrator perspective of the web console, go to Administration → CustomResourceDefinitions.
2. Use the Search by name field to search for the tektonconfigs.operator.tekton.dev custom resource definition (CRD) and click TektonConfig to view the CRD Details page.
3. Click the Instances tab.
4. Click the **config** instance to view the **TektonConfig** CR details.

5. Click the **YAML** tab.

6. Edit the **loglevel** fields under the `.options.configMaps.pac-config-logging.data` parameter based on your requirements.

**Example TektonConfig CR with the Pipelines as Code log level fields set to warn**

```yaml
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  platforms:
    openshift:
      pipelinesAsCode:
        options:
          configMaps:
            pac-config-logging:
              data:
                loglevel.pac-watcher: warn ⁠¹
                loglevel.pipelines-as-code-webhook: warn ⁠²
                loglevel.pipelinesascode: warn ³
              zap-logger-config: |
                { ⁠¹
                  "level": "info",
                  "development": false,
                  "sampling": {
                    "initial": 100,
                    "thereafter": 100
                  },
                  "outputPaths": ["stdout"],
                  "errorOutputPaths": ["stderr"],
                  "encoding": "json",
                  "encoderConfig": {
                    "timeKey": "ts",
                    "levelKey": "level",
                    "nameKey": "logger",
                    "callerKey": "caller",
                    "messageKey": "msg",
                    "stacktraceKey": "stacktrace",
                    "lineEnding": "",
                    "levelEncoder": "",
                    "timeEncoder": "iso8601",
                    "durationEncoder": "",
                    "callerEncoder": ""
                  }
                }

 ¹ The log level for the **pipelines-as-code-watcher** component.

 ² The log level for the **pipelines-as-code-webhook** component.

 ³ The log level for the **pipelines-as-code-controller** component.
```
7. Optional: Create a custom logging config map for the Pipelines as Code components by changing the `.env.value` for each component under the `.options.deployments` field. The example below shows the configuration with the custom config map called `custom-pac-config-logging`.

**Example TektonConfig CR with the Pipelines as Code custom logging config map**

```yaml
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  platforms:
    openshift:
      pipelinesAsCode:
        enable: true
        options:
          configMaps:
            custom-pac-config-logging:
              data:
                loglevel.pac-watcher: warn
                loglevel.pipelines-as-code-webhook: warn
                loglevel.pipelinesascode: warn
                zap-logger-config: |
                  |
                    {
                      "level": "info",
                      "development": false,
                      "sampling": {
                        "initial": 100,
                        "thereafter": 100
                      },
                      "outputPaths": ["stdout"],
                      "errorOutputPaths": ["stderr"],
                      "encoding": "json",
                      "encoderConfig": {
                        "timeKey": "ts",
                        "levelKey": "level",
                        "nameKey": "logger",
                        "callerKey": "caller",
                        "messageKey": "msg",
                        "stacktraceKey": "stacktrace",
                        "lineEnding": "",
                        "levelEncoder": "",
                        "timeEncoder": "iso8601",
                        "durationEncoder": "",
                        "callerEncoder": ""
                      }
                    }
              deployments:
                pipelines-as-code-controller:
                  spec:
                    template:
                      spec:
                        containers:
                          - name: pac-controller
                            env:
```

CHAPTER 7. PIPELINES AS CODE COMMAND REFERENCE
7.3. SPLITTING PIPELINES AS CODE LOGS BY NAMESPACE

Pipelines as Code logs contain the namespace information to make it possible to filter logs or split the logs by a particular namespace. For example, to view the Pipelines as Code logs related to the mynamespace namespace, enter the following command:

```
$ oc logs pipelines-as-code-controller-<unique-id> -n openshift-pipelines | grep mynamespace
```

Replace `pipelines-as-code-controller-<unique-id>` with the Pipelines as Code controller name.

7.4. ADDITIONAL RESOURCES

- Installing OpenShift Pipelines
- Installing tkn