System Templates

Overview

System Templates are added and maintained by administrators for an instance of Pipelines install. These templates can be stored in a template source repository, which can then be uploaded to Pipelines. When loaded into Pipelines, these templates can be used in any pipeline like any other resource.

Page Contents

Related Content

Managing Pipelines Templates

Structure of Template Source Repository

The system template files should be stored in the template source repository in a specific order for them to be uploaded to Pipelines:

- There must be a directory called **templates** in the root directory of the repository.
- In the **templates** directory, there must be a subdirectory path of the form: `namespace/<name>/<files>`.

  Where:

  - **namespace** is the name the template belongs to.
  - **name** is the name of the template.
  - **files** are the template-related files.
- The following files can be added as part of the templates:

File	Description	Mandatory /Optional
templateDefinition.yml	This should contain the pipeline definition and should be a valid `yml` file.	Mandatory
values.yml.example	Use this file as a sample for creating a values.yml file. This should be a valid yml file and it is used to specify the details required for templateDefinition.yml file.	Optional
readme.md	This should contain the documentation for the template.	Optional

- Both `templateDefintion.yml` and `values.yml` support the in-built keyword `{{gitBranch}}`. The value of `{{gitBranch}}` is substituted with the branch against which the pipeline source was created. For more information, see Creating Multibranch Pipelines.
- When used in a conditional or comparison logic, `{{gitBranch}}` placeholders must be wrapped within double quotes.
  Example: `{{ if eq "{{gitBranch}}" "master" }}`

templateDefinition.yml: This is a sample templateDefinition.yml. This can be edited to create your own version of the file.

templateDefinition.yml

```
pipelines:
  - name: basic
    steps:
      - name: basic1
        type: Bash
        configuration:
          runtime:
            type: image
        execution:
          onExecute:
            - printenv {{ .Values.foo.bar }}
      - name: basic2
        type: Bash
        configuration:
          runtime:
            type: image
          inputSteps:
            - name: basic1
        execution:
          onExecute:
            - printenv {{ .Values.foo.baz }}
      - name: basic3
        type: Bash
        configuration:
          runtime:
            type: image
          inputSteps:
            - name: basic2
        execution:
          onExecute:
            - printenv {{ .Values.foo.zoo }}
```

values.yml.example: Use this file as a sample to create the `values.yml` file.

values.yml.example

```
artifactoryIntegration: myArtifactoryIntegration

GitRepo:
  name: myGitRepo
  gitProvider: myGitIntegration
  path: myorg/myrepo
  branches:
    include: master


foo:
  name: bar

Image:
  name: myDockerImage
  sourceRepository: mySourceRepo


Pipeline:
  name: myDockerPipeline

DockerBuild:
  name: myDockerBuild
  dockerFileName: Dockerfile
  dockerImageName: <image name>

DockerPush:
  targetRepository: docker-local

Bash:
  name: myBashStep
```

Template Functions List

The pipeline `templateDefinition.yml` supports these helm chart style functions:

- **String Functions**: `trim`, `wrap`, `randAlpha`, `plural`, and others.
  - **String List Functions**: `splitList`, `sortAlpha`, and others.
- **Integer Math Functions**: `add`, `max`, `mul`, and others.
  - **Integer Slice Functions**: `until`, `untilStep`
- **Float Math Functions**: `addf`, `maxf`, `mulf`, and others.
- **Date Functions**: `now`, `date`, and others.
- **Defaults Functions**: `default`, `empty`, `coalesce`, `fromJson`, `toJson`, `toPrettyJson`, `toRawJson`, `ternary`
- **Encoding Functions**: `b64enc`, `b64dec`, and others.
- **Lists and List Functions**: `list`, `first`, `uniq`, and others.
- **Dictionaries and Dict Functions**: `get`, `set`, `dict`, `hasKey`, `pluck`, `dig`, `deepCopy`, and others.
- **Type Conversion Functions**: `atoi`, `int64`, `toString`, and others.
- **Path and Filepath Functions**: `base`, `dir`, `ext`, `clean`, `isAbs`, `osBase`, `osDir`, `osExt`, `osClean`, `osIsAbs`
- **Flow Control Functions**: `fail`
- Advanced Functions
  - **UUID Functions**: `uuidv4`
  - **Version Comparison Functions**: `semver`, `semverCompare`
  - **Reflection**: `typeOf`, `kindIs`, `typeIsLike`, and others.
  - **Cryptographic and Security Functions**: `derivePassword`, `sha256sum`, `genPrivateKey`, and others.
  - **Network**: `getHostByName`

For more information about these functions, see Sprig Function Documentation and Template Functions.

Unsupported Functions

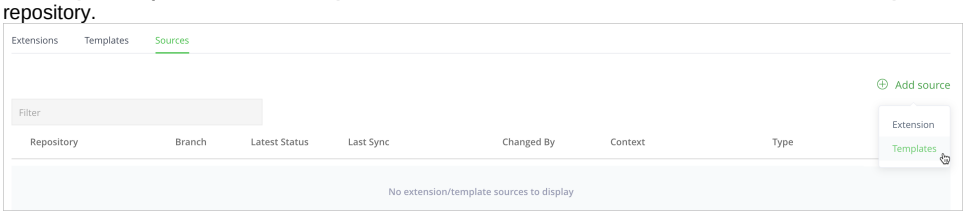
The following functions are not supported:

- `tpl`
- `toToml`
- `toYaml`
- `fromYaml`
- `fromYamlArray`
- `toJson`
- `fromJson`
- `fromJsonArray`
- `lookup`
- `expandenv`
- `env`

Adding System Templates

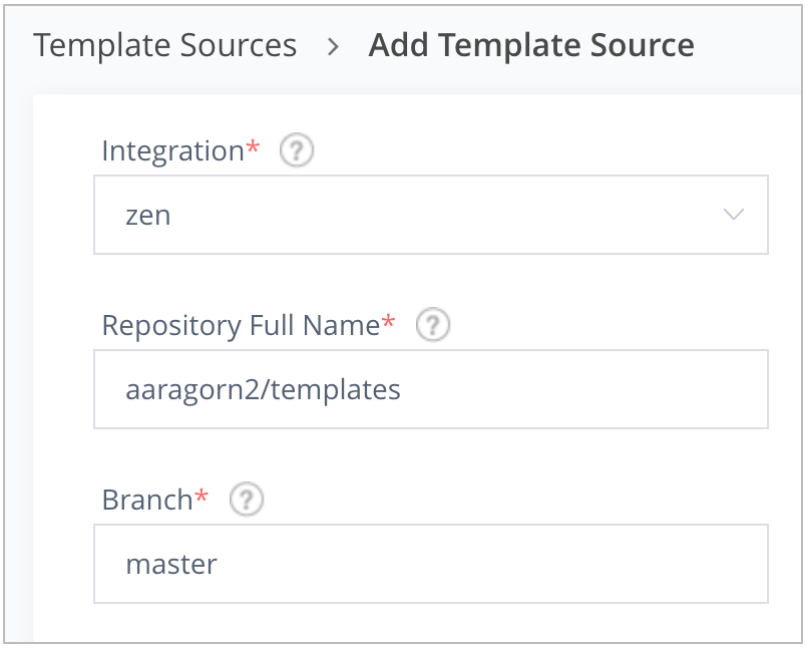
Perform the following steps to add system templates:

1. Ensure that templateDefinition.yml and values.yml files are available in the appropriate repository.
2. **Add an admin project integration**:
   System templates are uploaded to pipelines through an SCM repository. To add the repository, add a SCM admin integration from **Administration | Pipelines | Integrations**. For more information, see Administering Integrations.
3. **Add a template source**:
   Go to **Pipelines | Extensions & Templates** and click the **Sources** tab. Click **Add Source** and **Templates** to add the template source repository.



4. Select the admin integration, add the repository name, and branch and click **Create Source** to create the template source.

> The admin integration should have admin access to the template source repository.



Pipelines adds all the templates from the template source and performs a sync, and the newly added or updated templates are available in Pipelines as the latest version. If there is an error during the sync, it fails.

These templates are also uploaded to Artifactory.

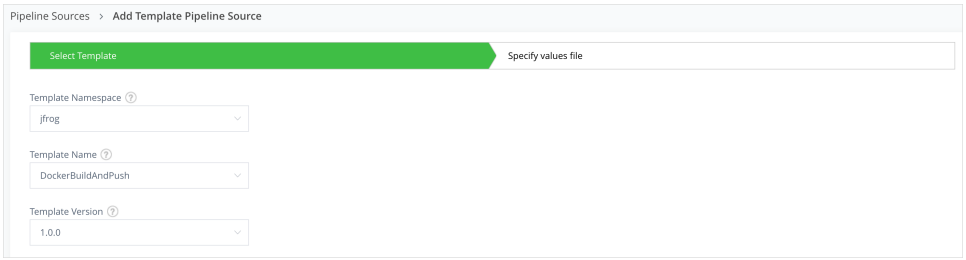
Managing Templates

The templates from the newly uploaded template source is now available for creating pipelines. For information about managing, retiring, and git-tagging and releasing templates, see Managing Pipelines Templates.

Using System Templates to Create a Pipeline Source

Perform the following steps to use a system template to create a pipeline source:

1. Go to **Administration | Pipelines | Pipeline Sources**, and click **Add Pipeline Source**, and **From Template**.
2. Complete the resulting **Template Properties** form:

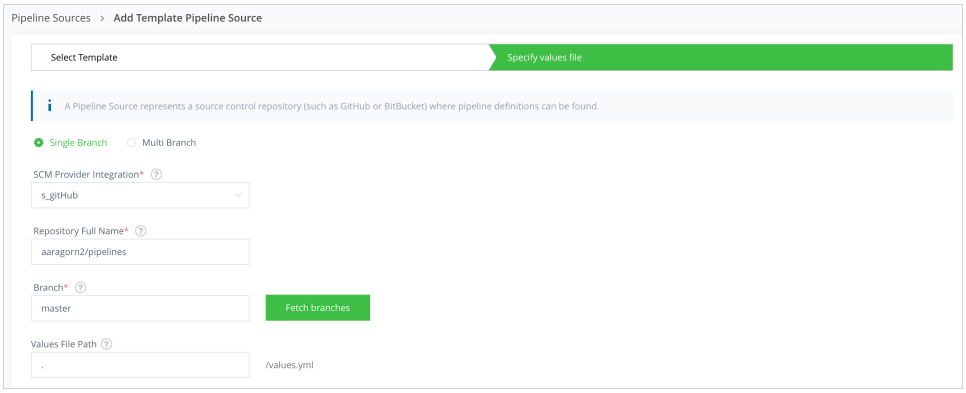


   - Click the *Select Template Namespace* field and select the namespace where the required system template is defined.
   - Click the *Select Template Name* field and select the relevant system template.
   - Click the *Select Template Version* field and select the relevant version for the template.

3. Click **Next** and complete the resulting **Specify values file** form to add the `value.yml` file.

   In the **Values File Path** field, provide the path to the values.yml file, which contains the values for the system template. By default, the field is set to take the file from the root directory.

   For more information about the **Specify values file** form, see the section *Adding a Pipeline Source* in Administering Pipeline Sources.



4. Click **Create Source** to complete adding the Pipeline Source.

The combination of pipeline template name, namespace, version and `values.yml` is parsed to create the pipeline definition.

After your Pipeline Source syncs successfully, you can view the newly added pipeline by navigating to the My Pipelines on the left navbar and clicking on **Pipelines My Pipelines**.