Pipelines Extension Step Model

Overview

Pipelines enables creation of user-defined steps to extend the Pipelines DSL.

Steps are discrete units of execution in a pipeline. The Pipelines DSL defines two sets of built-in step type:

- **Generic steps** - For general-purpose execution, execute any series of shell commands for the supported runtime. For example: Bash, PowerShell.
- **Native steps** - Perform a specific set of actions as an encapsulated unit. For example, DockerPush, HelmDeploy.

Extension steps enable Pipelines users to extend the Pipelines DSL by specifying their own step types that, like native steps, perform an encapsulated action. When loaded into Pipelines, these user-defined steps can be used in any pipeline just like any other step in the Pipelines DSL. In this way, teams and organizations can create and share their own re-usable, custom step types for actions frequently perform in their pipelines.

Extension steps are defined within a namespace, to ensure that all of steps in the set have unique names.

Extension steps are versioned, and can be invoked in pipelines by their semantic version number to help assure compatibility.

Page Contents

Files

Extension step definitions must be stored in a subdirectory path of the form: `steps/<namespace>/<stepTypeName>`.

- `namespace` is the namespace for the set of extension sets. This parent subdirectory may contain multiple step definition subdirectories.
- `stepTypeName` is the named `type` of the step. Must be alphabetic characters only, and is case-sensitive.

The subdirectory can hold the following files to define the step:

File	Description	Required/Optional
`stepModel.yml`	Syntax model for the step.	Required
`onExecute.sh` and/or `onExecute.ps1`	Shell script to be executed in the `onExecute` block. No user commands will be permitted.	`onExecute.sh` is required if: • Platforms tag is not set, in which case the default OS is `Linux`. or • Platforms tag is set to `Linux`. `onExecute.ps1` is required if Platforms tag is set to `Windows`.
`onSuccess.sh` and/or `onSuccess.ps1`	Shell script to be executed in the `onSuccess` block, in advance of user commands.	Optional
`onFailure.sh` and/or `onFailure.ps1`	Shell script to be executed in the `onFailure` block, in advance of user commands.	Optional

`onComplete.sh` and/or `onComplete.ps1`	Shell script to be executed in the `onComplete` block, in advance of user commands.	Optional
`ReadMe.md`	Documentation for the extension step	Optional
`icon.svg`	Icon graphic to represent the step type in the interactive diagram. If not provided, Pipelines will use the default icon for the step.	Optional

Extension step definitions are loaded from the source repository when it is configured in the Pipelines UI as an extension source.

For information on administering extension sources and extension version lifecycle staging, see Managing Pipelines Extensions.

Syntax Model

This is the syntax model for the step.

stepModel.yml

```
description: <string>                    # User can provide an optional description

platforms:                                                # optional
  - os: Linux
  - os: Windows

configuration:                  # array of properties
  <property name string>:
    type: <data type>           # required
        required: <boolean>                       # optional
    validate:                   # optional
      <validation specification>

   # more property definitions


userDefinedDataType:             # array of data type definitions
  - type: <string>               # Defines a new data type
    configuration:
      - <string>:                # Specifies a property of the data type
        type: <data type>           # required
                required: <boolean>                       # optional
        validate:                   # optional
          <validation specification>

   # more data type property definitions
```

Tags

You can define the following tags in the `stepModel.yml` file.

- description
- platforms
- configuration
- userDefinedDataType

description

A user-friendly description of the resource's function that will be available for display in the Pipelines UI. This is optional.

plaforms

Defines the operating system for the node where the step will execute. Linux and Windows operating systems are supported.

This tag is optional and Linux is the default operating system when this tag is not specified.

Based on the operating system, respective shell scripts (Bash and/or PowerShell) must be available. For Linux, `onExecute.sh` and for Windows, `onExecute.ps1` scripts must be available. If both operating systems are mentioned, then both the scripts must be available.

Tag	Description of usage	Required/Optional
`os`	Specifies the operating system. Linux and Windows are supported.	Optional

os Examples

```
platforms:                                                # optional
  - os: Linux
  - os: Windows
```

configuration

A step extension inherits all of the same tags as the generic Bash step to specify scoped environment variables, runtimes, node pools, and other standard properties. It also inherits the tags for `integrations` , `inputsteps`, `inputresources` and `outputresources.` In addition, a step extension can define other properties that will be unique to the step.

The `configuration` tag begins a block of property definitions. Each property definition begins with the name of the property (letters only, case-sensitive), followed by these subordinate tags:

Tag	Description of usage	Required/Optional
`type`	Specifies an inbuilt or user-defined data type	Required
required	When set as `true`, specifies that the property is mandatory. If no value is provided, pipeline sync will fail. Default is `false`.	Optional
`validate`	Begins a validation specification block	Optional

configuration Examples

```
configuration:
  healthCheckUrl:
    type: Url
        required: true;
  notifyOnSuccess:
    type: Boolean
  notifyOnFailure:
    type: Boolean
  stuff:
    type: String[]
    validate:
      isRegex:  ["^[a-z]+$",'i']    # Each element in array is validated with this
```

userDefinedDataType

Begins a set of user-defined data type specifications. User-defined data types specify property types with validation rules, and can be used as `type` values in the step's property definitions.

The tag syntax is the same as used for the Resource Model.

Shell Scripts

An extension step definition may include optional Bash shell or PowerShell scripts to be executed for the appropriate tags in step's the `execution` blog. These scripts define the operation of the step.

onExecute.sh/onExecute.ps1

When present in the step definition's repository directory, the `onExecute.sh/onExecute.ps1` script is executed as the step's `onExecute` block. This should perform the primary function of the step.

The script must return a `true` value if the step succeeded, or `false` if it fails.

onExecute.sh Example

```
checkHealth() {
  local success=true
  local url=$(find_step_configuration_value "healthCheckUrl")
  {
    local statusCode=$(curl --silent --output /dev/stderr --write-out "%{http_code}" "$url")
  } || exitCode=$?
  if test $statusCode -ne 200; then
    export success=false
  fi

  $success
}

execute_command checkHealth
```

onSuccess.sh/onSuccess.ps1

When present in the step definition's repository directory, the `onSuccess.sh/onSuccess.ps1` script is executed as part of the step's `onSuccess` block, in advance of user commands.

This script is executed when the `onExecute` script returns `true.`

onSuccess.sh Example

```
sendSuccessNotification() {
  local notifyOnSuccess=$(find_step_configuration_value "notifyOnSuccess")
  if [ -z "$notifyOnSuccess" ]; then
    notifyOnSuccess=false
  fi
  if [ "$notifyOnSuccess" == "true" ]; then
    echo "Health check succeeded"
  fi
}

execute_command sendSuccessNotification
```

onFailure.sh/onFailure.ps1

When present in the step definition's repository directory, the `onFailure.sh/onFailure.ps1` script is executed as part of the step's `onFailure` block, in advance of user commands.

This script will be executed when the `onExecute` script returns `false.`

onFailure.sh Example

```
sendFailNotification() {
  local notifyOnFailure=$(find_step_configuration_value "notifyOnFailure")
  if [ -z "$notifyOnFailure" ]; then
    notifyOnFailure=false
  fi
  if [ "$notifyOnFailure" == "true" ]; then
    echo "Health check failed"
  fi
}

execute_command sendFailNotification
```

onComplete.sh/onComplete.ps1

When present in the step definition's repository directory, the `onComplete.sh/onComplete.ps1` script is executed as part of the step's `onComplete` block, in advance of user commands.

For example:

onComplete.sh Example

```
echo "All done!"
```