

Deployit Trigger Plugin Manual

Version 3.8.5

Table of Contents

Table of Contents	2
Preface	3
Requirements	3
Actions	3
Email Action	3
State Transitions	4
Task state transitions	4
Step state transitions	4
CI Reference	5
Configuration Item Overview	5
Other Configuration Items	5
Configuration Item Details	5
mail.SmtpServer	5
trigger.EmailNotification	5
trigger.StepTrigger	6
trigger.TaskTrigger	6
trigger.Trigger	6

Preface

The Trigger plugin allows you to configure Deployit to send emails for certain events. For example, you can add rules to send an email whenever a step fails, or when a deployment has completed successfully.

When Deployit resolves a deployment plan, a task comprising of multiple steps is created and subsequently executed. The task, along with the steps, transition through various states before finally completing. The Trigger Plugin allows a user to associate actions, example sending an email, to be triggered for one of these state transitions.

Requirements

This plugin is bundled with Deployit from version 3.8 and has no additional requirements.

Actions

With the trigger plugin, you can define notification actions for certain events.

The following Deployit objects are made available to the actions:

- **Deployed Application:** The entire deployed application containing application and environment configuration items. See UDM CI reference for [udm.DeployedApplication](#).
- **Task:** Contains information about the task. The following properties are available,
 - id
 - state
 - description
 - startDate
 - completionDate
 - nrSteps : number of steps in the task.
 - currentStepNr : the current step been executed.
 - failureCount : number of times the task has failed
 - owner
 - steps : List of steps in the task. Not available when action triggered from StepTrigger. See step information below for properties.
- **Step:** Contains information about a step. Not available when action triggered from TaskTrigger. The following properties are available,
 - description
 - state
 - log
 - startDate
 - completionDate
 - failureCount
- **Action:** Reference to the executing action itself.

Email Action

This section describes how to configure an email action.

First, you will need to define a [mail.SmtpServer](#) CI under the *Configuration* root. See the **Generic Model Plugin Manual** for more details.

The [trigger.EmailNotification](#) CI is used to define the message template for the emails that will be sent.

Under the *Configuration* root, define a `trigger.EmailNotification` configuration item. In the CLI you can do something like:

```
myEmailAction = factory.configurationItem("Configuration/MyFailedDeploymentNotification",
    "trigger.EmailNotification")
myEmailAction.mailServer = "Configuration/MailServer"
myEmailAction.subject = "Application ${deployedApplication.version.application.name} failed."
myEmailAction.toAddresses = ["support@mycompany.com"]
myEmailAction.body = "Deployment of ${deployedApplication.version.application.name} was cancelled on environment
    ${deployedApplication.environment.name}"
repository.create(myEmailAction)
```

The *subject*, *toAddresses*, *fromAddress*, *body* properties accept [FreeMarker template syntax](#) and can access the following Deployit objects:

- `${deployedApplication}`
- `${task}`
- `${step}`

For example, `${deployedApplication.version.application.name}` refers to the name of the application being deployed.

The email body can also be defined in an external template file. Set the path to the file in the `bodyTemplatePath` property. This can be either an absolute path, or a relative path that will be resolved via Deployit's classpath. By specifying a relative path, Deployit will look in the `ext` directory of the Deployit Server and in all (packaged) plugin jar files.

State Transitions

To enable a trigger for deployments, add it to the `triggers` property of an Environment. The trigger will then listen to state transitions in tasks and steps that occur during a deployment. When the state transition described by the trigger matches, the associated actions are executed.

Deployit 3.8 is shipped with the EmailNotification trigger. Custom trigger actions can be written in Java.

Task state transitions

From	To	Description
Queued	Pending	Thread becomes available to execute task.
Pending	Executing	Task starts to execute.
Executing	Stopped	Task is stopped due to step failure, pause step or user aborted step.
Executing	Executed	Task has executed successfully.
Executed	Done	Task is stored in Deployit's archive.
Stopped	Executing	User continues the execution of a failed, paused or aborted step.
Stopped	Cancelled	User cancels task.

A [TaskTrigger](#) can be defined under the *Configuration* root and associated with the environment on which it should be triggered.

```
taskTrigger = factory.configurationItem("Configuration/TriggerOnCancel","trigger.TaskTrigger")
taskTrigger.fromState = "ANY"
taskTrigger.toState = "CANCELLED"
taskTrigger.actions = [myEmailAction]
repository.create(taskTrigger)

env = repository.read("Environments/Dev")
env.triggers = ["Configuration/TriggerOnCancel"]
repository.update(env)
```

Step state transitions

From	To	Description
Pending	Executing	Step starts to execute.
Pending	Skip	User marks pending step to be skipped.
Skip	Pending	Step is unskipped.
Skip	Skipped	Step was skipped during execution.
Executing	Done	Step completed executing successfully.
Executing	Failed	Step failed during execution.
Executing	Paused	Step paused its execution.
Paused	Skip	User marks paused step to be skipped.
Paused	Executing	User continues the execution of paused step.
Failed	Skip	User marks failed step to be skipped.
Failed	Executing	User retries the execution of failed step.

A [StepTrigger](#) can be defined under the *Configuration* root and associated with the

environment on which it should be triggered.

```
stepTrigger = factory.configurationItem("Configuration/TriggerOnFailure", "trigger.StepTrigger")
stepTrigger.fromState = "EXECUTING"
stepTrigger.toState = "FAILED"
stepTrigger.actions = [myEmailAction]
repository.create(stepTrigger)

env = repository.read("Environments/Dev")
env.triggers = ["Configuration/TriggerOnFailure"]
repository.update(env)
```

CI Reference

Configuration Item Overview

Other Configuration Items

CI	Description
mail.SmtplibServer	SMTP Mail Server Configuration
trigger.EmailNotification	Email Action
trigger.StepTrigger	Defines actions to executed for the specified state transition of a Step
trigger.TaskTrigger	Defines actions to executed for the specified state transition of a Task
trigger.Trigger	Trigger with associated actions

Configuration Item Details

mail.SmtplibServer

Type Hierarchy udm.BaseConfigurationItem

Interfaces udm.ConfigurationItem

SMTP Mail Server Configuration

Public Properties	
* fromAddress : STRING	Default from address to use for messages sent with this server.
* host : STRING	SMTP host
* port : INTEGER = 25	SMTP port
password : STRING	Password to authenticate with host
smtpProperties : MAP_STRING_STRING	Refer to http://javamail.kenai.com/nonav/javadocs/com/sun/mail/smtp/package-summary.html for all properties that can be used.
testAddress : STRING	The address to which a test mail is sent when using the 'Send Test Mail' control task.
username : STRING	Username to authenticate with host
Control Tasks	
sendTestMail	no description

trigger.EmailNotification

Type Hierarchy udm.BaseConfigurationItem

Interfaces trigger.Action, udm.ConfigurationItem

Email Action

Public Properties
* mailServer : <code>CI<mail.SmtpServer ></code> The mail server used to send the email.
* subject : <code>STRING</code> Mail subject
* toAddresses : <code>LIST_OF_STRING</code> Mail addresses of recipients.
body : <code>STRING</code> Mail body content in the form of a Freemarker template.
bodyTemplatePath : <code>STRING</code> Freemarker template used to render mail body content. Path can be absolute or relative to Deployit's classpath.
fromAddress : <code>STRING</code> From mail address. Defaults to SMTPServer fromAddress.

trigger.StepTrigger

Type Hierarchy [trigger.Trigger](#) >> `udm.BaseConfigurationItem`

Interfaces `udm.ConfigurationItem`

Defines actions to executed for the specified state transition of a Step.

Public Properties
* actions : <code>LIST_OF_CI<trigger.Action></code> Actions to execute when specified state transition occurs.
* fromState : <code>ENUM [ANY, PENDING, SKIP, EXECUTING, DONE, FAILED, PAUSED, SKIPPED] = ANY</code> Trigger actions when the Step transitions from this state.
* toState : <code>ENUM [ANY, PENDING, SKIP, EXECUTING, DONE, FAILED, PAUSED, SKIPPED]</code> Trigger actions when the Step transitions to this state.

trigger.TaskTrigger

Type Hierarchy [trigger.Trigger](#) >> `udm.BaseConfigurationItem`

Interfaces `udm.ConfigurationItem`

Defines actions to executed for the specified state transition of a Task.

Public Properties
* actions : <code>LIST_OF_CI<trigger.Action></code> Actions to execute when specified state transition occurs.
* fromState : <code>ENUM [ANY, QUEUED, PENDING, EXECUTING, DONE, STOPPED, EXECUTED, CANCELLED] = ANY</code> Trigger actions when the Task transitions from this state.
* toState : <code>ENUM [ANY, QUEUED, PENDING, EXECUTING, DONE, STOPPED, EXECUTED, CANCELLED]</code> Trigger actions when the Task transitions to this state.

trigger.Trigger

Virtual Type

Type Hierarchy `udm.BaseConfigurationItem`

Interfaces `udm.ConfigurationItem`

Trigger with associated actions.

Public Properties
* actions : <code>LIST_OF_CI<trigger.Action></code> Actions to execute when specified state transition occurs.