# Pipeline

A pipeline describes a workflow operation in Zuul. It associates jobs for a given project with triggering and reporting events.

Its flexible configuration allows for characterizing any number of workflows, and by specifying each as a named configuration, makes it easy to apply similar workflow operations to projects or groups of projects.

By way of example, one of the primary uses of Zuul is to perform project gating. To do so, one can create a gate pipeline which tells Zuul that when a certain event (such as approval by a code reviewer) occurs, the corresponding change or pull request should be enqueued into the pipeline. When that happens, the jobs which have been configured to run for that project in the gate pipeline are run, and when they complete, the pipeline reports the results to the user.

Here is an example check pipeline, which runs whenever a new patchset is created in Gerrit. If the associated jobs all report success, the pipeline reports back to Gerrit with Verified vote of +1, or if at least one of them fails, a -1:

- **pipeline**:

**name**: check

**manager**: independent



**trigger**:



**my\_gerrit**:

- **event**: patchset-created



**success**:

**my\_gerrit**:

**Verified**: 1

**failure**:

**my\_gerrit**:

**Verified**: -1

**pipeline**

The attributes available on a pipeline are as follows (all are optional unless otherwise specified):



**pipeline.name*(required)***



This is used later in the project definition to indicate what jobs should be run for events in the pipeline.

**pipeline.manager*(required)***



There are several schemes for managing pipelines. The following table summarizes their features; each is described in detail below.

| **Manager** | **Use Case** | **Dependencies** | **Merge** | **Shared Queues** | **Window** |
| --- | --- | --- | --- | --- | --- |
| Independent | check, post | No | No | No | Unlimited |
| Dependent | gate | Yes | Yes | Yes | Variable |
| Serial | deploy | No | No | Yes | 1 |
| Supercedent | post, promote | No | No | Project-ref | 1 |

**independent**



Every event in this pipeline should be treated as independent of other events in the pipeline. This is appropriate when the order of events in the pipeline doesn’t matter because the results of the actions this pipeline performs can not affect other events in the pipeline. For example, when a change is first uploaded for review, you may want to run tests on that change to provide early feedback to reviewers. At the end of the tests, the change is not going to be merged, so it is safe to run these tests in parallel without regard to any other changes in the pipeline. They are independent.

Another type of pipeline that is independent is a post-merge pipeline. In that case, the changes have already merged, so the results can not affect any other events in the pipeline.

**dependent**



The dependent pipeline manager is designed for gating. It ensures that every change is tested exactly as it is going to be merged into the repository. An ideal gating system would test one change at a time, applied to the tip of the repository, and only if that change passed tests would it be merged. Then the next change in line would be tested the same way. In order to achieve parallel testing of changes, the dependent pipeline manager performs speculative execution on changes. It orders changes based on their entry into the pipeline. It begins testing all changes in parallel, assuming that each change ahead in the pipeline will pass its tests. If they all succeed, all the changes can be tested and merged in parallel. If a change near the front of the pipeline fails its tests, each change behind it ignores whatever tests have been completed and are tested again without the change in front. This way gate tests may run in parallel but still be tested correctly, exactly as they will appear in the repository when merged.



For more detail on the theory and operation of Zuul’s dependent pipeline manager, see: Project Gating.

**serial**

This pipeline manager supports shared queues (like depedent pipelines) but only one item in each shared queue is processed at a time.

This may be useful for post-merge pipelines which perform partial production deployments (i.e., there are jobs with file matchers which only deploy to affected parts of the system). In such a case it is important for every change to be processed, but they must still be processed one at a time in order to ensure that the production system is not inadvertently regressed. Support for shared queues ensures that if multiple projects are involved deployment runs still execute sequentially.

**pipeline.description**

This field may be used to provide a textual description of the pipeline. It may appear in the status page or in documentation.

**pipeline.success-message   
Default:Build successful.**

The introductory text in reports when all the voting jobs are successful.

**pipeline.failure-message   
Default:Build failed.**

The introductory text in reports when at least one voting job fails.

**pipeline.start-message   
Default:Starting {pipeline.name} jobs.**

The introductory text in reports when jobs are started. Three replacement fields are available status\_url, pipeline and change.

**pipeline.enqueue-message**

The introductory text in reports when an item is enqueued. Empty by default.

**pipeline.merge-conflict-message   
Default:Merge failed.**

The introductory text in the message reported when a change fails to merge with the current state of the repository. Defaults to “Merge failed.”

**pipeline.no-jobs-message**

The introductory text in reports when an item is dequeued without running any jobs. Empty by default.

**pipeline.trigger**

At least one trigger source must be supplied for each pipeline. Triggers are not exclusive – matching events may be placed in multiple pipelines, and they will behave independently in each of the pipelines they match.

Triggers are loaded from their connection name. The driver type of the connection will dictate which options are available. See Drivers.

**pipeline.require**

If this section is present, it establishes prerequisites for any kind of item entering the Pipeline. Regardless of how the item is to be enqueued (via any trigger or automatic dependency resolution), the conditions specified here must be met or the item will not be enqueued. These requirements may vary depending on the source of the item being enqueued.

Requirements are loaded from their connection name. The driver type of the connection will dictate which options are available. See Drivers.

**pipeline.reject**

If this section is present, it establishes prerequisites that can block an item from being enqueued. It can be considered a negative version of pipeline.require.

Requirements are loaded from their connection name. The driver type of the connection will dictate which options are available. See Drivers.

**pipeline.precedence   
Default:normal**

Indicates how the build scheduler should prioritize jobs for different pipelines. Each pipeline may have one precedence, jobs for pipelines with a higher precedence will be run before ones with lower. The value should be one of high, normal, or low. Default: normal.

The following options configure reporters. Reporters are complementary to triggers; where a trigger is an event on a connection which causes Zuul to enqueue an item, a reporter is the action performed on a connection when an item is dequeued after its jobs complete. The actual syntax for a reporter is defined by the driver which implements it. See Drivers for more information.

**pipeline.success**

Describes where Zuul should report to if all the jobs complete successfully. This section is optional; if it is omitted, Zuul will run jobs and do nothing on success – it will not report at all. If the section is present, the listed reporters will be asked to report on the jobs. The reporters are listed by their connection name. The options available depend on the driver for the supplied connection.

**pipeline.failure**

These reporters describe what Zuul should do if at least one job fails.

**pipeline.merge-conflict**

These reporters describe what Zuul should do if it is unable to merge the patchset into the current state of the target branch. If no merge-conflict reporters are listed then the failure reporters will be used.

**pipeline.config-error**

These reporters describe what Zuul should do if it encounters a configuration error while trying to enqueue the item. If no config-error reporters are listed then the failure reporters will be used.

**pipeline.enqueue**

These reporters describe what Zuul should do when an item is enqueued into the pipeline. This may be used to indicate to a system or user that Zuul is aware of the triggering event even though it has not evaluated whether any jobs will run.

**pipeline.start**

These reporters describe what Zuul should do when jobs start running for an item in the pipeline. This can be used, for example, to reset a previously reported result.