**Queue**

Projects that interact with each other should share a queue. This is especially used in a dependent pipeline. The project.queue can optionally refer to a specific queue object that can further configure the behavior of the queue.

Here is an example queue configuration.

**- queue:**

**name: integrated**

**per-branch: false**

**queue**

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

**queue.name*(required)***

This is used later in the project definition to refer to this queue.

**queue.per-branch  
Default:false**

Queues by default define a single queue for all projects and branches that use it. This is especially important if projects want to do upgrade tests between different branches in the gate. If a set of projects doesn’t have this use case it can configure the queue to create a shared queue per branch for all projects. This can be useful for large projects to improve the throughput of a gate pipeline as this results in shorter queues and thus less impact when a job fails in the gate. Note that this means that all projects that should be gated must have aligned branch names when using per branch queues. Otherwise changes that belong together end up in different queues.

**queue.allow-circular-dependencies  
Default:false**

Determines whether Zuul is allowed to process circular dependencies between changes for this queue. All projects that are part of a dependency cycle must share the same change queue.

If Zuul detects a dependency cycle it will ensure that every change also includes all other changes that are part of the cycle. However each change will still be a normal item in the queue with its own jobs.

Reporting of success will be postponed until all items in the cycle succeed. In the case of a failure in any of those items the whole cycle will be dequeued.

An error message will be posted to all items of the cycle if some items fail to report (e.g. merge failure when some items were already merged). In this case the target branch(es) might be in a broken state.

In general, circular dependencies are considered to be an antipattern since they add extra constraints to continuous deployment systems. Additionally, due to the lack of atomicity in merge operations in code review systems (this includes Gerrit, even with submitWholeTopic set), it may be possible for only part of a cycle to be merged. In that case, manual interventions (such as reverting a commit, or bypassing gating to force-merge the remaining commits) may be required.

**queue.dependencies-by-topic  
Default:false**

Determines whether Zuul should query the code review system for changes under the same topic and treat those as a set of circular dependencies.

Note that the Gerrit code review system supports a setting called change.submitWholeTopic, which, when set, will cause all changes under the same topic to be merged simultaneously. Zuul automatically observes this setting and treats all changes to be submitted together as circular dependencies. If this setting is enabled in gerrit, do not enable dependencies-by-topic in associated Zuul queues.

Because change.submitWholeTopic is applied system-wide in Gerrit, some Zuul users may wish to emulate the behavior for some projects without enabling it for all of Gerrit. In this case, setting dependencies-by-topic will cause Zuul to approxiamate the Gerrit behavior only for changes enqueued into queues where this is set.

This setting requires queue.allow-circular-dependencies to also be set. All of the caveats noted there continue to apply.