**Project**

A project corresponds to a source code repository with which Zuul is configured to interact. The main responsibility of the project configuration item is to specify which jobs should run in which pipelines for a given project. Within each project definition, a section for each pipeline may appear. This project-pipeline definition is what determines how a project participates in a pipeline.

Multiple project definitions may appear for the same project (for example, in a central config projects as well as in a repo’s own .zuul.yaml). In this case, all of the project definitions for the relevant branch are combined (the jobs listed in all of the matching definitions will be run). If a project definition appears in a config-project, it will apply to all branches of the project. If it appears in a branch of an untrusted-project it will only apply to changes on that branch. In the case of an item which does not have a branch (for example, a tag), all of the project definitions will be combined.

Consider the following project definition:

**- project:**

**name: yoyodyne**

**queue: integrated**

**check:**

**jobs:**

**- check-syntax**

**- unit-tests**

**gate:**

**jobs:**

**- unit-tests**

**- integration-tests**

The project has two project-pipeline stanzas, one for the check pipeline, and one for gate. Each specifies which jobs should run when a change for that project enters the respective pipeline – when a change enters check, the check-syntax and unit-test jobs are run.

Pipelines which use the dependent pipeline manager (e.g., the gate example shown earlier) maintain separate queues for groups of projects. When Zuul serializes a set of changes which represent future potential project states, it must know about all of the projects within Zuul which may have an effect on the outcome of the jobs it runs. If project *A* uses project *B* as a library, then Zuul must be told about that relationship so that it knows to serialize changes to A and B together, so that it does not merge a change to B while it is testing a change to A.

Zuul could simply assume that all projects are related, or even infer relationships by which projects a job indicates it uses, however, in a large system that would become unwieldy very quickly, and unnecessarily delay changes to unrelated projects. To allow for flexibility in the construction of groups of related projects, the change queues used by dependent pipeline managers are specified manually. To group two or more related projects into a shared queue for a dependent pipeline, set the queue parameter to the same value for those projects.

The gate project-pipeline definition above specifies that this project participates in the integrated shared queue for that pipeline.

**project**

The following attributes may appear in a project:

**project.name**

The name of the project. If Zuul is configured with two or more unique projects with the same name, the canonical hostname for the project should be included (e.g., *git.example.com/foo*). This can also be a regex. In this case the regex must start with ^ and match the full project name following the same rule as name without regex. If not given it is implicitly derived from the project where this is defined.

**project.templates**

A list of Project Template references; the project-pipeline definitions of each Project Template will be applied to this project. If more than one template includes jobs for a given pipeline, they will be combined, as will any jobs specified in project-pipeline definitions on the project itself.

**project.default-branch  
Default:master**

The name of a branch that Zuul should check out in jobs if no better match is found. Typically Zuul will check out the branch which matches the change under test, or if a job has specified an job.override-checkout, it will check that out. However, if there is no matching or override branch, then Zuul will checkout the default branch.

Each project may only have one default-branch therefore Zuul will use the first value that it encounters for a given project (regardless of in which branch the definition appears). It may not appear in a Project Template definition.

This setting also affects the order in which configuration objects are processed. Zuul will process the default branch first before any other branches.

The Gerrit and GitHub drivers will automatically use the default branch as specified for the repository in their respective systems as a default value for this setting. It may be overridden by setting this value explicitly.

**project.merge-mode  
Default:(driver specific)**

The merge mode which is used by Git for this project. Be sure this matches what the remote system which performs merges (i.e., Gerrit). The requested merge mode will also be used by the GitHub and GitLab drivers when performing merges.

The default is merge for all drivers except Gerrit, where the default is merge-resolve.

Each project may only have one merge-mode therefore Zuul will use the first value that it encounters for a given project (regardless of in which branch the definition appears). It may not appear in a Project Template definition.

It must be one of the following values:

**merge**

Uses the default git merge strategy (recursive). This maps to the merge mode merge in GitHub and GitLab.

**merge-resolve**

Uses the resolve git merge strategy. This is a very conservative merge strategy which most closely matches the behavior of Gerrit. This maps to the merge mode merge in GitHub and GitLab.

**rebase**

Rebases the changes onto the branch. This is only supported by GitHub and maps to the rebase merge mode (but does not alter committer information in the way that GitHub does in the repos that Zuul prepares for jobs).

**project.vars  
Default:None**

A dictionary of variables to be made available for all jobs in all pipelines of this project. For more information see variable inheritance.

**project.queue**

This specifies the name of the shared queue this project is in. Any projects which interact with each other in tests should be part of the same shared queue in order to ensure that they don’t merge changes which break the others. This is a free-form string; just set the same value for each group of projects.

**Project Template**

A Project Template defines one or more project-pipeline definitions which can be re-used by multiple projects.

A Project Template uses the same syntax as a Project definition, however, in the case of a template, the project.name attribute does not refer to the name of a project, but rather names the template so that it can be referenced in a Project definition.

Because Project Templates may be used outside of the projects where they are defined, they honor the implied branch pragmas (unlike Projects). The same heuristics described in job.branches that determine what implied branches a Job will receive apply to Project Templates (with the exception that it is not possible to explicity set a branch matcher on a Project Template).