**Job**

A job is a unit of work performed by Zuul on an item enqueued into a pipeline. Items may run any number of jobs (which may depend on each other). Each job is an invocation of an Ansible playbook with a specific inventory of hosts. The actual tasks that are run by the job appear in the playbook for that job while the attributes that appear in the Zuul configuration specify information about when, where, and how the job should be run.

In addition to the job’s main playbook, each job may specify one or more pre- and post-playbooks. These are run, in order, before and after (respectively) the main playbook. They may be used to set up and tear down resources needed by the main playbook. When combined with inheritance, they provide powerful tools for job construction. A job only has a single main playbook, and when inheriting from a parent, the child’s main playbook overrides (or replaces) the parent’s. However, the pre- and post-playbooks are appended and prepended in a nesting fashion. So if a parent job and child job both specified pre and post playbooks, the sequence of playbooks run would be:

* parent pre-run playbook
* child pre-run playbook
* child playbook
* child post-run playbook
* parent post-run playbook
* parent cleanup-run playbook

Further inheritance would nest even deeper.

Here is an example of two job definitions:

**- job:**

**name: base**

**pre-run: copy-git-repos**

**post-run: copy-logs**

**- job:**

**name: run-tests**

**parent: base**

**nodeset:**

**nodes:**

**- name: test-node**

**label: fedora**

The following attributes are available on a job; all are optional unless otherwise specified:

**job.name*(required)***

The name of the job. By default, Zuul looks for a playbook with this name to use as the main playbook for the job. This name is also referenced later in a project pipeline configuration.

**job.parent  
Default:Tenant default-parent**

Specifies a job to inherit from. The parent job can be defined in this or any other project. Any attributes not specified on a job will be collected from its parent. If no value is supplied here, the job specified by tenant.default-parent will be used. If **parent** is set to null (which is only valid in a config-project), this is a base job.

**job.description**

A textual description of the job. Not currently used directly by Zuul, but it is used by the zuul-sphinx extension to Sphinx to auto-document Zuul jobs (in which case it is interpreted as ReStructuredText.

**job.final  
Default:false**

To prevent other jobs from inheriting from this job, and also to prevent changing execution-related attributes when this job is specified in a project’s pipeline, set this attribute to true.

**job.success-message  
Default:SUCCESS**

Normally when a job succeeds, the string SUCCESS is reported as the result for the job. If set, this option may be used to supply a different string.

**job.failure-message  
Default:FAILURE**

Normally when a job fails, the string FAILURE is reported as the result for the job. If set, this option may be used to supply a different string.

**job.hold-following-changes  
Default:false**

In a dependent pipeline, this option may be used to indicate that no jobs should start on any items which depend on the current item until this job has completed successfully. This may be used to conserve build resources, at the expense of inhibiting the parallelization which speeds the processing of items in a dependent pipeline.

**job.voting  
Default:true**

Indicates whether the result of this job should be used in determining the overall result of the item.

**job.provides**

A list of free-form strings which identifies resources provided by this job which may be used by other jobs for other changes using the job.requires attribute.

When inheriting jobs or applying variants, the list of *provides* is extended (*provides* specified in a job definition are added to any supplied by their parents).

**job.requires**

A list of free-form strings which identify resources which may be provided by other jobs for other changes (via the job.provides attribute) that are used by this job.

When Zuul encounters a job with a *requires* attribute, it searches for those values in the *provides* attributes of any jobs associated with any queue items ahead of the current change. In this way, if a change uses either git dependencies or a *Depends-On* header to indicate a dependency on another change, Zuul will be able to determine that the parent change affects the run-time environment of the child change. If such a relationship is found, the job with *requires* will not start until all of the jobs with matching *provides* have completed or paused. Additionally, the artifacts returned by the *provides* jobs will be made available to the *requires* job.

When inheriting jobs or applying variants, the list of *requires* is extended (*requires* specified in a job definition are added to any supplied by their parents).

For example, a job which produces a builder container image in one project that is then consumed by a container image build job in another project might look like this:

**- job:**

**name: build-builder-image**

**provides: images**

**- job:**

**name: build-final-image**

**requires: images**

**- project:**

**name: builder-project**

**check:**

**jobs:**

**- build-builder-image**

**- project:**

**name: final-project**

**check:**

**jobs:**

**- build-final-image**

**job.secrets**

A list of secrets which may be used by the job. A Secret is a named collection of private information defined separately in the configuration. The secrets that appear here must be defined in the same project as this job definition.

Each item in the list may may be supplied either as a string, in which case it references the name of a Secret definition, or as a dict. If an element in this list is given as a dict, it may have the following fields:

**job.secrets.name*(required)***

The name to use for the Ansible variable into which the secret content will be placed.

**job.secrets.secret*(required)***

The name to use to find the secret’s definition in the configuration.

**job.secrets.pass-to-parent  
Default:false**

A boolean indicating that this secret should be made available to playbooks in parent jobs. Use caution when setting this value – parent jobs may be in different projects with different security standards. Setting this to true makes the secret available to those playbooks and therefore subject to intentional or accidental exposure.

For example:

**- secret:**

**name: important-secret**

**data:**

**key: encrypted-secret-key-data**

**- job:**

**name: amazing-job**

**secrets:**

**- name: ssh\_key**

**secret: important-secret**

**job.nodeset**

The nodes which should be supplied to the job. This parameter may be supplied either as a string, in which case it references a Nodeset definition which appears elsewhere in the configuration, or a dictionary, in which case it is interpreted in the same way as a Nodeset definition, though the top-level nodeset name attribute should be omitted (in essence, it is an anonymous Nodeset definition unique to this job; the nodes themselves still require names). See the Nodeset reference for the syntax to use in that case.

If a job has an empty (or no) Nodeset definition, it will still run and is able to perform limited actions within the Zuul executor sandbox. Note so-called “executor-only” jobs run with an empty inventory, and hence Ansible’s *implicit localhost*. This means an executor-only playbook must be written to match localhost directly; i.e.

**- hosts: localhost**

**tasks:**

**...**

not with hosts: all (as this does not match the implicit localhost and the playbook will not run). There are also caveats around things like enumerating the magic variable hostvars in this situation. For more information see the Ansible implicit localhost documentation.

A useful example of executor-only jobs is saving resources by directly utilising the prior results from testing a committed change. For example, a review which updates documentation source files would generally test validity by building a documentation tree. When this change is committed, the pre-built output can be copied in an executor-only job directly to the publishing location in a post-commit *promote* pipeline; avoiding having to use a node to rebuild the documentation for final publishing.

**job.pre-run**

The name of a playbook or list of playbooks to run before the main body of a job. Values are either a string describing the full path to the playbook in the repo where the job is defined, or a dictionary described below.

When a job inherits from a parent, the child’s pre-run playbooks are run after the parent’s. See Job for more information.

If the value is a dictionary, the following attributes are available:

**job.pre-run.name**

The path to the playbook relative to the root of the repo.

**job.pre-run.semaphoreℑ**

The name of a Semaphore (or list of them) or Global Semaphore which should be acquired and released when the playbook begins and ends. If the semaphore is at maximum capacity, then Zuul will wait until it can be acquired before starting the playbook. The format is either a string, or a list of strings.

If multiple semaphores are requested, the playbook will not start until all have been acquired, and Zuul will wait until all are available before acquiring any. The time spent waiting for pre-run playbook semaphores is counted against the job.timeout.

None of the semaphores specified for a playbook may also be specified in the same job.

**job.post-run**

The name of a playbook or list of playbooks to run after the main body of a job. Values are either a string describing the full path to the playbook in the repo where the job is defined, or a dictionary described below.

When a job inherits from a parent, the child’s post-run playbooks are run before the parent’s. See Job for more information.

The name of a playbook or list of playbooks to run after job execution. Values are either a string describing the full path to the playbook in the repo where the job is defined, or a dictionary described below.

The cleanup phase is performed regardless of the job’s result, even when the job is canceled. Cleanup results are not taken into account when reporting the job result.

When a job inherits from a parent, the child’s cleanup-run playbooks are run before the parent’s. See Job for more information.

There is a hard-coded five minute timeout for cleanup playbooks.

If the value is a dictionary, the following attributes are available:

**job.run**

The name of a playbook or list of playbooks for this job. If it is not supplied, the parent’s playbook will be used (and likewise up the inheritance chain). Values are either a string describing the full path to the playbook in the repo where the job is defined, or a dictionary described below.

If the value is a dictionary, the following attributes are available:

**job.run.name**

The path to the playbook relative to the root of the repo.

**job.dependencies**

A list of other jobs upon which this job depends. Zuul will not start executing this job until all of its dependencies have completed successfully or have been paused, and if one or more of them fail, this job will not be run.

The format for this attribute is either a list of strings or dictionaries. Strings are interpreted as job names, dictionaries, if used, may have the following attributes:

**job.dependencies.name*(required)***

The name of the required job.

**job.dependencies.soft  
Default:false**

A boolean value which indicates whether this job is a *hard* or *soft* dependency. A *hard* dependency will cause an error if the specified job is not run. That is, if job B depends on job A, but job A is not run for any reason (for example, it contains a file matcher which does not match), then Zuul will not run any jobs and report an error. A *soft* dependency will simply be ignored if the dependent job is not run.