# Tenant Configuration

After zuul.conf is configured, Zuul component servers will be able to start, but a tenant configuration is required in order for Zuul to perform any actions. The tenant configuration file specifies upon which projects Zuul should operate. These repositories are grouped into tenants. The configuration of each tenant is separate from the rest (no pipelines, jobs, etc are shared between them).

A project may appear in more than one tenant; this may be useful if you wish to use common job definitions across multiple tenants.

Actions normally available to the Zuul operator only can be performed by specific users on Zuul’s REST API if admin rules are listed for the tenant. Authorization rules are also defined in the tenant configuration file.

The tenant configuration file is specified by the scheduler.tenant\_config setting in zuul.conf. It is a YAML file which, like other Zuul configuration files, is a list of configuration objects, though only a few types of objects (described below) are supported.

Alternatively the scheduler.tenant\_config\_script can be the path to an executable that will be executed and its stdout used as the tenant configuration. The executable must return a valid tenant YAML formatted output.

Tenant configuration is checked for updates any time a scheduler is started, and changes to it are read automatically. If the tenant configuration is altered during operation, you can signal a scheduler to read and apply the updated state in order to avoid restarting. See the section on Reconfiguration for instructions. Ideally, tenant configuration deployment via configuration management should also be made to trigger a smart-reconfigure once the file is replaced.

## **Tenant**

A tenant is a collection of projects which share a Zuul configuration. Some examples of tenant definitions are:

- **tenant**:

**name**: my-tenant

**max-nodes-per-job**: 5

**exclude-unprotected-branches**: false

**source**:

**gerrit**:

**config-projects**:

- common-config

- **shared-jobs**:

**include**: job

**untrusted-projects**:

- **zuul/zuul-jobs**:

**shadow**: common-config

- project1

- **project2**:

**exclude-unprotected-branches**: true

- **tenant**:

**name**: my-tenant

**admin-rules**:

- acl1

- acl2

**source**:

**gerrit**:

**config-projects**:

- common-config

**untrusted-projects**:

- **exclude**:

- job

- semaphore

- project

- project-template

- nodeset

- secret

**projects**:

- project1

- **project2**:

**exclude-unprotected-branches**: true

**tenant**

The following attributes are supported:

**tenant.name*(required)***

The name of the tenant. This may appear in URLs, paths, and monitoring fields, and so should be restricted to URL friendly characters (ASCII letters, numbers, hyphen and underscore) and you should avoid changing it unless necessary.

**tenant.source*(required)***

A dictionary of sources to consult for projects. A tenant may contain projects from multiple sources; each of those sources must be listed here, along with the projects it supports. The name of a connection is used as the dictionary key (e.g. gerrit in the example above), and the value is a further dictionary containing the keys below.

The next two attributes, **config-projects** and **untrusted-projects** provide the bulk of the information for tenant configuration. They list all of the projects upon which Zuul will act.

The order of the projects listed in a tenant is important. A job which is defined in one project may not be redefined in another project; therefore, once a job appears in one project, a project listed later will be unable to define a job with that name. Further, some aspects of project configuration (such as the merge mode) may only be set on the first appearance of a project definition.

Zuul loads the configuration from all **config-projects** in the order listed, followed by all **untrusted-projects** in order.

**tenant.config-projects**

A list of projects to be treated as config projects in this tenant. The jobs in a config project are trusted, which means they run with extra privileges, do not have their configuration dynamically loaded for proposed changes, and Zuul config files are only searched for in the master branch.

The items in the list follow the same format described in **untrusted-projects**.

**tenant.config-projects.<project>**

The config-projects have an additional config option that may be specified optionally.

**tenant.config-projects.<project>.load-branch   
Default:master**

Define which branch is loaded from a config project. By default config projects load Zuul configuration only from the master branch.

**tenant.untrusted-projects**

A list of projects to be treated as untrusted in this tenant. An untrusted-project is the typical project operated on by Zuul. Their jobs run in a more restrictive environment, they may not define pipelines, their configuration dynamically changes in response to proposed changes, and Zuul will read configuration files in all of their branches.

**tenant.untrusted-projects.<project>**

The items in the list may either be simple string values of the project names, or a dictionary with the project name as key and the following values:

**tenant.untrusted-projects.<project>.include**

Normally Zuul will load all of the Configuration Items appropriate for the type of project (config or untrusted) in question. However, if you only want to load some items, the **include** attribute can be used to specify that only the specified items should be loaded. Supplied as a string, or a list of strings.

The following **configuration items** are recognized:

* pipeline
* job
* semaphore
* project
* project-template
* nodeset
* secret

**tenant.untrusted-projects.<project>.exclude**

A list of **configuration items** that should not be loaded.

**tenant.untrusted-projects.<project>.shadow**

Normally, only one project in Zuul may contain definitions for a given job. If a project earlier in the configuration defines a job which a later project redefines, the later definition is considered an error and is not permitted. The **shadow** attribute of a project indicates that job definitions in this project which conflict with the named projects should be ignored, and those in the named project should be used instead. The named projects must still appear earlier in the configuration. In the example above, if a job definition appears in both the common-config and zuul-jobs projects, the definition in common-config will be used.

**tenant.untrusted-projects.<project>.exclude-unprotected-branches**

Define if unprotected branches should be processed. Defaults to the tenant wide setting of exclude-unprotected-branches. This currently only affects GitHub and GitLab projects.

**tenant.untrusted-projects.<project>.include-branches**

A list of regexes matching branches which should be processed. If omitted, all branches are included. Operates after exclude-unprotected-branches and so may be used to further reduce the set of branches (but not increase it).

It has priority over exclude-branches.

**tenant.untrusted-projects.<project>.exclude-branches**

A list of regexes matching branches which should be processed. If omitted, all branches are included. Operates after exclude-unprotected-branches and so may be used to further reduce the set of branches (but not increase it).

It will not exclude a branch which already matched include-branches.

**tenant.untrusted-projects.<project>.always-dynamic-branches**

A list of regular expressions matching branches which should be treated as if every change newly proposes dynamic Zuul configuration. In other words, the only time Zuul will realize any configuration related to these branches is during the time it is running jobs for a proposed change.

This is potentially useful for situations with large numbers of rarely used feature branches, but comes at the cost of a significant reduction in Zuul features for these branches.

Every regular expression listed here will also implicitly be included in exclude-branches, therefore Zuul will not load any static in-repo configuration from this branch. These branches will not be available for use in overriding checkouts of repos, nor will they be included in the git repos that Zuul prepares for required-projects (unless there is a change in the dependency tree for this branch).

In particular, this means that the only jobs which can be specified for these branches are pre-merge and gating jobs (such as check and gate). No post-merge or periodic jobs will run for these branches.

Using this setting also incurs additional processing for each change submitted for these branches as Zuul must recalculate the configuration layout it uses for such a change as if it included a change to a zuul.yaml file, even if the change does not alter the configuration).

With all these caveats in mind, this can be useful for repos with large numbers of rarely used branches as it allows Zuul to omit their configuration in most circumstances and only calculate the configuration of a single additional branch when it is used.

**tenant.untrusted-projects.<project>.implied-branch-matchers**

This is a boolean, which, if set, may be used to enable (true) or disable (false) the addition of implied branch matchers to job and project-template definitions. Normally Zuul decides whether to add these based on heuristics described in job.branches. This attribute overrides that behavior.

This can be useful if branch settings for this project may produce an unpredictable number of branches to load from. Setting this value explicitly here can avoid unexpected behavior changes as branches are added or removed from the load set.

The pragma.implied-branch-matchers pragma will override the setting here if present.

Note that if a job contains an explicit branch matcher, it will be used regardless of the value supplied here.

**tenant.untrusted-projects.<project>.extra-config-paths**

Normally Zuul loads in-repo configuration from the first of these paths:

* zuul.yaml
* zuul.d/\*
* .zuul.yaml
* .zuul.d/\*

If this option is supplied then, after the normal process completes, Zuul will also load any configuration found in the files or paths supplied here. This can be a string or a list. If a list of multiple items, Zuul will load configuration from all of the items in the list (it will not stop at the first extra configuration found). Directories should be listed with a trailing /. Example:

**extra-config-paths**:

- zuul-extra.yaml

- zuul-extra.d/

This feature may be useful to allow a project that primarily holds shared jobs or roles to include additional in-repo configuration for its own testing (which may not be relevant to other users of the project).

**tenant.untrusted-projects.<project-group>**

The items in the list are dictionaries with the following attributes. A **configuration items** definition is applied to the list of projects.

**tenant.untrusted-projects.<project-group>.include**

A list of **configuration items** that should be loaded.

**tenant.untrusted-projects.<project-group>.exclude**

A list of **configuration items** that should not be loaded.

**tenant.untrusted-projects.<project-group>.projects**

A list of **project** items.

**tenant.max-dependencies**

This setting can be used to limit the number of dependencies that Zuul will consider when enqueing a change in any pipeline in this tenant. If used, it should be set to a value that is higher than the highest number of dependencies that are expected to be encountered. If, when enqueing a change, Zuul detects that the dependencies will exceed this value, Zuul will not enqueue the change and will provide no feedback to the user. This is meant only to protect the Zuul server from resource exhaustion when excessive dependencies are present. The default (unset) is no limit. Note that the value 0 does not disable this option; instead it limits Zuul to zero dependencies. This is distinct from <gerrit connection>.max\_dependencies.

**tenant.max-nodes-per-job   
Default:5**

The maximum number of nodes a job can request. A value of ‘-1’ value removes the limit.

**tenant.max-job-timeout   
Default:10800**

The maximum timeout for jobs. A value of ‘-1’ value removes the limit.

**tenant.exclude-unprotected-branches   
Default:false**

When using a branch and pull model on a shared repository there are usually one or more protected branches which are gated and a dynamic number of personal/feature branches which are the source for the pull requests. These branches can potentially include broken Zuul config and therefore break the global tenant wide configuration. In order to deal with this Zuul’s operations can be limited to the protected branches which are gated. This is a tenant wide setting and can be overridden per project. This currently only affects GitHub and GitLab projects.

**tenant.default-parent   
Default:base**

If a job is defined without an explicit job.parent attribute, this job will be configured as the job’s parent. This allows an administrator to configure a default base job to implement local policies such as node setup and artifact publishing.

**tenant.default-ansible-version**

Default ansible version to use for jobs that doesn’t specify a version. See job.ansible-version for details.

**tenant.allowed-triggers   
Default:all connections**

The list of connections a tenant can trigger from. When set, this setting can be used to restrict what connections a tenant can use as trigger. Without this setting, the tenant can use any connection as a trigger.

**tenant.allowed-reporters   
Default:all connections**

The list of connections a tenant can report to. When set, this setting can be used to restrict what connections a tenant can use as reporter. Without this setting, the tenant can report to any connection.

**tenant.allowed-labels   
Default:[]**

The list of labels (as strings or regular expressions) a tenant can use in a job’s nodeset. When set, this setting can be used to restrict what labels a tenant can use. Without this setting, the tenant can use any labels.

**tenant.disallowed-labels   
Default:[]**

The list of labels (as strings or regular expressions) a tenant is forbidden to use in a job’s nodeset. When set, this setting can be used to restrict what labels a tenant can use. Without this setting, the tenant can use any labels permitted by tenant.allowed-labels. This check is applied after the check for allowed-labels and may therefore be used to further restrict the set of permitted labels.

**tenant.web-root**

If this tenant has a whitelabeled installation of zuul-web, set its externally visible URL here (e.g., https://tenant.example.com/). This will override the web.root setting when constructing URLs for this tenant.

**tenant.admin-rules**

A list of authorization rules to be checked in order to grant administrative access to the tenant through Zuul’s REST API and web interface.

At least one rule in the list must match for the user to be allowed to execute privileged actions. A matching rule will also allow the user access to the tenant in general (i.e., the rule does not need to be duplicated in access-rules).

More information on tenant-scoped actions can be found in Authenticated Access.

**tenant.access-rules**

A list of authorization rules to be checked in order to grant read access to the tenant through Zuul’s REST API and web interface.

If no rules are listed, then anonymous access to the tenant is permitted. If any rules are present then at least one rule in the list must match for the user to be allowed to access the tenant.

More information on tenant-scoped actions can be found in Authenticated Access.

**tenant.authentication-realm**

Each authenticator defined in Zuul’s configuration is associated to a realm. When authenticating through Zuul’s Web User Interface under this tenant, the Web UI will redirect the user to this realm’s authentication service. The authenticator must be of the type OpenIDConnect.

**Note**

Defining a default realm for a tenant will not invalidate access tokens issued from other configured realms. This is intended so that an operator can issue an overriding access token manually. If this is an issue, it is advised to add finer filtering to admin rules, for example, filtering by the iss claim (generally equal to the issuer ID).

**tenant.semaphores**

A list of names of global-semaphore objects to allow jobs in this tenant to access.

## **Global Semaphore**

Semaphores are normally defined in in-repo configuration (see Semaphore), however to support use-cases where semaphores are used to represent constrained global resources that may be used by multiple Zuul tenants, semaphores may be defined within the main tenant configuration file.

In order for a job to use a global semaphore, the semaphore must first be defined in the tenant configuration file with global-semaphore and then added to each tenant which should have access to it with tenant.semaphores. Once that is done, Zuul jobs may use that semaphore in the same way they would use a normal tenant-scoped semaphore.

If any tenant which is granted access to a global semaphore also has a tenant-scoped semaphore defined with the same name, that definition will be treated as a configuration error and subsequently ignored in favor of the global semaphore.

An example definition looks similar to the normal semaphore object:

- **global-semaphore**:

**name**: global-semaphore-foo

**max**: 5

**global-semaphore**

The following attributes are available:

**global-semaphore.name*(required)***

The name of the semaphore, referenced by jobs.

**global-semaphore.max   
Default:1**

The maximum number of running jobs which can use this semaphore.

## **Authorization Rule**

An authorization rule is a set of conditions the claims of a user’s JWT must match in order to be allowed to perform actions at a tenant’s level.

When an authorization rule is included in the tenant’s admin-rules, the protected actions available are **autohold**, **enqueue**, **dequeue** and **promote**.

Below are some examples of how authorization rules can be defined:

- **authorization-rule**:

**name**: affiliate\_or\_admin

**conditions**:

- **resources\_access**:

**account**:

**roles**: "affiliate"

**iss**: external\_institution

- **resources\_access.account.roles**: "admin"

- **authorization-rule**:

**name**: alice\_or\_bob

**conditions**:

- **zuul\_uid**: alice

- **zuul\_uid**: bob

Zuul previously used admin-rule for these definitions. That form is still permitted for backwards compatibility, but is deprecated and will be removed in a future version of Zuul.

**authorization-rule**

The following attributes are supported:

**authorization-rule.name*(required)***

The name of the rule, so that it can be referenced in the admin-rules attribute of a tenant’s definition. It must be unique.

**authorization-rule.conditions*(required)***

This is the list of conditions that define a rule. A JWT must match **at least one** of the conditions for the rule to apply. A condition is a dictionary where keys are claims. **All** the associated values must match the claims in the user’s token; in other words the condition dictionary must be a “sub-dictionary” of the user’s JWT.

Zuul’s authorization engine will adapt matching tests depending on the nature of the claim in the token, eg:

* if the claim is a JSON list, check that the condition value is in the claim
* if the claim is a string, check that the condition value is equal to the claim’s value

The claim names can also be written in the XPath format for clarity: the condition

**resources\_access**:

**account**:

**roles**: "affiliate"

is equivalent to the condition

**resources\_access.account.roles**: "affiliate"

The special zuul\_uid claim refers to the uid\_claim setting in an authenticator’s configuration. By default it refers to the sub claim of a token. For more details see the Authenticated Access.

Under the above example, the following token would match rules affiliate\_or\_admin and alice\_or\_bob:

{

'iss': 'external\_institution',

'aud': 'my\_zuul\_deployment',

'exp': 1234567890,

'iat': 1234556780,

'sub': 'alice',

'resources\_access': {

'account': {

'roles': ['affiliate', 'other\_role']

}

},

}

And this token would only match rule affiliate\_or\_admin:

{

'iss': 'some\_other\_institution',

'aud': 'my\_zuul\_deployment',

'exp': 1234567890,

'sub': 'carol',

'iat': 1234556780,

'resources\_access': {

'account': {

'roles': ['admin', 'other\_role']

}

},

}

## **Authorization Rule Templating**

The special word “{tenant.name}” can be used in conditions’ values. It will be automatically substituted for the relevant tenant when evaluating authorizations for a given set of claims. For example, consider the following rule:

- **authorization-rule**:

**name**: tenant\_in\_groups

**conditions**:

- **groups**: "{tenant.name}"

If applied to the following tenants:

- **tenant**:

**name**: tenant-one

**admin-rules**:

- tenant\_in\_groups

- **tenant**:

**name**: tenant-two

**admin-rules**:

- tenant\_in\_groups

Then this set of claims will be allowed to perform protected actions on **tenant-one**:

{

'iss': 'some\_other\_institution',

'aud': 'my\_zuul\_deployment',

'exp': 1234567890,

'sub': 'carol',

'iat': 1234556780,

'groups': ['tenant-one', 'some-other-group'],

}

And this set of claims will be allowed to perform protected actions on **tenant-one** and **tenant-two**:

{

'iss': 'some\_other\_institution',

'aud': 'my\_zuul\_deployment',

'exp': 1234567890,

'sub': 'carol',

'iat': 1234556780,

'groups': ['tenant-one', 'tenant-two'],

}

## **API Root**

Most actions in zuul-web, zuul-client, and the REST API are understood to be within the context of a specific tenant and therefore the authorization rules specified by that tenant apply. When zuul-web is deployed in a multi-tenant scenario (the default), there are a few extra actions or API methods which are outside of the context of an individual tenant (for example, listing the tenants or observing the state of Zuul system components). To control access to these methods, an api-root object can be used.

At most one api-root object may appear in the tenant configuration file. If more than one appears, it is an error. If there is no api-root object, then anonymous read-only access to the tenant list and other root-level API methods is assumed.

The /api/info endpoint is never protected by Zuul since it supplies the authentication information needed by the web UI.

API root access is not a pre-requisite to access tenant-specific URLs.

**api-root**

The following attributes are supported:

**api-root.authentication-realm**

Each authenticator defined in Zuul’s configuration is associated to a realm. When authenticating through Zuul’s Web User Interface at the multi-tenant root, the Web UI will redirect the user to this realm’s authentication service. The authenticator must be of the type OpenIDConnect.

**Note**

Defining a default realm for the root API will not invalidate access tokens issued from other configured realms. This is intended so that an operator can issue an overriding access token manually. If this is an issue, it is advised to add finer filtering to admin rules, for example, filtering by the iss claim (generally equal to the issuer ID).

**api-root.access-rules**

A list of authorization rules to be checked in order to grant read access to the top-level (i.e., non-tenant-specific) portion of Zuul’s REST API and web interface.

If no rules are listed, then anonymous access to top-level methods is permitted. If any rules are present then at at least one rule in the list must match for the user to be allowed access.

More information on tenant-scoped actions can be found in Authenticated Access.

# 