Triage Process and GitHub Labels for Angular

This document describes how the Angular team uses labels and milestones to triage issues on GitHub. The basic idea of the process is that caretaker only assigns a component (`comp: *`) label. The owner of the component is then responsible for the detailed / component-level triage.

Label Types

Components

The caretaker should be able to determine which component the issue belongs to. The components have a clear piece of source code associated with it within the `/packages/` folder of this repo.

- `comp: animations`
- `comp: bazel` - @angular/bazel rules
- `comp: benchpress`
- `comp: common` - this includes core components / pipes
- `comp: common/http` - this includes core components / pipes
- `comp: core & compiler` - because core, compiler, compiler-cli and browser-platforms are very intertwined, we will be treating them as one
- `comp: ivy` - a subset of core representing the new Ivy renderer.
- `comp: ngcc` - a subset of ivy representing the Angular Compatibility Compiler
- `comp: ve` - a subset of `core & compiler` representing changes specific to ViewEngine (legacy compiler/renderer).
- `comp: docs-infra` - the angular.io application and docs-related tooling
- `comp: elements`
- `comp: forms`
- `comp: http`
- `comp: i18n`
- `comp: language-service`
- `comp: metadata-extractor`
- `comp: router`
- `comp: server`
- `comp: service-worker`
- `comp: testing`
- `comp: upgrade`
- `comp: web-worker`
- `comp: zones`

There are few components which are cross-cutting. They don't have a clear location in the source tree. We will treat them as a component even thought no specific source tree is associated with them.

- `comp: build & ci` - build and CI infrastructure for the angular/angular repo
- `comp: dev-infra` - angular's common internal developer infrastructure scripting and tooling
- `comp: docs` - documentation, including API docs, guides, tutorial
- `comp: packaging` - packaging format of @angular/* npm packages
- `comp: performance`
- `comp: security`

Sometimes, especially in the case of cross-cutting issues or PRs, these PRs or issues belong to multiple components. In these cases, all applicable component labels should be used to triage the issue or PR.

Community engagement

- `help wanted` - Indicates an issue whose complexity/scope makes it suitable for a community contributor to pick up.
- `good first issue` - Indicates an issue that is suitable for first-time contributors. (This label should be applied *in addition* to `help wanted` for better discoverability.)

`help wanted` and `good first issue` are default GitHub labels familiar to many developers.

Caretaker Triage Process (Initial Triage)

The caretaker assigns `comp: *` labels to new issues as they come in. Untriaged issues can be found by selecting the issues with no milestone.

If an issue or PR obviously relates to a release regression, the caretaker must assign an appropriate priority (`P0` or `P1`) and ensure that someone from the team is actively working to resolve it.

Initial triage should occur daily so that issues can move into detailed triage.

Once the initial triage is done, the ng-bot automatically adds the milestone `needs triage` .

Detailed Triage

Detailed triage can be done by anyone familiar with the issue subject matter.

Step 1: Does the issue have enough information?

Gauge whether the issue has enough information to act upon. This typically includes a test case via StackBlitz or GitHub and steps to reproduce. If the issue may be legitimate but needs more information, add the "needs clarification" label. These labels can be revisited if the author can provide further clarification. If the issue does have enough information, move on to step 2.

Step 2: Bug, feature, or discussion?

By default, all issues are considered bugs. Bug reports require only a priority label.

If the issue is a feature request, apply the "feature" label. Use your judgement to determine whether the feature request is reasonable. If it's clear that the issue requests something infeasible, close the issue with a comment explaining why.

If the issue is an RFC or discussion, apply the "discussion" label. Use your judgement to determine whether this discussion belongs on GitHub. Discussions here should pertain to the technical implementation details of Angular. Redirect requests for debugging help or advice to a more appropriate channel unless they're capturing a legitimate bug.

Step 3: Set a Priority

For bug reports, set a priority label.

Label	Description
P0	An issue that causes a full outage, breakage, or major function unavailability for everyone, without

	any known workaround. The issue must be fixed immediately, taking precedence over all other work. Should receive updates at least once per day.
P1	An issue that significantly impacts a large percentage of users; if there is a workaround it is partial or overly painful. The issue should be resolved before the next release.
P2	The issue is important to a large percentage of users, with a workaround. Issues that are significantly ugly or painful (especially first-use or install-time issues). Issues with workarounds that would otherwise be P0 or P1.
P3	An issue that is relevant to core functions, but does not impede progress. Important, but not urgent.
P4	A relatively minor issue that is not relevant to core functions, or relates only to the attractiveness or pleasantness of use of the system. Good to have but not necessary changes/fixes.
P5	The team acknowledges the request but (due to any number of reasons) does not plan to work on or accept contributions for this request. The issue remains open for discussion.

Issues marked with "feature" or "discussion" don't require a priority.

Step 4: Apply additional information labels

Many optional labels provide additional context for issues. Consider adding any of the following if they apply to the issue:

- Browser or operating system labels (`windows` , `browser: ie 11` , etc.)
- Labels that inform the severity (`regression` , `has workaround` , `no workaround`)
- Labels that categorize the bug (`performance` , `refactoring` , `memory leak`)
- Community engagement labels (`help wanted` , `good first issue`)

Once this triage is done, the ng-bot automatically changes the milestone from `needs triage` to `Backlog` .

Triaging PRs

PRs labels signal their state. Every triaged PR must have a `action: *` label assigned to it:

- `action: discuss` : Discussion is needed, to be led by the author.
  - ***Who adds it:*** *Typically the PR author.*
  - ***Who removes it:*** *Whoever added it.*
- `action: review` (optional): One or more reviews are pending. The label is optional, since the review status can be derived from GitHub's Reviewers interface.
  - ***Who adds it:*** *Any team member. The caretaker can use it to differentiate PRs pending review from merge-ready PRs.*
  - ***Who removes it:*** *Whoever added it or the reviewer adding the last missing review.*
- `action: cleanup` : More work is needed from the author.
  - ***Who adds it:*** *The reviewer requesting changes to the PR.*
  - ***Who removes it:*** *Either the author (after implementing the requested changes) or the reviewer (after confirming the requested changes have been implemented).*
- `action: merge` : The PR author is ready for the changes to be merged by the caretaker as soon as the PR is green (or merge-assistance label is applied and caretaker has deemed it acceptable manually). In other words, this label indicates to "auto submit when ready".
  - ***Who adds it:*** *Typically the PR author.*

    - ***Who removes it:*** *Whoever added it.*

In addition, PRs can have the following states:

- `state: WIP` : PR is experimental or rapidly changing. Not ready for review or triage.
    - ***Who adds it:*** *The PR author.*
    - ***Who removes it:*** *Whoever added it.*
- `state: blocked` : PR is blocked on an issue or other PR. Not ready for merge.
    - ***Who adds it:*** *Any team member.*
    - ***Who removes it:*** *Any team member.*

When a PR is ready for review, a review should be requested using the Reviewers interface in GitHub.

PR Target

In our git workflow, we merge changes either to the `master` branch, the active patch branch (e.g. `5.0.x`), or to both.

The decision about the target must be done by the PR author and/or reviewer. This decision is then honored when the PR is being merged by the caretaker.

To communicate the target we use GitHub labels and only one target label may be applied to a PR.

Targeting an active release train:

- `target: major` : Any breaking change
- `target: minor` : Any new feature
- `target: patch` : Bug fixes, refactorings, documentation changes, etc. that pose no or very low risk of adversely affecting existing applications.

Special Cases:

- `target: rc` : A critical fix for an active release-train while it is in a feature freeze or RC phase
- `target: lts` : A critical fix for a specific release-train that is still within the long term support phase

Notes:

- To land a change only in a patch/RC branch, without landing it in any other active release-train branch (such as `master`), the patch/RC branch can be targeted in the GitHub UI with the appropriate `target: patch` / `target: rc` label.
- `target: lts` PRs must target the specific LTS branch they would need to merge into in the GitHub UI, in cases which a change is desired in multiple LTS branches, individual PRs for each LTS branch must be created

If a PR is missing the `target:*` label, it will be marked as pending by the angular robot status checks.

PR Approvals

Before a PR can be merged it must be approved by the appropriate reviewer(s).

To ensure that the right people review each change, we set review requests using PullApprove (via `.pullapprove`) and require that each PR has at least one approval from an appropriate code owner.

If the PR author is a code owner themselves, the approval can come from *any* repo collaborator (person with write access). In any case, the reviewer should actually look through the code and provide feedback if necessary.

Note that approved state does not mean a PR is ready to be merged. For example, a reviewer might approve the PR but request a minor tweak that doesn't need further review, e.g., a rebase or small uncontroversial change. Only the `action: merge` label means that the PR is ready for merging.

Special Labels

`cla: yes`, `cla: no`

- ***Who adds it:*** *@googlebot, or a Googler manually overriding the status in case the bot got it wrong.*
- ***Who removes it:*** *@googlebot.*

Managed by googlebot. Indicates whether a PR has a CLA on file for its author(s). Only issues with `cla:yes` should be merged into master.

`aio: preview`

- ***Who adds it:*** *Any team member. (Typically the author or a reviewer.)*
- ***Who removes it:*** *Any team member. (Typically, whoever added it.)*

Applying this label to a PR makes the angular.io preview available regardless of the author. More info

`action: merge-assistance`

- ***Who adds it:*** *Any team member.*
- ***Who removes it:*** *Any team member.*

This label can be added to let the caretaker know that the PR needs special attention. There should always be a comment added to the PR to explain why the caretaker's assistance is needed. The comment should be formatted like this: `merge-assistance: <explain what kind of assistance you need, and if not obvious why>`

For example, the PR owner might not be a Googler and needs help to run g3sync; or one of the checks is failing due to external causes and the PR should still be merged.

`action: rerun CI at HEAD`

- ***Who adds it:*** *Any team member.*
- ***Who removes it:*** *The Angular Bot, once it triggers the CI rerun.*

This label can be added to instruct the Angular Bot to rerun the CI jobs for the PR at latest HEAD of the branch it targets.