Documenting a feature for a release

Each major Kubernetes release introduces new features that require documentation. New releases also bring updates to existing features and documentation (such as upgrading a feature from alpha to beta).

Generally, the SIG responsible for a feature submits draft documentation of the feature as a pull request to the appropriate development branch of the `kubernetes/website` repository, and someone on the SIG Docs team provides editorial feedback or edits the draft directly. This section covers the branching conventions and process used during a release by both groups.

To learn about announcing features on the blog, read post-release communications.

For documentation contributors

In general, documentation contributors don't write content from scratch for a release. Instead, they work with the SIG creating a new feature to refine the draft documentation and make it release ready.

After you've chosen a feature to document or assist, ask about it in the `#sig-docs` Slack channel, in a weekly SIG Docs meeting, or directly on the PR filed by the feature SIG. If you're given the go-ahead, you can edit into the PR using one of the techniques described in Commit into another person's PR.

Find out about upcoming features

To find out about upcoming features, attend the weekly SIG Release meeting (see the community page for upcoming meetings) and monitor the release-specific documentation in the kubernetes/sig-release repository. Each release has a sub-directory in the /sig-release/tree/master/releases/ directory. The sub-directory contains a release schedule, a draft of the release notes, and a document listing each person on the release team.

The release schedule contains links to all other documents, meetings, meeting minutes, and milestones relating to the release. It also contains information about the goals and timeline of the release, and any special processes in place for this release. Near the bottom of the document, several release-related terms are defined.

This document also contains a link to the **Feature tracking sheet**, which is the official way to find out about all new features scheduled to go into the release.

The release team document lists who is responsible for each release role. If it's not clear who to talk to about a specific feature or question you have, either attend the release meeting to ask your question, or contact the release lead so that they can redirect you.

The release notes draft is a good place to find out about specific features, changes, deprecations, and more about the release. The content is not finalized until late in the release cycle, so use caution.

Feature tracking sheet

The feature tracking sheet for a given Kubernetes release lists each feature that is planned for a release. Each line item includes the name of the feature, a link to the feature's main GitHub issue, its stability level (Alpha, Beta, or Stable), the SIG and individual responsible for implementing it, whether it needs docs, a draft release note for the feature, and whether it has been merged. Keep the following in mind:

- Beta and Stable features are generally a higher documentation priority than Alpha features.
- It's hard to test (and therefore to document) a feature that hasn't been merged, or is at least considered feature-complete in its PR.
- Determining whether a feature needs documentation is a manual process. Even if a feature is not marked as needing docs, you may need to document the feature.

For developers or other SIG members

This section is information for members of other Kubernetes SIGs documenting new features for a release.

If you are a member of a SIG developing a new feature for Kubernetes, you need to work with SIG Docs to be sure your feature is documented in time for the release. Check the feature tracking spreadsheet or check in the `#sig-release`

Kubernetes Slack channel to verify scheduling details and deadlines.

Open a placeholder PR

1. Open a **draft** pull request against the `dev-1.35` branch in the `kubernetes/website` repository, with a small commit that you will amend later. To create a draft pull request, use the **Create Pull Request** drop-down and select **Create Draft Pull Request**, then click **Draft Pull Request**.
2. Edit the pull request description to include links to kubernetes/kubernetes PR(s) and kubernetes/enhancements issue(s).
3. Leave a comment on the related kubernetes/enhancements issue with a link to the PR to notify the docs person managing this release that the feature docs are coming and should be tracked for the release.

If your feature does not need any documentation changes, make sure the sig-release team knows this, by mentioning it in the `#sig-release` Slack channel. If the feature does need documentation but the PR is not created, the feature may be removed from the milestone.

PR ready for review

When ready, populate your placeholder PR with feature documentation and change the state of the PR from draft to **ready for review**. To mark a pull request as ready for review, navigate to the merge box and click **Ready for review**.

Do your best to describe your feature and how to use it. If you need help structuring your documentation, ask in the `#sig-docs` Slack channel.

When you complete your content, the documentation person assigned to your feature reviews it. To ensure technical accuracy, the content may also require a technical review from corresponding SIG(s). Use their suggestions to get the content to a release ready state.

If your feature needs documentation and the first draft content is not received, the feature may be removed from the milestone.

Feature gates

If your feature is an Alpha or Beta feature and is behind a feature gate, you need a feature gate file for it inside `content/en/docs/reference/command-line-tools-reference/feature-gates/`. The name of the file should be the name of the feature gate with `.md` as the suffix. You can look at other files already in the same directory for a hint about what yours should look like. Usually a single paragraph is enough; for longer explanations, add documentation elsewhere and link to that.

Also, to ensure your feature gate appears in the Alpha/Beta Feature gates table, include the following details in the front matter of your Markdown description file:

```yaml
stages:
  - stage: <alpha/beta/stable/deprecated>  # Specify the development stage of the feature gate
    defaultValue: <true or false>     # Set to true if enabled by default, false otherwise
    fromVersion: <Version>            # Version from which the feature gate is available
    toVersion: <Version>              # (Optional) The version until which the feature gate is available
```

With net new feature gates, a separate description of the feature gate is also required; create a new Markdown file inside `content/en/docs/reference/command-line-tools-reference/feature-gates/` (use other files as a template).

When you change a feature gate from disabled-by-default to enabled-by-default, you may also need to change other documentation (not just the list of feature gates). Watch out for language such as "The `exampleSetting` field is a beta field and disabled by default. You can enable it by enabling the `ProcessExampleThings` feature gate."

If your feature is GA'ed or deprecated, include an additional `stage` entry within the `stages` block in the description file. Ensure that the Alpha and Beta stages remain intact. This step transitions the feature gate from the Feature gates for Alpha/Beta table to Feature gates for graduated or deprecated features table. For example:

```yaml
stages:
  - stage: alpha
    defaultValue: false
    fromVersion: "1.12"
    toVersion: "1.12"
  - stage: beta
    defaultValue: true
    fromVersion: "1.13"
  # Added a `toVersion` to the previous stage.
```

```yaml
    toVersion: "1.18"
  # Added 'stable' stage block to existing stages.
  - stage: stable
    defaultValue: true
    fromVersion: "1.19"
    toVersion: "1.27"
```

Eventually, Kubernetes will stop including the feature gate at all. To signify the removal of a feature gate, include `removed: true` in the front matter of the respective description file. Making that change means that the feature gate information moves from the Feature gates for graduated or deprecated features section to a dedicated page titled Feature Gates (removed), including its description.

All PRs reviewed and ready to merge

If your PR has not yet been merged into the `dev-1.35` branch by the release deadline, work with the docs person managing the release to get it in by the deadline. If your feature needs documentation and the docs are not ready, the feature may be removed from the milestone.

Opening a pull request

Note:

Code developers: If you are documenting a new feature for an upcoming Kubernetes release, see Document a new feature.

To contribute new content pages or improve existing content pages, open a pull request (PR). Make sure you follow all the requirements in the Before you begin section.

If your change is small, or you're unfamiliar with git, read Changes using GitHub to learn how to edit a page.

If your changes are large, read Work from a local fork to learn how to make changes locally on your computer.

Changes using GitHub

If you're less experienced with git workflows, here's an easier method of opening a pull request. Figure 1 outlines the steps and the details follow.

flowchart LR A([fa:fa-user New
Contributor]) --- id1[(kubernetes/website
GitHub)] subgraph tasks[Changes using GitHub] direction TB 0[ ] -.- 1[1. Edit this page] --> 2[2. Use GitHub markdown
editor to make changes] 2 --> 3[3. Select Commit changes...] end subgraph tasks2[ ] direction TB 4[4. Select Propose file change] --> 5[5. Select Create pull request] --> 6[6. Fill in Open a pull request] 6 --> 7[7. Select Create pull request] end id1 --> tasks --> tasks2 classDef grey fill:#dddddd,stroke:#ffffff,stroke-width:px,color:#000000, font-size:15px; classDef white fill:#ffffff,stroke:#000,stroke-width:px,color:#000,font-weight:bold classDef k8s fill:#326ce5,stroke:#fff,stroke-width:1px,color:#fff; classDef spacewhite fill:#ffffff,stroke:#fff,stroke-width:0px,color:#000 class A,1,2,3,4,5,6,7 grey class 0 spacewhite class tasks,tasks2 white class id1 k8s

Figure 1. Steps for opening a PR using GitHub.

1. On the page where you see the issue, select the **Edit this page** option in the right-hand side navigation panel.

2. Make your changes in the GitHub markdown editor.

3. On the right above the editor, Select **Commit changes**. In the first field, give your commit message a title. In the second field, provide a description.

   Note:

   Do not use any GitHub Keywords in your commit message. You can add those to the pull request description later.

4. Select **Propose changes**.

5. Select **Create pull request**.

6. The **Open a pull request** screen appears. Fill in the form:

   - The **Add a title** field of the pull request defaults to the commit summary. You can change it if needed.
   - The **Add a description** field contains your extended commit message, if you have one, and some template text. Add the details the template text asks for, then delete the extra template text.
   - Leave the **Allow edits from maintainers** checkbox selected.

   Note:

   PR descriptions are a great way to help reviewers understand your change. For more information, see [Opening a PR]().

7. Select **Create pull request**.

Addressing feedback in GitHub

Before merging a pull request, Kubernetes community members review and approve it. The `k8s-ci-robot` suggests reviewers based on the nearest owner mentioned in the pages. If you have someone specific in mind, leave a comment with their GitHub username in it.

If a reviewer asks you to make changes:

1. Go to the **Files changed** tab.
2. Select the pencil (edit) icon on any files changed by the pull request.
3. Make the changes requested.
4. Commit the changes.

If you are waiting on a reviewer, reach out once every 7 days. You can also post a message in the `#sig-docs` Slack channel.

When your review is complete, a reviewer merges your PR and your changes go live a few minutes later.

Work from a local fork

If you're more experienced with git, or if your changes are larger than a few lines, work from a local fork.

Make sure you have [git]() installed on your computer. You can also use a git UI application.

Figure 2 shows the steps to follow when you work from a local fork. The details for each step follow.

flowchart LR 1[Fork the kubernetes/website
repository] --> 2[Create local clone
and set upstream] subgraph changes[Your changes] direction TB S[ ] -.- 3[Create a branch
example: my_new_branch] --> 3a[Make changes using
text editor] --> 4["Preview your changes
locally using Hugo
(localhost:1313)
or build container image"] end subgraph changes2[Commit / Push] direction TB T[ ] -.- 5[Commit your
changes] --> 6[Push commit to
origin/my_new_branch] end 2 --> changes --> changes2 classDef grey fill:#dddddd,stroke:#ffffff,stroke-width:px,color:#000000, font-size:15px; classDef white fill:#ffffff,stroke:#000,stroke-width:px,color:#000,font-weight:bold classDef k8s fill:#326ce5,stroke:#fff,stroke-width:1px,color:#fff; classDef spacewhite fill:#ffffff,stroke:#fff,stroke-width:0px,color:#000 class 1,2,3,3a,4,5,6 grey class S,T spacewhite class changes,changes2 white

Figure 2. Working from a local fork to make your changes.

Fork the kubernetes/website repository

1. Navigate to the [`kubernetes/website`]() repository.
2. Select **Fork**.

Create a local clone and set the upstream

1. In a terminal window, clone your fork and update the Docsy Hugo theme:

   ```
   git clone git@github.com:<github_username>/website
   cd website
   ```

2. Navigate to the new `website` directory. Set the `kubernetes/website` repository as the `upstream` remote:

   ```
   cd website

   git remote add upstream https://github.com/kubernetes/website.git
   ```

3. Confirm your `origin` and `upstream` repositories:

   ```
   git remote -v
   ```

   Output is similar to:

   ```
   origin  git@github.com:<github_username>/website.git (fetch)
   origin  git@github.com:<github_username>/website.git (push)
   upstream        https://github.com/kubernetes/website.git (fetch)
   upstream        https://github.com/kubernetes/website.git (push)
   ```

4. Fetch commits from your fork's `origin/main` and `kubernetes/website`'s `upstream/main`:

   ```
   git fetch origin
   git fetch upstream
   ```

   This makes sure your local repository is up to date before you start making changes.

   Note:

   This workflow is different than the Kubernetes Community GitHub Workflow. You do not need to merge your local copy of `main` with `upstream/main` before pushing updates to your fork.

Create a branch

1. Decide which branch to base your work on:

   - For improvements to existing content, use `upstream/main`.
   - For new content about existing features, use `upstream/main`.
   - For localized content, use the localization's conventions. For more information, see localizing Kubernetes documentation.
   - For new features in an upcoming Kubernetes release, use the feature branch. For more information, see documenting for a release.
   - For long-running efforts that multiple SIG Docs contributors collaborate on, like content reorganization, use a specific feature branch created for that effort.

   If you need help choosing a branch, ask in the `#sig-docs` Slack channel.

2. Create a new branch based on the branch identified in step 1. This example assumes the base branch is `upstream/main`:

   ```
   git checkout -b <my_new_branch> upstream/main
   ```

3. Make your changes using a text editor.

At any time, use the `git status` command to see what files you've changed.

Commit your changes

When you are ready to submit a pull request, commit your changes.

1. In your local repository, check which files you need to commit:

   ```
   git status
   ```

   Output is similar to:

```
On branch <my_new_branch>
Your branch is up to date with 'origin/<my_new_branch>'.

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)

modified:   content/en/docs/contribute/new-content/contributing-content.md

no changes added to commit (use "git add" and/or "git commit -a")
```

2. Add the files listed under **Changes not staged for commit** to the commit:

   ```
   git add <your_file_name>
   ```

   Repeat this for each file.

3. After adding all the files, create a commit:

   ```
   git commit -m "Your commit message"
   ```

   Note:

   Do not use any GitHub Keywords in your commit message. You can add those to the pull request description later.

4. Push your local branch and its new commit to your remote fork:

   ```
   git push origin <my_new_branch>
   ```

Preview your changes locally

It's a good idea to preview your changes locally before pushing them or opening a pull request. A preview lets you catch build errors or markdown formatting problems.

You can either build the website's container image or run Hugo locally. Building the container image is slower but displays Hugo shortcodes, which can be useful for debugging.

- Hugo in a container
- Hugo on the command line

Note:

The commands below use Docker as default container engine. Set the `CONTAINER_ENGINE` environment variable to override this behaviour.

1. Build the container image locally
   You only need this step if you are testing a change to the Hugo tool itself

   ```
   # Run this in a terminal (if required)
   make container-image
   ```

2. Fetch submodule dependencies in your local repository:

   ```
   # Run this in a terminal
   make module-init
   ```

3. Start Hugo in a container:

   ```
   # Run this in a terminal
   make container-serve
   ```

4. In a web browser, navigate to `http://localhost:1313`. Hugo watches the changes and rebuilds the site as needed.

5. To stop the local Hugo instance, go back to the terminal and type `Ctrl+C`, or close the terminal window.

Alternately, install and use the `hugo` command on your computer:

1. Install the Hugo (Extended edition) and Node version specified in `website/netlify.toml`.

2. Install any dependencies:

   ```
   npm ci
   ```

3. In a terminal, go to your Kubernetes website repository and start the Hugo server:

   ```
   cd <path_to_your_repo>/website
   make serve
   ```

   If you're on a Windows machine or unable to run the `make` command, use the following command:

   ```
   hugo server --buildFuture
   ```

4. In a web browser, navigate to `http://localhost:1313`. Hugo watches the changes and rebuilds the site as needed.

5. To stop the local Hugo instance, go back to the terminal and type `Ctrl+C`, or close the terminal window.

Open a pull request from your fork to kubernetes/website

Figure 3 shows the steps to open a PR from your fork to the kubernetes/website. The details follow.

Please, note that contributors can mention `kubernetes/website` as `k/website`.

> flowchart LR subgraph first[ ] direction TB 1[1. Go to kubernetes/website repository] --> 2[2. Select New Pull Request] 2 --> 3[3. Select compare across forks] 3 --> 4[4. Select your fork from
> head repository drop-down menu] end subgraph second [ ] direction TB 5[5. Select your branch from
> the compare drop-down menu] --> 6[6. Select Create Pull Request] 6 --> 7[7. Add a description
> to your PR] 7 --> 8[8. Select Create pull request] end first --> second classDef grey
> fill:#dddddd,stroke:#ffffff,stroke-width:px,color:#000000, font-size:15px; classDef white
> fill:#ffffff,stroke:#000,stroke-width:px,color:#000,font-weight:bold class 1,2,3,4,5,6,7,8 grey class first,second white

Figure 3. Steps to open a PR from your fork to the kubernetes/website.

1. In a web browser, go to the `kubernetes/website` repository.

2. Select **New Pull Request**.

3. Select **compare across forks**.

4. From the **head repository** drop-down menu, select your fork.

5. From the **compare** drop-down menu, select your branch.

6. Select **Create Pull Request**.

7. Add a description for your pull request:

   - **Title** (50 characters or less): Summarize the intent of the change.

   - **Description**: Describe the change in more detail.

     - If there is a related GitHub issue, include `Fixes #12345` or `Closes #12345` in the description. GitHub's automation closes the mentioned issue after merging the PR if used. If there are other related PRs, link those as well.
     - If you want advice on something specific, include any questions you'd like reviewers to think about in your description.

8. Select the **Create pull request** button.

Congratulations! Your pull request is available in Pull requests.

After opening a PR, GitHub runs automated tests and tries to deploy a preview using Netlify.

- If the Netlify build fails, select **Details** for more information.
- If the Netlify build succeeds, select **Details** opens a staged version of the Kubernetes website with your changes applied. This is how reviewers check your changes.

GitHub also automatically assigns labels to a PR, to help reviewers. You can add them too, if needed. For more information, see Adding and removing issue labels.

Addressing feedback locally

1. After making your changes, amend your previous commit:

   ```
   git commit -a --amend
   ```

   - `-a`: commits all changes
   - `--amend`: amends the previous commit, rather than creating a new one

2. Update your commit message if needed.

3. Use `git push origin <my_new_branch>` to push your changes and re-run the Netlify tests.

   Note:

   If you use `git commit -m` instead of amending, you must squash your commits before merging.

Changes from reviewers

Sometimes reviewers commit to your pull request. Before making any other changes, fetch those commits.

1. Fetch commits from your remote fork and rebase your working branch:

   ```
   git fetch origin
   git rebase origin/<your-branch-name>
   ```

2. After rebasing, force-push new changes to your fork:

   ```
   git push --force-with-lease origin <your-branch-name>
   ```

Merge conflicts and rebasing

Note:

For more information, see Git Branching - Basic Branching and Merging, Advanced Merging, or ask in the `#sig-docs` Slack channel for help.

If another contributor commits changes to the same file in another PR, it can create a merge conflict. You must resolve all merge conflicts in your PR.

1. Update your fork and rebase your local branch:

   ```
   git fetch origin
   git rebase origin/<your-branch-name>
   ```

   Then force-push the changes to your fork:

   ```
   git push --force-with-lease origin <your-branch-name>
   ```

2. Fetch changes from `kubernetes/website`'s `upstream/main` and rebase your branch:

   ```
   git fetch upstream
   git rebase upstream/main
   ```

3. Inspect the results of the rebase:

   ```
   git status
   ```

   This results in a number of files marked as conflicted.

4. Open each conflicted file and look for the conflict markers: >>>, <<<, and ===. Resolve the conflict and delete the conflict marker.

   Note:

For more information, see [How conflicts are presented]().

5. Add the files to the changeset:

   ```
   git add <filename>
   ```

6. Continue the rebase:

   ```
   git rebase --continue
   ```

7. Repeat steps 2 to 5 as needed.

   After applying all commits, the `git status` command shows that the rebase is complete.

8. Force-push the branch to your fork:

   ```
   git push --force-with-lease origin <your-branch-name>
   ```

   The pull request no longer shows any conflicts.

Squashing commits

Note:

For more information, see [Git Tools - Rewriting History](), or ask in the `#sig-docs` Slack channel for help.

If your PR has multiple commits, you must squash them into a single commit before merging your PR. You can check the number of commits on your PR's **Commits** tab or by running the `git log` command locally.

Note:

This topic assumes `vim` as the command line text editor.

1. Start an interactive rebase:

   ```
   git rebase -i HEAD~<number_of_commits_in_branch>
   ```

   Squashing commits is a form of rebasing. The `-i` switch tells git you want to rebase interactively. `HEAD~<number_of_commits_in_branch` indicates how many commits to look at for the rebase.

   Output is similar to:

   ```
   pick d875112ca Original commit
   pick 4fa167b80 Address feedback 1
   pick 7d54e15ee Address feedback 2

   # Rebase 3d18sf680..7d54e15ee onto 3d183f680 (3 commands)

   ...

   # These lines can be re-ordered; they are executed from top to bottom.
   ```

   The first section of the output lists the commits in the rebase. The second section lists the options for each commit. Changing the word `pick` changes the status of the commit once the rebase is complete.

   For the purposes of rebasing, focus on `squash` and `pick`.

   Note:

   For more information, see [Interactive Mode]().

2. Start editing the file.

   Change the original text:

   ```
   pick d875112ca Original commit
   pick 4fa167b80 Address feedback 1
   pick 7d54e15ee Address feedback 2
   ```

To:

```
pick d875112ca Original commit
squash 4fa167b80 Address feedback 1
squash 7d54e15ee Address feedback 2
```

This squashes commits `4fa167b80 Address feedback 1` and `7d54e15ee Address feedback 2` into `d875112ca Original commit`, leaving only `d875112ca Original commit` as a part of the timeline.

3. Save and exit your file.

4. Push your squashed commit:

```
git push --force-with-lease origin <branch_name>
```

Contribute to other repos

The Kubernetes project contains 50+ repositories. Many of these repositories contain documentation: user-facing help text, error messages, API references or code comments.

If you see text you'd like to improve, use GitHub to search all repositories in the Kubernetes organization. This can help you figure out where to submit your issue or PR.

Each repository has its own processes and procedures. Before you file an issue or submit a PR, read that repository's `README.md`, `CONTRIBUTING.md`, and `code-of-conduct.md`, if they exist.

Most repositories use issue and PR templates. Have a look through some open issues and PRs to get a feel for that team's processes. Make sure to fill out the templates with as much detail as possible when you file issues or PRs.

What's next

- Read Reviewing to learn more about the review process.

Contributing new content

This section contains information you should know before contributing new content.

There are also dedicated pages about submitting case studies and blog articles.

New content task flow

flowchart LR subgraph second[Before you begin] direction TB S[ ] -.- A[Sign the CNCF CLA] --> B[Choose Git branch] B --> C[One language per PR] C --> F[Check out
contributor tools] end subgraph first[Contributing Basics] direction TB T[ ] -.- D[Write docs in markdown
and build site with Hugo] --- E[source in GitHub] E --- G['/content/../docs' folder contains docs
for multiple languages] G --- H[Review Hugo page content
types and shortcodes] end first ----> second classDef grey fill:#dddddd,stroke:#ffffff,stroke-width:px,color:#000000, font-size:15px; classDef white fill:#ffffff,stroke:#000,stroke-width:px,color:#000,font-weight:bold classDef spacewhite fill:#ffffff,stroke:#fff,stroke-width:0px,color:#000 class A,B,C,D,E,F,G,H grey class S,T spacewhite class first,second white

Figure - Contributing new content preparation

The figure above depicts the information you should know prior to submitting new content. The information details follow.

Contributing basics

- Write Kubernetes documentation in Markdown and build the Kubernetes site using Hugo.
- Kubernetes documentation uses CommonMark as its flavor of Markdown.
- The source is in GitHub. You can find Kubernetes documentation at `/content/en/docs/`. Some of the reference documentation is automatically generated from scripts in the `update-imported-docs/` directory.

- Page content types describe the presentation of documentation content in Hugo.
- You can use Docsy shortcodes or custom Hugo shortcodes to contribute to Kubernetes documentation.
- In addition to the standard Hugo shortcodes, we use a number of custom Hugo shortcodes in our documentation to control the presentation of content.
- Documentation source is available in multiple languages in `/content/`. Each language has its own folder with a two-letter code determined by the ISO 639-1 standard . For example, English documentation source is stored in `/content/en/docs/`.
- For more information about contributing to documentation in multiple languages or starting a new translation, see localization.

Before you begin

Sign the CNCF CLA

All Kubernetes contributors **must** read the Contributor guide and sign the Contributor License Agreement (CLA) .

Pull requests from contributors who haven't signed the CLA fail the automated tests. The name and email you provide must match those found in your `git config`, and your git name and email must match those used for the CNCF CLA.

Choose which Git branch to use

When opening a pull request, you need to know in advance which branch to base your work on.

Scenario	Branch
Existing or new English language content for the current release	`main`
Content for a feature change release	The branch which corresponds to the major and minor version the feature change is in, using the pattern `dev-<version>`. For example, if a feature changes in the `v1.35` release, then add documentation changes to the `dev-1.35` branch.
Content in other languages (localizations)	Use the localization's convention. See the Localization branching strategy for more information.

If you're still not sure which branch to choose, ask in `#sig-docs` on Slack.

Note:

If you already submitted your pull request and you know that the base branch was wrong, you (and only you, the submitter) can change it.

Languages per PR

Limit pull requests to one language per PR. If you need to make an identical change to the same code sample in multiple languages, open a separate PR for each language.

Tools for contributors

The doc contributors tools directory in the `kubernetes/website` repository contains tools to help your contribution journey go more smoothly.

What's next

- Read about submitting blog articles.

- Opening a pull request
- Documenting a feature for a release
- Submitting case studies

Submitting case studies

Case studies highlight how organizations are using Kubernetes to solve real-world problems. The Kubernetes marketing team and members of the CNCF collaborate with you on all case studies.

Case studies require extensive review before they're approved.

Submit a case study

Have a look at the source for the existing case studies.

Refer to the case study guidelines and submit your request as outlined in the guidelines.