Localizing Kubernetes documentation

This page shows you how to localize the docs for a different language.

Contribute to an existing localization

You can help add or improve the content of an existing localization. In Kubernetes Slack, you can find a channel for each localization. There is also a general SIG Docs Localizations Slack channel where you can say hello.

Note:

For extra details on how to contribute to a specific localization, look for a localized version of this page.

Find your two-letter language code

First, consult the ISO 639-1 standard to find your localization's two-letter language code. For example, the two-letter code for Korean is `ko`.

Some languages use a lowercase version of the country code as defined by the ISO-3166 along with their language codes. For example, the Brazilian Portuguese language code is `pt-br`.

Fork and clone the repo

First, create your own fork of the kubernetes/website repository.

Then, clone your fork and `cd` into it:

```
git clone https://github.com/<username>/website
cd website
```

The website content directory includes subdirectories for each language. The localization you want to help out with is inside `content/<two-letter-code>`.

Suggest changes

Create or update your chosen localized page based on the English original. See localize content for more details.

If you notice a technical inaccuracy or other problem with the upstream (English) documentation, you should fix the upstream documentation first and then repeat the equivalent fix by updating the localization you're working on.

Limit changes in a pull requests to a single localization. Reviewing pull requests that change content in multiple localizations is problematic.

Follow Suggesting Content Improvements to propose changes to that localization. The process is similar to proposing changes to the upstream (English) content.

Start a new localization

If you want the Kubernetes documentation localized into a new language, here's what you need to do.

Because contributors can't approve their own pull requests, you need *at least two contributors* to begin a localization.

All localization teams must be self-sufficient. The Kubernetes website is happy to host your work, but it's up to you to translate it and keep existing localized content current.

You'll need to know the two-letter language code for your language. Consult the ISO 639-1 standard to find your localization's two-letter language code. For example, the two-letter code for Korean is `ko`.

If the language you are starting a localization for is spoken in various places with significant differences between the variants, it might make sense to combine the lowercased ISO-3166 country code with the language two-letter code. For example, Brazilian Portuguese is localized as `pt-br`.

When you start a new localization, you must localize all the minimum required content before the Kubernetes project can publish your changes to the live website.

SIG Docs can help you work on a separate branch so that you can incrementally work towards that goal.

Find community

Let Kubernetes SIG Docs know you're interested in creating a localization! Join the SIG Docs Slack channel and the SIG Docs Localizations Slack channel. Other localization teams are happy to help you get started and answer your questions.

Please also consider participating in the SIG Docs Localization Subgroup meeting. The mission of the SIG Docs localization subgroup is to work across the SIG Docs localization teams to collaborate on defining and documenting the processes for creating localized contribution guides. In addition, the SIG Docs localization subgroup looks for opportunities to create and share common tools across localization teams and identify new requirements for the SIG Docs Leadership team. If you have questions about this meeting, please inquire on the SIG Docs Localizations Slack channel.

You can also create a Slack channel for your localization in the `kubernetes/community` repository. For an example of adding a Slack channel, see the PR for adding a channel for Persian.

Join the Kubernetes GitHub organization

When you've opened a localization PR, you can become members of the Kubernetes GitHub organization. Each person on the team needs to create their own Organization Membership Request in the `kubernetes/org` repository.

Add your localization team in GitHub

Next, add your Kubernetes localization team to `sig-docs/teams.yaml`. For an example of adding a localization team, see the PR to add the Spanish localization team.

Members of `@kubernetes/sig-docs-**-owners` can approve PRs that change content within (and only within) your localization directory: `/content/**/`. For each localization, The `@kubernetes/sig-docs-**-reviews` team automates review assignments for new PRs. Members of `@kubernetes/website-maintainers` can create new localization branches to coordinate translation efforts. Members of `@kubernetes/website-milestone-maintainers` can use the `/milestone` Prow command to assign a milestone to issues or PRs.

Configure the workflow

Next, add a GitHub label for your localization in the `kubernetes/test-infra` repository. A label lets you filter issues and pull requests for your specific language.

For an example of adding a label, see the PR for adding the Italian language label.

Modify the site configuration

The Kubernetes website uses Hugo as its web framework. The website's Hugo configuration resides in the `hugo.toml` file. You'll need to modify `hugo.toml` to support a new localization.

Add a configuration block for the new language to `hugo.toml` under the existing `[languages]` block. The German block, for example, looks like:

```
[languages.de]
title = "Kubernetes"
languageName = "Deutsch (German)"
weight = 5
contentDir = "content/de"
languagedirection = "ltr"

[languages.de.params]
time_format_blog = "02.01.2006"
language_alternatives = ["en"]
description = "Produktionsreife Container-Orchestrierung"
languageNameLatinScript = "Deutsch"
```

The language selection bar lists the value for `languageName`. Assign "language name in native script and language (English language name in Latin script)" to `languageName`. For example, `languageName = "한국어 (Korean)"` or `languageName = "Deutsch (German)"`.

`languageNameLatinScript` can be used to access the language name in Latin script and use it in the theme. Assign "language name in latin script" to `languageNameLatinScript`. For example, `languageNameLatinScript ="Korean"` or `languageNameLatinScript = "Deutsch"`.

The `weight` parameter determines the order of languages in the language selection bar. A lower weight takes precedence, resulting in the language appearing first. When assigning the `weight` parameter, it is important to examine the existing languages block and adjust their weights to ensure they are in a sorted order relative to all languages, including any newly added language.

For more information about Hugo's multilingual support, see "Multilingual Mode".

Add a new localization directory

Add a language-specific subdirectory to the `content` folder in the repository. For example, the two-letter code for German is `de`:

```
mkdir content/de
```

You also need to create a directory inside `i18n` for localized strings; look at existing localizations for an example.

For example, for German the strings live in `i18n/de/de.toml`.

Localize the community code of conduct

Open a PR against the `cncf/foundation` repository to add the code of conduct in your language.

Set up the OWNERS files

To set the roles of each user contributing to the localization, create an `OWNERS` file inside the language-specific subdirectory with:

- **reviewers**: A list of kubernetes teams with reviewer roles, in this case,
- the `sig-docs-**-reviews` team created in Add your localization team in GitHub.
- **approvers**: A list of kubernetes teams with approvers roles, in this case,
- the `sig-docs-**-owners` team created in Add your localization team in GitHub.
- **labels**: A list of GitHub labels to automatically apply to a PR, in this case, the language label created in Configure the workflow.

More information about the `OWNERS` file can be found at go.k8s.io/owners.

The Spanish OWNERS file, with language code `es`, looks like this:

```
# See the OWNERS docs at https://go.k8s.io/owners
```

```
# This is the localization project for Spanish.# Teams and members are visible at https://github.com/orgs/kubernetes/teams.reviewe
```

After adding the language-specific `OWNERS` file, update the root `OWNERS_ALIASES` file with the new Kubernetes teams for the localization, `sig-docs-**-owners` and `sig-docs-**-reviews`.

For each team, add the list of GitHub users requested in Add your localization team in GitHub, in alphabetical order.

```
--- a/OWNERS_ALIASES
+++ b/OWNERS_ALIASES@@ -48,6 +48,14 @@ aliases:     - stewart-yu     - xiangpengzhao     - zhangxiaoyu-zidif+  sig-docs-es-owners:
```

Open a pull request

Next, open a pull request (PR) to add a localization to the `kubernetes/website` repository. The PR must include all the minimum required content before it can be approved.

For an example of adding a new localization, see the PR to enable docs in French.

Add a localized README file

To guide other localization contributors, add a new `README-**.md` to the top level of kubernetes/website, where `**` is the two-letter language code. For example, a German README file would be `README-de.md`.

Guide localization contributors in the localized `README-**.md` file. Include the same information contained in `README.md` as well as:

- A point of contact for the localization project
- Any information specific to the localization

After you create the localized README, add a link to the file from the main English `README.md`, and include contact information in English. You can provide a GitHub ID, email address, Slack channel, or another method of contact. You must also provide a link to your localized Community Code of Conduct.

Launch your new localization

When a localization meets the requirements for workflow and minimum output, SIG Docs does the following:

- Enables language selection on the website.
- Publicizes the localization's availability through Cloud Native Computing Foundation(CNCF) channels, including the Kubernetes blog.

Localize content

Localizing *all* the Kubernetes documentation is an enormous task. It's okay to start small and expand over time.

Minimum required content

At a minimum, all localizations must include:

Description	URLs
Home	All heading and subheading URLs
Setup	All heading and subheading URLs
Tutorials	Kubernetes Basics, Hello Minikube
Site strings	All site strings in a new localized TOML file
Releases	All heading and subheading URLs

Translated documents must reside in their own `content/**/` subdirectory, but otherwise, follow the same URL path as the English source. For example, to prepare the Kubernetes Basics tutorial for translation into German, create a subdirectory under the `content/de/` directory and copy the English source or directory:

```
mkdir -p content/de/docs/tutorials
cp -ra content/en/docs/tutorials/kubernetes-basics/ content/de/docs/tutorials/
```

Translation tools can speed up the translation process. For example, some editors offer plugins to quickly translate text.

Caution:

Machine-generated translation is insufficient on its own. Localization requires extensive human review to meet minimum standards of quality.

To ensure accuracy in grammar and meaning, members of your localization team should carefully review all machine-generated translations before publishing.

Localize SVG images

The Kubernetes project recommends using vector (SVG) images where possible, as these are much easier for a localization team to edit. If you find a raster image that needs localizing, consider first redrawing the English version as a vector image, and then localize that.

When translating text within SVG (Scalable Vector Graphics) images, it's essential to follow certain guidelines to ensure accuracy and maintain consistency across different language versions. SVG images are commonly used in the Kubernetes documentation to illustrate concepts, workflows, and diagrams.

1. **Identifying translatable text**: Start by identifying the text elements within the SVG image that need to be translated. These elements typically include labels, captions, annotations, or any text that conveys information.

2. **Editing SVG files**: SVG files are XML-based, which means they can be edited using a text editor. However, it's important to note that most of the documentation images in Kubernetes already convert text to curves to avoid font compatibility issues. In such cases, it is recommended to use specialized SVG editing software, such as Inkscape, for editing, open the SVG file and locate the text elements that require translation.

3. **Translating the text**: Replace the original text with the translated version in the desired language. Ensure the translated text accurately conveys the intended meaning and fits within the available space in the image. The Open Sans font family should be used when working with languages that use the Latin alphabet. You can download the Open Sans typeface from here: Open Sans Typeface.

4. **Converting text to curves**: As already mentioned, to address font compatibility issues, it is recommended to convert the translated text to curves or paths. Converting text to curves ensures that the final image displays the translated text correctly, even if the user's system does not have the exact font used in the original SVG.

5. **Reviewing and testing**: After making the necessary translations and converting text to curves, save and review the updated SVG image to ensure the text is properly displayed and aligned. Check Preview your changes locally.

Source files

Localizations must be based on the English files from a specific release targeted by the localization team. Each localization team can decide which release to target, referred to as the *target version* below.

To find source files for your target version:

1. Navigate to the Kubernetes website repository at https://github.com/kubernetes/website.

2. Select a branch for your target version from the following table:

Target version	Branch
Latest version	`main`
Previous version	`release-1.33`
Next version	`dev-1.35`

The `main` branch holds content for the current release `v1.34`. The release team creates a `release-1.34` branch before the next release: v1.35.

Site strings in i18n

Localizations must include the contents of `i18n/en/en.toml` in a new language-specific file. Using German as an example: `i18n/de/de.toml`.

Add a new localization directory and file to `i18n/`. For example, with German (`de`):

```
mkdir -p i18n/de
cp i18n/en/en.toml i18n/de/de.toml
```

Revise the comments at the top of the file to suit your localization, then translate the value of each string. For example, this is the German-language placeholder text for the search form:

```
[ui_search]
other = "Suchen"
```

Localizing site strings lets you customize site-wide text and features: for example, the legal copyright text in the footer on each page.

Language-specific localization guide

As a localization team, you can formalize the best practices your team follows by creating a language-specific localization guide.

For example, see the Korean Localization Guide, which includes content on the following subjects:

- Sprint cadence and releases
- Branch strategy
- Pull request workflow
- Style guide
- Glossary of localized and non-localized terms
- Markdown conventions
- Kubernetes API object terminology

Language-specific Zoom meetings

If the localization project needs a separate meeting time, contact a SIG Docs Co-Chair or Tech Lead to create a new reoccurring Zoom meeting and calendar invite. This is only needed when the team is large enough to sustain and require a separate meeting.

Per CNCF policy, the localization teams must upload their meetings to the SIG Docs YouTube playlist. A SIG Docs Co-Chair or Tech Lead can help with the process until SIG Docs automates it.

Branch strategy

Because localization projects are highly collaborative efforts, we encourage teams to work in shared localization branches - especially when starting out and the localization is not yet live.

To collaborate on a localization branch:

1. A team member of @kubernetes/website-maintainers opens a localization branch from a source branch on https://github.com/kubernetes/website.

   Your team approvers joined the `@kubernetes/website-maintainers` team when you added your localization team to the `kubernetes/org` repository.

   We recommend the following branch naming scheme:

   ```
   dev-<source version>-<language code>.<team milestone>
   ```

   For example, an approver on a German localization team opens the localization branch `dev-1.12-de.1` directly against the `kubernetes/website` repository, based on the source branch for Kubernetes v1.12.

2. Individual contributors open feature branches based on the localization branch.

   For example, a German contributor opens a pull request with changes to `kubernetes:dev-1.12-de.1` from `username:local-branch-name`.

3. Approvers review and merge feature branches into the localization branch.

4. Periodically, an approver merges the localization branch with its source branch by opening and approving a new pull request. Be sure to squash the commits before approving the pull request.

Repeat steps 1-4 as needed until the localization is complete. For example, subsequent German localization branches would be: `dev-1.12-de.2`, `dev-1.12-de.3`, etc.

Teams must merge localized content into the same branch from which the content was sourced. For example:

- A localization branch sourced from `main` must be merged into `main`.
- A localization branch sourced from `release-1.33` must be merged into `release-1.33`.

Note:

If your localization branch was created from `main` branch, but it is not merged into `main` before the new release branch `release-1.34` created, merge it into both `main` and new release branch `release-1.34`. To merge your localization branch into the new release branch `release-1.34`, you need to switch the upstream branch of your localization branch to `release-1.34`.

At the beginning of every team milestone, it's helpful to open an issue comparing upstream changes between the previous localization branch and the current localization branch. There are two scripts for comparing upstream changes.

- `upstream_changes.py` is useful for checking the changes made to a specific file. And
- `diff_l10n_branches.py` is useful for creating a list of outdated files for a specific localization branch.

While only approvers can open a new localization branch and merge pull requests, anyone can open a pull request for a new localization branch. No special permissions are required.

For more information about working from forks or directly from the repository, see "fork and clone the repo".

Upstream contributions

SIG Docs welcomes upstream contributions and corrections to the English source.