Blog guidelines

These guidelines cover the main Kubernetes blog and the Kubernetes contributor blog.

All blog content must also adhere to the overall policy in the content guide.

Before you begin

Make sure you are familiar with the introduction sections of contributing to Kubernetes blogs, not just to learn about the two official blogs and the differences between them, but also to get an overview of the process.

Original content

The Kubernetes project accepts **original content only**, in English.

Note:

The Kubernetes project cannot accept content for the blog if it has already been submitted or published outside of the Kubernetes project.

The official blogs are not available as a medium to repurpose existing content from any third party as new content.

This restriction even carries across to promoting other Linux Foundation and CNCF projects. Many CNCF projects have their own blog. These are often a better choice for posts about a specific project, even if that other project is designed specifically to work with Kubernetes (or with Linux, etc).

Relevant content

Articles must contain content that applies broadly to the Kubernetes community. For example, a submission should focus on upstream Kubernetes as opposed to vendor-specific configurations. For articles submitted to the main blog that are not mirror articles, hyperlinks in the article should commonly be to the official Kubernetes documentation. When making external references, links should be diverse - for example, a submission shouldn't contain only links back to a single company's blog.

The official Kubernetes blogs are **not** the place for vendor pitches or for articles that promote a specific solution from outside Kubernetes.

Sometimes this is a delicate balance. You can ask in Slack (#sig-docs-blog) for guidance on whether a post is appropriate for the Kubernetes blog and / or contributor blog - don't hesitate to reach out.

The content guide applies unconditionally to blog articles and the PRs that add them. Bear in mind that some restrictions in the guide state that they are only relevant to documentation; those marked restrictions don't apply to blog articles.

Localization

The website is localized into many languages; English is the “upstream” for all the other localizations. Even if you speak another language and would be happy to provide a localization, that should be in a separate pull request (see languages per PR).

Copyright and reuse

You must write original content and you must have permission to license that content to the Cloud Native Computing Foundation (so that the Kubernetes project can legally publish it). This means that not only is direct plagiarism forbidden, you cannot write a blog article if you don't have permission to meet the CNCF copyright license conditions (for example, if your employer has a policy about intellectual property that restricts what you are allowed to do).

The license for the blog allows commercial use of the content for commercial purposes, but not the other way around.

Special interest groups and working groups

Topics related to participation in or results of Kubernetes SIG activities are always on topic (see the work in the Contributor Comms Team for support on these posts).

The project typically mirrors these articles to both blogs.

National restrictions on content

The Kubernetes website has an Internet Content Provider (ICP) licence from the government of China. Although it's unlikely to be a problem, Kubernetes cannot publish articles that would be blocked by the Chinese government's official filtering of internet content.

Blog-specific content guidance

As well as the general style guide, blog articles should (not must) align to the blog-specific style recommendations.

The remainder of this page is additional guidance; these are not strict rules that articles must follow, but reviewers are likely to (and should) ask for edits to articles that are obviously not aligned with the recommendations here.

Diagrams and illustrations

For illustrations - including diagrams or charts - use the figure shortcode where feasible. You should set an `alt` attribute for accessibility.

Use vector images for illustrations, technical diagrams and similar graphics; SVG format is recommended as a strong preference.

Articles that use raster images for illustrations are more difficult to maintain and in some cases the blog team may ask authors to revise the article before it could be published.

Timelessness

Blog posts should aim to be future proof

- Given the development velocity of the project, SIG Docs prefers *timeless* writing: content that won't require updates to stay accurate for the reader.
- It can be a better choice to add a tutorial or update official documentation than to write a high level overview as a blog post.
- Consider concentrating the long technical content as a call to action of the blog post, and focus on the problem space or why readers should care.

Content examples

Here are some examples of content that is appropriate for the main Kubernetes blog:

- Announcements about new Kubernetes capabilities
- Explanations of how to achieve an outcome using Kubernetes; for example, tell us about your low-toil improvement on the basic idea of a rolling deploy
- Comparisons of several different software options that have a link to Kubernetes and cloud native. It's OK to have a link to one of these options so long as you fully disclose your conflict of interest / relationship.
- Stories about problems or incidents, and how you resolved them
- Articles discussing building a cloud native platform for specific use cases
- Your opinion about the good or bad points about Kubernetes
- Announcements and stories about non-core Kubernetes, such as the Gateway API
- Post-release announcements and updates
- Messages about important Kubernetes security vulnerabilities
- Kubernetes projects updates
- Tutorials and walkthroughs
- Thought leadership around Kubernetes and cloud native
- The components of Kubernetes are purposely modular, so writing about existing integration points like CNI and CSI are on topic. Provided you don't write a vendor pitch, you can also write about what is on the other end of these integrations.

Here are some examples of content that is appropriate for the Kubernetes contributor blog:

- articles about how to test your change to Kubernetes code
- content around non-code contribution
- discussions about alpha features where the design is still under discussion
- "Meet the team" articles about working groups, special interest groups, etc.
- a guide about how to write secure code that will become part of Kubernetes itself
- articles about maintainer summits and the outcome of those summits

Examples of content that wouldn't be accepted

However, the project will not publish:

- vendor pitches
- an article you've published elsewhere, even if only to your own low-traffic blog
- large chunks of example source code with only a minimal explanation
- updates about an external project that works with our relies on Kubernetes (put those on the external project's own blog)
- articles about using Kubernetes with a specific cloud provider
- articles that criticise specific people, groups of people, or businesses
- articles that have important technical mistakes or misleading details (for example: if you recommend turning off an important security control in production clusters, because it can be inconvenient, the Kubernetes project is likely to reject the article).

Contributing to Kubernetes blogs

There are two official Kubernetes blogs, and the CNCF has its own blog where you can cover Kubernetes too. For the main Kubernetes blog, we (the Kubernetes project) like to publish articles with different perspectives and special focuses, that have a link to Kubernetes.

With only a few special case exceptions, we only publish content that hasn't been submitted or published anywhere else.

Read the blog guidelines for more about that aspect.

Official Kubernetes blogs

Main blog

The main Kubernetes blog is used by the project to communicate new features, community reports, and any news that might be relevant to the Kubernetes community. This includes end users and developers. Most of the blog's content is about things happening in the core project, but Kubernetes as a project encourages you to submit about things happening elsewhere in the ecosystem too!

Anyone can write a blog post and submit it for publication. With only a few special case exceptions, we only publish content that hasn't been submitted or published anywhere else.

Contributor blog

The Kubernetes contributor blog is aimed at an audience of people who work **on** Kubernetes more than people who work **with** Kubernetes. The Kubernetes project deliberately publishes some articles to both blogs.

Anyone can write a blog post and submit it for review.

Article updates and maintenance

The Kubernetes project does not maintain older articles for its blogs. This means that any published article more than one year old will normally **not** be eligible for issues or pull requests that ask for changes. To avoid establishing precedent, even technically correct pull requests are likely to be rejected.

However, there are exceptions like the following:

- (updates to) articles marked as evergreen
- removing or correcting articles giving advice that is now wrong and dangerous to follow
- fixes to ensure that an existing article still renders correctly

For any article that is over a year old and not marked as *evergreen*, the website automatically displays a notice that the content may be stale.

Evergreen articles

You can mark an article as evergreen by setting `evergreen: true` in the front matter.

We only mark blog articles as maintained (`evergreen: true` in front matter) if the Kubernetes project can commit to maintaining them indefinitely. Some blog articles absolutely merit this; for example, the release comms team always marks official release announcements as evergreen.

What's next

- Discover the official blogs:
  - Kubernetes blog
  - Kubernetes contributor blog
- Read about reviewing blog pull requests

- Submitting articles to Kubernetes blogs
- Blog guidelines
- Blog article mirroring
- Post-release communications
- Helping as a blog writing buddy

Submitting articles to Kubernetes blogs

There are two official Kubernetes blogs, and the CNCF has its own blog where you can cover Kubernetes too. For the main Kubernetes blog, we (the Kubernetes project) like to publish articles with different perspectives and special focuses, that have a link to Kubernetes.

With only a few special case exceptions, we only publish content that hasn't been submitted or published anywhere else.

Writing for the Kubernetes blog(s)

As an author, you have three different routes towards publication.

Recommended route

The approach the Kubernetes project recommends is: pitch your article by contacting the blog team. You can do that via Kubernetes Slack (#sig-docs-blog). For articles that you want to publish to the contributor blog only, you can also pitch directly to SIG ContribEx comms.

Unless there's a problem with your submission, the blog team / SIG ContribEx will pair you up with:

- a blog *editor*
- your *writing buddy* (another blog author)

When the team pairs you up with another author, the idea is that you both support each other by reviewing the other author's draft article. You don't need to be a subject matter expert; most of the people who read the article also won't be experts. We, the Kubernetes blog team, call the other author a writing buddy.

The editor is there to help you along the journey from draft to publication. They will either be directly able to approve your article for publication, or can arrange for the approval to happen.

Read authoring a blog article to learn more about the process.

Starting with a pull request

The second route to writing for our blogs is to start directly with a pull request in GitHub. The blog team actually don't recommend this; GitHub is quite useful for collaborating on code, but isn't an ideal fit for prose text.

It's absolutely fine to open a placeholder pull request with just an empty commit, and then work elsewhere before returning to your placeholder PR.

Similar to the recommended route, we'll try to pair you up with a writing buddy and a blog editor. They'll help you get the article ready for publication.

Post-release blog article process

The third route is for blog articles about changes in Kubernetes relating to a release. Each time there is a release, the Release Comms team takes over the blog publication schedule. People adding features to a release, or who are planning other changes that the project needs to announce, can liaise with Release Comms to get their article planned, drafted, edited, and eventually published.

Article scheduling

For the Kubernetes blog, the blog team usually schedules blog articles to publish on weekdays (Gregorian calendar, as used in the USA and other countries). When it's important to publish on a specific date that falls on a weekend, the blog team try to accommodate that.

The section on authoring a blog article explains what to do:

- initially, don't specify a date for the article
- however, do set the article as draft (put `draft: true` in the front matter)

When the Prow bot merges the PR you write, it will be a draft and won't be set to publish. A Kubernetes contributor (either you, your writing buddy or someone from the blog team) then opens a small follow-up PR that marks it for publication. Merging that second PR releases the previously-draft article so that it can automatically publish.

On the day the article is scheduled to publish, automation triggers a website build and your article becomes visible.

Authoring an article

After you've pitched, we'll encourage you to use either HackMD (a web Markdown editor) or a Google doc, to share an editable version of the article text. Your writing buddy can read your draft text and then either directly make suggestions or provide other feedback. They should also let you know if what you're drafting feedback isn't in line with the blog guidelines.

At the same time, you'll normally be **their** writing buddy and can follow our guide about supporting their work.

Initial administrative steps

You should sign the CLA if you have not yet done so. It is best to make sure you start this early on; if you are writing as part of your job, you may need to check with the workplace legal team or with your manager, to make sure that you are allowed to sign.

Initial drafting

The blog team recommends that you either use HackMD (a web Markdown editor) or a Google doc, to prepare and share an initial, live-editable version of the article text.

Note:

If you choose to use Google Docs, you can set your document into Markdown mode.

Your writing buddy can provide comments and / or feedback for your draft text and will (or should) check that it's in line with the guidelines. At the same time, you'll be their writing buddy and can follow the guide that explains how you'll be supporting their work.

Don't worry too much at this stage about getting the Markdown formatting exactly right, though.

If you have images, you can paste in a bitmap copy for early feedback. The blog team can help you (later in the process), to get illustrations ready for final publication.

Markdown for publication

Have a look at the Markdown format for existing blog posts in the website repository in GitHub.

If you're not already familiar, read contributing basics. This section of the page assumes that you don't have a local clone of your fork and that you are working within the GitHub web UI. You do need to make a remote fork of the website repository if you haven't already done so.

In the GitHub repository, click the **Create new file** button. Copy your existing content from HackMD or Google Docs, then paste it into the editor. There are more details later in the section about what goes into that file. Name the file to match the proposed title of the blog post, but don't put the date in the file name. The blog reviewers will work with you to set the final file name and the date when the article will be published.

1. When you save the file, GitHub will walk you through the pull request process.
2. Your writing buddy can review your submission and work with you on feedback and final details. A blog editor approves your pull request to merge, as a draft that is not yet scheduled.

Front matter

The Markdown file you write should use YAML-format Hugo front matter.

Here's an example:

```
---
layout: blogtitle: "Your Title Here"draft: true # will be changed to date: YYYY-MM-DD before publicationslug: lowercase-text-for-l
```

- initially, don't specify a date for the article
- however, do set the article as draft (put `draft: true` in the article front matter)

Article content

Make sure to use second-level Markdown headings (`##` not `#`) as the topmost heading level in the article. The `title` you set in the front matter becomes the first-level heading for that page.

You should follow the style guide, but with the following exceptions:

- we are OK to have authors write an article in their own writing style, so long as most readers would follow the point being made
- it is OK to use "we" in a blog article that has multiple authors, or where the article introduction clearly indicates that the author is writing on behalf of a specific group. As you'll notice from this section, although we avoid using "we" in our documentation, it's OK to make justifiable exceptions.
- we avoid using Kubernetes shortcodes for callouts (such as `{{< caution >}}`). This is because callouts are aimed at documentation readers, and blog articles aren't documentation.
- statements about the future are OK, albeit we use them with care in official announcements on behalf of Kubernetes
- code samples used in blog articles don't need to use the `{{< code_sample >}}` shortcode, and often it is better (easier to maintain) if they do not

Diagrams and illustrations

For illustrations, diagrams or charts, use the figure shortcode can be used where feasible. You should set an `alt` attribute for accessibility.

For illustrations and technical diagrams, try to use vector graphics. The blog team recommend SVG over raster (bitmap / pixel) diagram formats, and also recommend SVG rather than Mermaid (you can still capture the Mermaid source in a comment). The preference for SVG over Mermaid is because when maintainers upgrade Mermaid or make changes to diagram rendering, they may not have an easy way to contact the original blog article author to check that the changes are OK.

The diagram guide is aimed at Kubernetes documentation, not blog articles. It is still good to align with it but:

- there is no need to caption diagrams as Figure 1, Figure 2, etc.

The requirement for scalable (vector) imanges makes the process more difficult for less-familiar folks to submit articles; Kubernetes SIG Docs continues to look for ways to lower this bar. If you have ideas on how to lower the barrier, please volunteer to help out.

For other images (such as photos), the blog team strongly encourages use of `alt` attributes. It is OK to use an empty `alt` attribute if accessibility software should not mention the image at all, but this is a rare situation.

Commit messages

At the point you mark your pull request ready for review, each commit message should be a short summary of the work being done. The first commit message should make sense as an overall description of the blog post.

Examples of a good commit message:

- *Add blog post on the foo kubernetes feature*
- *blog: foobar announcement*

Examples of bad commit messages:

- *Placeholder commit for announcement about foo*
- *Add blog post*
- *asdf*
- *initial commit*
- *draft post*

Squashing

Once you think the article is ready to merge, you should squash the commits in your pull request; if you're not sure how to, it's OK to ask the blog team for help.

Helping as a blog writing buddy

There are two official Kubernetes blogs, and the CNCF has its own blog where you can cover Kubernetes too. Read contributing to Kubernetes blogs to learn about these two blogs.

When people contribute to either blog as an author, the Kubernetes project pairs up authors as *writing buddies*. This page explains how to fulfil the buddy role.

You should make sure that you have at least read an outline of article submission before you read on within this page.

Buddy responsibilities

As a writing buddy, you:

- help the blog team get articles ready to merge and to publish
- support your buddy to produce content that is good to merge
- provide a review on the article that your buddy has written

When the team pairs you up with another author, the idea is that you both support each other by reviewing the other author's draft article. Most people reading articles on the Kubernetes blog are not experts; the content should try to make sense for that audience, or at least to support non-expert readers appropriately.

The blog team are also there to help you both along the journey from draft to publication. They will either be directly able to approve your article for publication, or can arrange for the approval to happen.

Supporting the blog team

Your main responsibility here is to communicate about your capacity, availability and progress in a reasonable timeline. If many weeks go by and your buddy hasn't heard from you, it makes the overall work take more time.

Supporting your buddy

There are two parts to the process

- Collaborative editing
- Markdown / Git editing

(This is the recommended option)

The blog team recommend that the main author for the article sets up collaborative editing using either a Google Doc or HackMD (their choice). The main author then shares that document with the following people:

- Any co-authors
- You (their writing buddy)
- Ideally, with a nominated person from the blog team.

As a writing buddy, you then read the draft text and either directly make suggestions or provide feedback in a different way. The author of the blog is very commonly also **your** writing buddy in turn, so they will provide the same kind of feedback on the draft for your blog article.

Your role here is to recommend the smallest set of changes that will get the article look good for publication. If there's a diagram that really doesn't make sense, or the writing is really unclear: provide feedback. If you have a slight different of opinion about wording or punctuation, skip it. Let the article author(s) write in their own style, provided that they align to the blog guidelines.

After this is ready, the lead author will open a pull request and use Markdown to submit the article. You then provide a review.

Some authors prefer to start with collaborative editing; others like to go straight into GitHub.

Whichever route they take, your role is to provide feedback that lets the blog team provide a simple signoff and confirm that the article can merge as a draft. See submitting articles to Kubernetes blogs for what the authors need to do.

Use GitHub suggestions to point out any required changes.

Once the Markdown and other content (such as images) look right, you provide a formal review.

Pull request review

Follow the blog section of *Reviewing pull requests*.

When you think that the open blog pull request is good enough to merge, add the `/lgtm` comment to the pull request.

This indicates to the repository automation tooling (Prow) that the content "looks good to me". Prow moves things forward. The `/lgtm` command lets you add your opinion to the record whether or not you are formally a member of the Kubernetes project.

Either you or the article author(s) should let the blog team know that there is an article ready for signoff. It should already be marked as `draft: true` in the front matter, as explained in the submission guidance.

Subsequent steps

For you as a writing buddy, **there is no step four**. Once the pull request is good to merge, the blog team (or, for the contributor site, the contributor comms team) take things from there. It's possible that you'll need to return to an earlier step based on feedback, but you can usually expect that your work as a buddy is done.

Blog article mirroring

There are two official Kubernetes blogs, and the CNCF has its own blog where you can cover Kubernetes too. For the main Kubernetes blog, we (the Kubernetes project) like to publish articles with different perspectives and special focuses, that have a link to Kubernetes.

Some articles appear on both blogs: there is a primary version of the article, and a *mirror article* on the other blog.

This page describes the criteria for mirroring, the motivation for mirroring, and explains what you should do to ensure that an article publishes to both blogs.

Before you begin

Make sure you are familiar with the introduction sections of contributing to Kubernetes blogs, not just to learn about the two official blogs and the differences between them, but also to get an overview of the process.

Why we mirror

Mirroring is nearly always from the contributor blog to the main blog. The project does this for articles that are about the contributor community, or a part of it, but are also relevant to the wider set of readers for Kubernetes' main blog.

As an author (or reviewer), consider the target audience and whether the blog post is appropriate for the main blog. For example: if the target audience are Kubernetes contributors only, then the contributor blog. may be more appropriate; if the blog post is about open source in general then it may be more suitable on another site outside the Kubernetes project.

This consideration about target audience applies to original and mirrored articles equally.

The Kubernetes project is willing to mirror any blog article that was published to https://kubernetes.dev/blog/ (the contributor blog), provided that all of the following criteria are met:

- the mirrored article has the same publication date as the original (it should have the same publication time too, but you can also set a time stamp up to 12 hours later for special cases)
- For PRs that add a mirrored article to the main blog *after* the original article has merged into the contributor blog, ensure that all of the following criteria are met:
  - No articles were published to the main blog after the original article was published to the contributor blog.
  - There are no main blog articles scheduled for publication between the publication time of the original article and the publication time of your mirrored article.

  This is because the Kubernetes project doesn't want to add articles to people's feeds, such as RSS, except at the very end of their feed.
- the original article doesn't contravene any strongly recommended review guidelines or community norms
- the mirrored article will have `canonicalUrl` set correctly in its front matter
- the audience for the original article would find it relevant
- the article content is not off-topic for the target blog where the mirror article would appear

Mirroring from the main blog to the contributor blog is rare, but could feasibly happen.

How to mirror

You make a PR against the other Git repository (usually, https://github.com/kubernetes/website) that adds the article. You do this *before* the articles merge.

As the article author, you should set the canonical URL for the mirrored article, to the URL of the original article (you can use a preview to predict the URL and fill this in ahead of actual publication). Use the `canonicalUrl` field in front matter for this.

Post-release communications

The Kubernetes *Release Comms* team (part of SIG Release) looks after release announcements, which go onto the main project blog.

After each release, the Release Comms team take over the main blog for a period and publish a series of additional articles to explain or announce changes related to that release. These additional articles are termed *post-release comms*.

Opting in to post-release comms

During a release cycle, as a contributor, you can opt in to post-release comms about an upcoming change to Kubernetes.

To opt in you open a draft, *placeholder* pull request (PR) against k/website. Initially, this can be an empty commit. Mention the KEP issue or other Kubernetes improvement issue in the description of your placeholder PR.

When you open the **draft** pull request, you open it against *main* as the base branch and not against the `dev-1.35` branch. This is different from the process for upcoming release changes and new features.

You should also leave a comment on the related kubernetes/enhancements issue with a link to the PR to notify the Release Comms team managing this release. Your comment helps the team see that the change needs announcing and that your SIG has opted in.

As well as the above, you should ideally contact the Release Comms team via Slack (channel `#release-comms`) to let them know that you have done this and would like to opt in.

Preparing the article content

You should follow the usual article submission process to turn your placeholder PR into something ready for review. However, for post-release comms, you may not have a *writing buddy*; instead, the Release Comms team may assign a member of the team to help guide what you're doing.

You should squash the commits in your pull request; if you're not sure how to, it's absolutely OK to ask Release Comms or the blog team for help.

Provided that your article is flagged as a draft (`draft: true`) in the front matter, the PR can merge at any time during the release cycle.

Publication

Ahead of the actual release, the Release Comms team check what content is ready (if it's not ready by the deadline, and you didn't get an exception, then the announcement won't be included). They build a schedule for the articles that will go out and open new pull requests to turn those articles from draft to published.

Caution:

All these pull requests to actually publish post-release articles **must** be held (Prow command `/hold`) until the release has actually happened.

The blog team approvers still provide final sign off on promoting the content from draft to accepted for publication. Ahead of release day, the PR (or PRs) for publishing these announcements should have LGTM ("looks good to me") and approved labels, along with the **do-not-merge/hold** label to ensure the PR doesn't merge too early.

Release Comms / the Release Team can then *unhold* that PR (or set of PRs) as soon as the website Git repository is unfrozen after the actual release.

On the day each article is scheduled to publish, automation triggers a website build and that article becomes visible.