Gateway API Releases

updated 2025-10-03

There are, as the saying goes, two kinds of software in the world: software that ships, and software that is pointless.

Gateway API is, thankfully, the first kind of software. But of course, it's not sufficient just to release: we want *high-quality*, *timely*, *predictable*, *low-stress* releases:

- *High-quality*: no project can afford to break their production users.
- *Timely*: we want releases to show up every quarter. This doesn't just reflect well on the health of the project – we generally have to release in order for real users to try out new things and give us feedback, so releasing frequently shortens our development cycle and lets us more rapidly converge on the things that everyone wants.
- *Predictable*: Gateway API is used by end users, but *also* by downstream projects that need to add support for new Gateway API features in their codebases. We don't want to surprise the projects that help us succeed.
- *Low-stress*: as a community-driven project, we rely on people donating their time and energy for the good of Gateway API, so we can't afford to burn people out.

These are often at odds with each other, especially when we consider Kubernetes' requirements around changes and CRD versioning! So, after looking at these four things and at the Gateway API 1.4 release feedback, we're proposing changes for the next release.

Provisional GEPs

Instead of the beginning of a release cycle being voting on ideas that will then have GEPs written, we're going to start with ideas for which a Provisional GEP has already been accepted. If you have an idea for an amazing thing, you need to write the GEP that says **what** the idea is, **who** the idea benefits, and **why** we should include it Gateway API, and you need to get that GEP merged as Provisional before it can be considered for Experimental.

You'll also need to get three implementations (or two, if it's a mesh feature) to agree that they'll actually support it if it's accepted into Experimental, which is likely to require talking with people! and, of course, may mean that you'll need to do some refinement of your Provisional GEP. This may seem frustrating at first, but it's likely to be an overall win to know that things coming into experimental have already had enough thought put into them that implementations are already engaged with them.

We also want to be very clear that Provisional GEPs are **not** limited to progressing during the release cycle! GEPs can merge to Provisional at any time, and to consider a GEP for a given release, it needs to be merged as Provisional before the start of that release.

For this next release, we will start by seeing how many proposals have been accepted into Provisional and have sponsors. If there are more than the current number of open Experimental slots (what a lovely problem to have!) either the maintainers will agree to raise the number of Experimental slots, or we will vote as a community on which to accept into Experimental.

Release Trains

So what happens if a GEP isn't merged before the start of a release? For that matter, what happens if we're halfway through implementation and we realize that a given feature has run into trouble?

To meet the four goals we listed above, it's really critical that we be able to drop things from a given release in order to allow the release to proceed in a predictable way. This implies that **we will set the date of a release, but allow the content to be flexible**: as long as we have stuff to release that improves the API (which we always should!), we'll drop things instead of delaying the release.

This also implies that it's possible to get to a release date and have only bugfixes instead of features. We should absolutely ship in that case - bugfixes improve the API! - but in deference to SemVer, we'd release it as a patch rather than a minor bump.

Monthly Experimental Releases

The biggest change, though, is that we're going to do experimental-channel releases **monthly**, so that we have more chances to get feedback and more opportunity to iterate quickly to get things to standard. The goal here is to make it possible to get new features into *and out of* Experimental more quickly: if we make new experimental features available for user feedback every month, we should be able to move them through the process more quickly.

To make sure that the Experimental channel doesn't get bogged down with things that aren't progressing, we will automatically remove GEPs from Experimental six months after they enter Experimental state. Any meaningful progress forward to Standard will reset the timer. (Note that this doesn't imply that there will be no communication about proposals not advanced before they're dropped! Such proposals would, at minimum, be discussed in the weekly calls, and "meaningful progress" can come from anyone – e.g. if

the GEP author isn't editing the GEP but implementations are committing conformance tests, that's definitely meaningful progress!)

Note that this timer also applies to existing Experimental items from the v1.4.0 timeframe – if e.g. the Mesh resource makes no meaningful progress in six months, it'll get dropped.

These monthly releases are going to be very different from standard-channel releases:

Monthlies Are Snapshots

A monthly experimental release will simply be a snapshot of `main`. This implies that monthlies *never* get bugfixes backported, and also that `main` needs to be kept in a shippable state.

To emphasize the bugfix point: standard-channel releases will always have release branches, so that if we find a bug affecting a standard-channel release, we can backport the fix and do a patch release. Monthlies will **never** have a release branch, so that level of fix is not possible: bugfixes in monthlies get delivered in later monthlies, which may have other features as well.

Monthlies Don't Use SemVer

Since monthlies are just snapshots, they'll be named `monthly-$year-$month`, e.g. `monthly-2025-10`. **This is not a semantic version**; breaking changes are permitted between any two monthlies (though they should be done with care).

New Experimental Resources Still Use `x-k8s.io` and X-Names

For example, XMesh does not change its name, nor its API group – but neither do TCPRoute and UDPRoute. Those two are experimental, but they're not *new*, so we're not changing them.

Monthlies Are Still the Experimental Channel's Content

Like our current experimental releases, monthlies include

- Resources that have graduated to GA API versions, including their non-experimental fields;
- Resources that only have alpha API versions, including X- resources; and
- All experimental fields in all resources.

The point here is that you will get a functioning Gateway API installation if you just e.g. `kubectl apply monthly-2025-10-install.yaml`; you will *not* need to apply a standard-channel YAML file and then a monthly YAML file.

Making Monthlies Safer

Monthlies will inherit the long-standing challenge that it's hard to make sure that people using the experimental channel are aware of which features are experimental and which are not. Knowing that a resource named "X-something" is experimental is one thing; knowing which of the many fields in a v1 HTTPRoute are experimental is very different, though. For example, if we add a `MoonPhase` stanza to HTTPRoute, many users may simply see "oh, HTTPRoute is v1, all good!" and not read the fine print about `MoonPhase`. We **do not** want people believing that experimental fields are actually standard, though.

However, the biggest challenge with monthly experimental releases is that they will offer *more* chances to break things, since the whole point of the monthlies is to let us move faster. For example, suppose that `monthly-2025-10` has a `MoonPhase` that's an `int`, but `monthly-2025-11` changes it to a `string`. That's allowed by the experimental channel rules, but if you've installed `monthly-2025-10` and you then apply `monthly-2025-11` on top of it, the API server will not be happy about that — and it may be unhappy in unpredictable ways.

In a perfect world, we would deal with this by moving the experimental channel into a separate API group, because that's the only way to truly isolate the channels. In the real world, though, this would place too much of a burden on implementations: the Kubernetes API clients don't have any support for this kind of channelized development, which would require the implementations to duplicate a lot of code and then maintain the duplications.

So, instead, we'll lean on validating admission policies (VAP). This feature became GA in Kubernetes 1.30, so it's probably OK to rely on them for the experimental channel at this point.

VAP for Upgrades

Every standard-channel release will include a VAP named `gateway-api-safe-upgrades` that will prohibit the following:

- Installation of experimental CRDs on top of standard channel CRDs (within the same API group)
- Installation of monthly releases

- Installation of older releases

Chihiro and Ian will have sufficient access to uninstall this VAP and then do whatever they want (though it will be reintroduced the next time they install a standard-channel release). Ana will all but certainly not have sufficient access to uninstall it; that's appropriate.

The VAP for upgrades will *not* block installing newer standard channel CRDs - that can continue to be a low-friction process because of our compatibility guarantees.

VAP for Experimental Fields in GA Resources

All releases of Gateway API - including monthlies! - will include a VAP named `gateway-api-guardrails` that will prohibit setting experimental fields in resources that are part of standard channel unless they have added the following annotation to the instance of the resource:

```
None
gateway.networking.k8s.io/unsafe-enable-experimental: reason
```

where `reason` is an arbitrary string supplied by the person creating the resource (for example, a project name, so that they could easily find all the resources for that project). Again, Chihiro and Ian will have sufficient access to uninstall this VAP if they want to skip the annotation, but Ana will not. The VAP will be reintroduced on upgrades.

The Default Stance: Reinstall

Note that the guardrail VAPs *don't* - and can't - address the case where `MoonPhase` changes types between monthlies. There's really no good way to tackle this, so if you are developing using monthlies, your default expectation should be that upgrading to a new monthly may require you to delete all your CRDs and reinstall. We'll try to minimize that, but the assumption is that monthlies are used in *development*, and in development, it shouldn't be a big deal to delete and reinstall.

The Developer Experience

The combination here allows for some reasonable experiences.

- If you install the standard channel, you won't see anything experimental at all, of course.

- If you install a monthly, the installed CRDs will include all experimental fields and experimental resources, but the experimental functionality will not be available to you unless you start annotating resources with `gateway.networking.k8s.io/unsafe-enable-experimental`. Without the annotation, you effectively have only the standard-channel functionality.
- Implementations SHOULD default to recommending that users of production releases of their products should install standard channel Gateway API CRDs.
- Implementations SHOULD build production releases of their products against standard channel modules. If an implementation builds against experimental modules, that should still work, since
    - we're not allowed to change standard-channel elements in experimental, and
    - users of these implementations' production releases really should be installing standard channel CRDs, not experimental.
- An implementation working on support for a new experimental feature can develop against experimental and, again, if the end user doesn't annotate for experimental fields, they'll be harmless.

Timing

Our first planned monthly release is `monthly-2025-10` on 27 October 2025. This is ambitious: monthlies need to be automated and there's a lot to be done.

Our next standard channel release - which, remember, has no number just yet - is planned for 26 January 2026. Until we get closer, I'm going to refer to this release as Osaka.