


Istio APIs Release Channels

Shared with Istio Community

	
Owner: Whitney Griffith (whitneygriffith) Working Group: Test and Release, User Experience	Status: WIP In Review Approved Obsolete Created: Nov 16, 2023 Approvers: howardjohn[], mitchconnors[], ericvn[], linsun[], louiscryan[], Release Target: 1.22

Objective

To transition to a new, simplified release model for Istio APIs that allows the flexibility of experimenting with new features, while still delivering a stable API.

Background

Key Motivations

1. Features within an Istio API exhibit varying levels of stability, leading to an independent feature lifecycle distinct from that of the API version.
2. Istio APIs implemented as Kubernetes CRDs encounter a [known challenge](#) for converting between API versions that have different schemas. Istio has chosen to overcome this by enforcing [identical](#) schemas for all versions of the same CRD. This created a new problem, where deprecated or unstable features are now present in all CRD versions without any feature gating. In other words, an API version that indicates stability is in fact misleading, as it contains unstable features that a user can apply at will.
3. Simplifying Istio APIs' Feature Phases will incentivize developers to accelerate conforming their features to stability standards while minimizing the [Costs of API Versions](#). At this time, in Istio we have Experimental, Alpha, Beta and Stable phases that require substantial operational oversight to manage the progression or removal of features within Istio APIs. The operational friction includes maintaining and enforcing at least four graduation criterias, API versions, etc. to inform users whether an API is unstable vs stable. Historically, we now have APIs in varying degrees of stability that are stickily adopted by users due to necessity vs stability concerns.

Proposal

Release Channels

There will be an [Extended](#) and a [Stable](#) Channel that will be used to deliver the most current Istio APIs and features.

The [Extended Channel](#) will be Istio as we know it today, where all existing APIs and API features will be available regardless of stability, the superset.

The [Stable Channel](#) will deliver only the stable API and API features, the subset.

In this way, Release Channels allow us to add new fields and resources to Istio APIs while still providing stable APIs and features through the [Stable Channel](#).

Stable Channel

- **Stability Assurance:** The Stable Channel is curated to include only the parts of Istio that have reached a stable, generally available (GA) state. This ensures reliability and consistency for production environments.
 - The APIs are stable as all features are stable.
- **GA Features Only:** Users selecting the Stable Channel can expect to access features that have undergone thorough testing and validation, and are deemed suitable for use in mission-critical applications.
 - A Stable Channel API has **v1** [API versions](#) served and only contains GA features. In the Stable Channel non-GA features of an API will not be available for use.
 - A non-GA API is graduated to the Stable Channel when there is a **v1** version of the API which represents a stable core of features.
- **Risk Mitigation:** Opting for the Stable Channel minimizes the risk of encountering unexpected behavior or breaking changes, providing a dependable foundation for deploying Istio in production environments.
 - Only backwards-compatible changes allowed
 - If Istio decides to take the API in a new direction with incompatible changes to the existing **v1** API, we strongly encourage creating a new CRD with a new name, etc.
 - Any backwards-compatible changes will be accessible when upgrading Istio and will be noted in the changelog for the new Istio release.
 - New fields will be added to the API with the `releaseChannel:extended` annotation usable only in the [Extended Channel](#). New fields will undergo its independent graduation [lifecycle](#) until it is determined to be stable, **v1**.

Extended Channel

- **Functionality Focus:** The Extended Channel encompasses the entirety of Istio's features, regardless of their current stage of development or stability.
- **Inclusive of Extended Features:** Users opting for the Extended Channel gain access to all features offered by Istio, including those in the non-GA stage. This allows for early adoption and testing of cutting-edge functionalities.
 - Breaking changes allowed for non-GA features and APIs that have not been graduated to **v1**.
 - Extended features can have major changes that will lead to data loss if not managed before upgrading
 - These changes can be:
 - Deprecating required and non-required extended fields
 - Changing the data type of extended fields
 - Changing validation rules of extended fields
 - Renaming extended fields
 - Changing default values of extended fields
 - Changing field semantics where the meaning or the purpose of an extended field is changed.
 - Changes to an existing **v1** field is not permissible
- **Varied Stability Levels:** Features in the Extended Channel may range from non-GA to stable, providing a broad spectrum of capabilities to users who are willing to explore and experiment with the latest advancements.
 - Non-GA fields in **v1** Kinds will be marked with the `releaseChannel:extended` annotation usable only in the Extended Channel.
 - Non-GA APIs will be marked with the `releaseChannel:extended` annotation usable only in the Extended Channel.
 - Messaging to users
 - The Extended channel is only for those users who are willing to actively shepherd their Istio deployments. You will potentially need to adjust your Istio feature usage *every upgrade* to comply with the latest version of these resources. This complexity will only increase as more clusters/workloads are added to the mesh. Thread carefully.
 - It will be relatively easy to go from Stable Channel to Extended Channel and relatively harder to move from Extended Channel to Stable Channel as Extended Channel is the superset.

API Management

API Versions

Each API version provides a unique way to interact with the API. For example, a user can create a Custom Resource based on the **v1alpha1** or a **v1** version of the API.

Moving forward Istio API versions will ideally only be **v1alpha1** and **v1**.

- New users will be guided to only use the **v1** and **v1alpha1** versions.

Existing users will not be forced to migrate until we remove the non **v1alpha1** and **v1** versions they are currently using. Removal will follow the deprecation policy for the respective API version's [feature phase](#) and be directed based on our revision support requirements. At that point, migration will involve [upgrading existing objects to a new stored version](#) and K8s will have first class support to do this in k8s [1.30](#).

Existing APIs

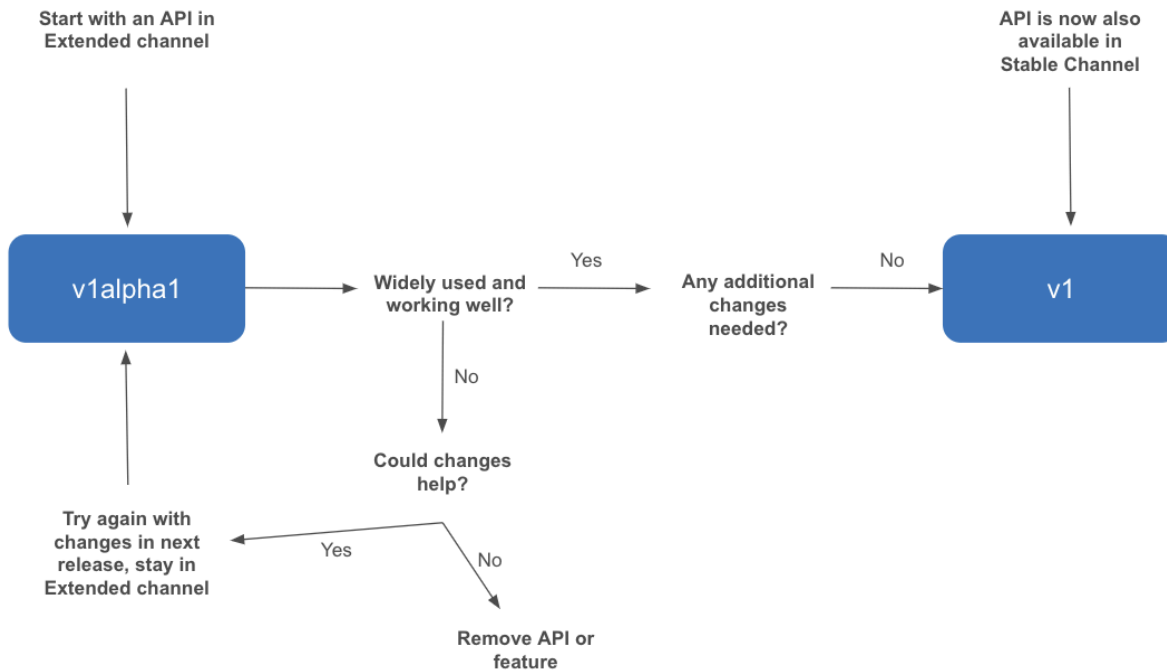
The below table reflects the Release Channels, existing Istio APIs will be placed in.

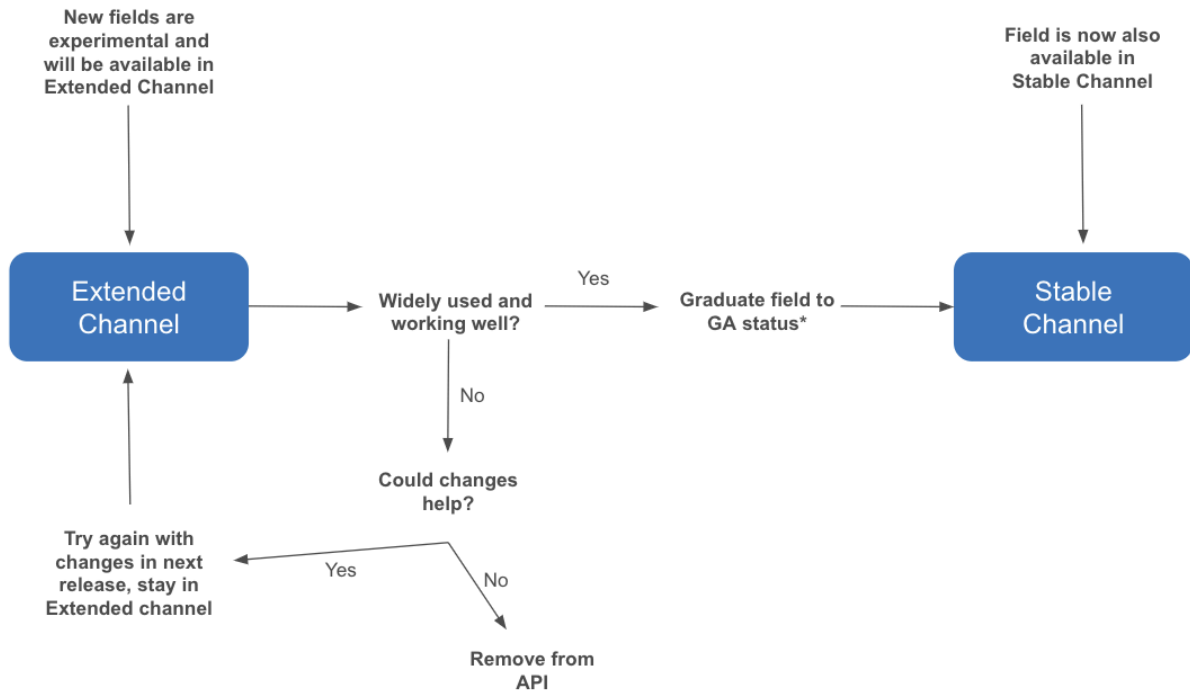
APIs	Stable Channel	Extended Channel
AuthorizationPolicy	✓	✓
DestinationRule	✓	✓
EnvoyFilter		✓
Gateway	✓	✓
PeerAuthentication	✓	✓
ProxyConfig		✓
RequestAuthentication	✓	✓
ServiceEntry	✓	✓
Sidecar	✓	✓
Telemetry API	✓ A subset of the v1alpha1 Telemetry API will be added to Stable Channel as proposed here	✓

VirtualService	✓	✓
WasmPlugin		✓
WorkloadEntry	✓	✓
WorkloadGroup	✓	✓

API Lifecycle

The criteria for graduation and removal of features and APIs will follow our official Istio Graduation Policy defined [here](#). We are also proposing additional [graduation](#) and [deprecation](#) criteria for APIs. The ability to limit the length of time an API or feature stays in the [Extended Channel](#) will be dependent on Istio's overall [Enhancement Strategy](#) to drive up and out momentum of features.





*This will be done by removing the extended annotation on the field

New APIs & Fields

1. New Istio resources will be created as a **v1alpha1** API in the [Extended Channel](#).
2. Once there is a stable core of features in the **v1alpha1** API, a **v1** API version will be created according to the [Graduation criteria](#) and will become available in the [Stable Channel](#).
3. Both the **v1alpha1** and **v1** version of the API will continue to exist until TBD.
4. If new, non-GA features are being introduced to the API, a `releaseChannel:extended` annotation will be added to the field in the **v1** and **v1alpha1** API versions. The CRD generation tooling will be modified such that when generating the CRDs for the [Stable Channel](#), any fields with this annotation will be excluded, but when building [Extended Channel](#) CRDs these non-GA fields will be included in addition to all stable fields for a **v1** resource.
5. When a non-GA field is determined to be stable according to the [Graduation criteria](#), the `releaseChannel:extended` annotation will be removed and the field will be available in the [Stable Channel](#).
6. If a non-GA resource or field is determined to be unsuitable for promotion, it will be [deprecated](#) accordingly and eventually removed in a future release.

Implementation

Tracking Issue: <https://github.com/istio/enhancements/issues/173>

Validating Admission Policy

Kubernetes has a [Validating Admission Policy](#) (Beta) that offers a declarative, in-process alternative to [Validating Admission Webhooks](#).

[Validating Admission Policies](#) use the Common Expression Language (CEL) to declare the validation rules of a policy. Validation admission policies are highly configurable, enabling policy authors to define policies that can be parameterized and scoped to resources or features as needed by cluster administrators. Configurations of the API that violate the [Validating Admission Policy](#) causes the API request to fail.

As such, the Release Channels logic will be implemented as a [Validating Admission Policy](#), where the sole use of stable APIs and features in the [Stable Channel](#) will be enforced using a [Validating Admission Policy](#).

Istio will provide a [Validating Admission Policy](#) to be used for the Stable Channel based on the [stability](#) of each API and API Feature. Vendors and users are able to further customize the Stable Channel [Validating Admission Policy](#) or add additional [Validating Admission Policy](#) for their feature scoping needs.

1. Defined Admission Policy Rules:
 - a. Validating Admission Policy rules are created to enforce the constraints specified for the [Stable Channel](#) based on stability. That is, the policy will ensure only stable APIs and features are used.
2. Admission Controller Configuration:
 - a. Users opt in to the Stable Channel when installing or upgrading Istio and in so doing, the Kubernetes Admission Controller is configured to use the [Validating Admission Policy](#).
 - b. The [Validating Admission Policy](#) is compatible with revision upgrades in Istio
3. Rule Evaluation:
 - a. When a resource creation or update request is made to the Kubernetes API server, the Admission Controller intercepts the request and evaluates it against the Stable Channel [Validating Admission Policy](#) rules.
 - b. Requests will be allowed or rejected based on the result of the evaluation.
 - c. If the request is rejected, users will be provided clear feedback about why their requests were rejected.

The first iteration of the Stable Channel [Validating Admission Policy](#) will be manually configured with the aim of automating configuring the policy rules based on changes to the APIs.

Decisions

v1beta1 APIs

1. Deprecate the **v1alphaX** versions and update Istiod to read the **v1beta1** resource
2. Add **v1beta1** resource to the [Extended Channel](#) as is
3. Create a **v1** version of the resource in the [Extended Channel](#) for all APIs but ProxyConfig
4. Deprecate **v1beta1** resource feature
 - a. Based on our [Beta Deprecation Policy](#), the **v1beta1** version can be removed with 3 months of advanced notice.
 - b. ProxyConfig deprecation plan will also contain revisiting the experimenting ProxyConfig annotations and providing an alternative based on user feedback

v1alpha1 version in Stable

Should we have **v1alpha1** API versions in the [Stable Channel](#)?

Consensus: Yes

Considerations	Alternatives
Reduces breakage when upgrading Istio and Switching between release channels	Users have to migrate the old CRs before upgrading. https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definition-versioning/#previous-storage-versions
Serving an Alpha version in Stable Channel feels weird even if extended features are stripped. However, we can overcome this by ensuring our Users are aware of our API versioning constructs.	Drop the Alpha version and mitigate breakage when upgrading Istio or switching release channels

Removal of v1alpha1

Once an extended resource has been graduated to the [Stable Channel](#), after several releases, the extended **v1alpha1** API version could be removed. If there are future extended fields, those will be added to the **v1** version of the resource, which will only be available in the [Extended Channel](#).

Consensus: No, we should not remove **v1alpha1** API

Considerations	Alternatives
<p>If v1alpha1 is removed, users that have not migrated to v1 will be broken.</p> <p>The ease of migrating to v1 may not be worth the removal.</p> <p>https://github.com/istio/istio/pull/49583#pullrequestreview-1904399360</p>	<p>Migration will involve upgrading existing objects to a new stored version and K8s will have first class support to do this in k8s 1.30.</p> <p>When there's first class support for migration, we can reconsider the value/effort tradeoff.</p>

Default Channel

The optimistic transition of channel defaults will be [Extended Channel](#) -> [Stable Channel](#).

Implementation Choice

[Validating Admission Policy](#) is preferred over the [Multiple CRDs](#) approach because:

1. [Validating Admission Policy](#) implements Release Channels as a layer on top of existing Istio, requiring minimal changes to Istio and the overall User Flow for installation and upgrades.
2. Revision based upgrades are compatible with [Validating Admission Policy](#).
3. [Validating Admission Policy](#) is easier for users and vendors to customize while still keeping Istio API versions the same.

Transition

As we transition to this model we will ensure the following:

- The API versions and guarantees will stay valid and be honored for all existing APIs.
- The old API framework (APIs, implementation) will continue to exist alongside the new model for the foreseeable future. The old API framework will be considered the [Extended Channel](#) and this will be the default channel for the foreseeable future. After which, we can pursue making [Stable Channel](#) the default channel, where users are consciously opting into a more extended, robust Istio.
- **v1beta1+** APIs will be added to the [Stable Channel](#), and work will be done to remove the **v1beta1** API versions so we can eventually have only **v1** APIs available throughout Istio. See decision [here](#).
- Additional work will need to be done to [clean up Existing APIs](#) to get to our optimal state of only **v1alpha1** and **v1** API versions.
- New API and API Feature development should be done using the [API Lifecycle Guidelines](#).

Additional thoughts:

- Will need to integrate with other Istio versioning work.

Roadmap

1. Adopt [Release Channels](#) for Istio APIs
2. Update Istio Feature Phases [Graduation Criteria](#)
3. Graduate Telemetry API to Stable Channel based on existing Graduation Criteria
4. Integrate with other Istio versioning work
 - a. Rolling out the Stable vs Extended concept to all Istio Features
5. Graduate Common APIs between Ambient and Classic
6. Improve Release Channels based on Feedback from Release Managers, Users, Etc.

Addendum

Graduation Criteria

This is the proposed additions to be made to the official [Istio Graduation Policy](#) for APIs specifically.

For an API to graduate to [Stable Channel](#), it must meet the following criteria:

- The core of the API is stable
- Full conformance and integration test coverage.
 - Integration tests cover edge cases as well as common use cases. Integration tests cover all issues reported on the feature. The feature has end-to-end tests covering the samples/tutorials for the feature.
 - We do not track code coverage atm, but can include a baseline when Istio adopts code coverage tracking.
- At least **two** release cycles in the [Extended Channel](#).
- No major changes (i.e. no design or behavioral changes) for at least **two** release cycles.
- Approval from the working group leads + reviewers.
- Approved Enhancements [Feature Checklist](#)

For a field or feature to graduate from Extended to Stable, it must meet the following criteria:

- Full conformance and integration test coverage.
- At least **two** release cycles in the Extended Channel.
- No major changes (i.e. no design or behavioral changes) for at least **two** release cycles.
- Approval from the working group leads + reviewers.
- Approved Enhancements [Feature Checklist](#)
- ☐ Runtime Compatibility

After a feature has spent at least **two** release cycles in the [Extended Channel](#):

- The Feature owner is required to evaluate if there are any changes that can help stabilize the feature or increase adoption.
 - An issue will be created to re-evaluate Feature
 - Feature owners will be assigned to issue and be pinged in relevant slack channel
 - Discussion items will be highlighted in combined working group meeting
- If there aren't any changes that can help or interest to make those changes within **four** additional release cycles, the [deprecation process](#) will be triggered for the extended feature. This is contingent on Istio being able to oversee and maintain up and out momentum of features
- If there are changes to be made, the Feature Owner will create issues and update the [Feature Checklist](#) to track these specific requirements for graduating the feature to Stable.
- After all requirements are completed, the Feature Owner will be able to mark this feature as ready to be graduated.
- If there are no major changes (i.e. no design or behavioral changes) in the next **two** release cycles, after being marked for graduation, the feature will be graduated to the [Stable Channel](#).
- If there are major changes during the next **two** release cycles, after being marked for graduation, the [Feature Checklist](#) should be updated, and a new target graduation date of the next **two** release cycles is set based on the completion of work.

The above can be streamlined using tools (Github and Slack integrations) and included as part of the Release Manager duties.

Deprecation Criteria

This is the proposed additions to be made to the official [Istio Deprecation Policy](#) for APIs specifically.

For a field or feature to be deprecated from the [Extended Channel](#), **v1alpha1** API, it must meet the following criteria:

- Documentation around any potential data loss risks during version conversion is captured. This can be guarded by testing conversion between the current **v1alpha1** API and the new **v1alpha1** API in a controlled environment.
- Sharing additional guidance on identifying and addressing any issues proactively is highly encouraged but optional.

Costs of API Versions

Each additional API version we support comes with increased costs to API maintainers, implementers, and users.

Istio API Maintainers:

- Maintain separate type definitions and generated code for each API version
- Deprecate and eventual remove stale features and unstable API versions
- Provide explicit upgrade requirements

Istiod Maintainers

- Handle breaking changes on upgrades
- Test all supported API versions
- Deprecate and remove support for stale features and unstable API versions
- Store the latest API version in etcd
- [TBD] Support multiple API versions based on what version is installed
 - Istiod currently only reads the oldest CRD version
 - This is a change from how Istiod operates today

Users

- Upgrade all manifests to use latest API versions
- Specify the API version to store in etcd
- Specify the API versions accepted

CRD Versioning in Kubernetes

Personas

Istio maintainers

Want to clearly indicate both API stability expectations and direction of future investment or maintenance-only/deprecation tracks for vendors, platform teams and end users. Existing usage of “[feature phase](#)” definitions hasn’t been sufficient to accomplish these goals, in part from a lack of “up or out” momentum with widely-adopted APIs languishing as v1beta1 CRDs, but also from constraints of an inflexible CRD versioning model - because Istio (like many projects using CRDs) has opted to not use a conversion webhook, [all versions must be identical](#).

Vendors

Sell products built on top of the open source Istio project, which may include additional functionality or provide a managed offering. May alter implementation details (such as a managed control plane or custom Ambient waypoint implementation), but generally aim to adhere to public APIs consistent with the upstream open source project. May want to block some features their operations or support teams don’t feel comfortable supporting or enable experimental features their users are demanding.

Platform teams

Internal team in an organization which owns core services and may offer “service mesh as a service” to internal application development teams to run and connect their services. May build abstractions over standard Istio APIs to ease adoption for developer teams or enforce best practices. May want to offer a more prescriptive “guided path” with only a limited set of functionality exposed to app dev teams, but still want direct control over more advanced functionality and need to understand anticipated support and stability to know what to expose to their customers.

End users

Confused by arbitrary vendor decisions around supported APIs and want to easily understand how to accomplish common tasks with Istio. May feel confused by having too many decision points when onboarding (Gateway API vs Istio APIs? Sidecar vs Ambient?) and don’t understand why configuration snippets from Stack Overflow don’t work as expected with their specific installation.

Background and possible solution space

Similar tradeoffs as brought up in <https://github.com/cilium/cilium/issues/29676> between user expectations and vendor needs. Starting point for conformance? Is that desirable/necessary?

- Field-level stability
 - Istio expects to already have a need for field-level granularity in the future for Gateway API extended conformance fields we may opt to not support.
 - Controller decides, report in status likely more aligned with expectations than admission controller
 - Costin “Btw - if user has multiple revisions of istiod, one for select workloads using extended features and one default on stable
 - Which is a good approach for supportability
 - Meaning only a small known set of workloads can use experimental
 - That doesn't work with either crd version or admission
- “v1 is forever”
 - “I don't think versions solve this. One way I saw it phrased was basically “Kubernetes versions solve the problem of representing *the same information* in a different syntax, not representing different information”
- Approaching this from a different point in time than Gateway API
 - Gateway API: “everything experimental, start graduating subset”
 - Has considered possibility of adding a [third, “stable” release channel](#) at some future point as API evolves if need arises
 - Istio: widely-adopted “effectively stable” API, recognized patterns for stability of new resources and changes, want to reflect status quo end user expectations of stability
 - May want an *additional* more-experimental channel in the future
- WASM truly experimental, EnvoyFilter inherently unstable
- Annotations and mesh config not included
- “too experimental to get user usage” has felt like less of an issue in Gateway API, where vendors are building and offerings features which are still experimental and users asking for more functionality - Kubernetes core is largely held back by that decision being in the hands of distributors and not end users

Storage Version

Read more about versions in CRDs:

<https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions-versioning/>

In Kubernetes Custom Resource Definitions (CRDs), the `storage` field within the `versions` block indicates whether a particular version of the custom resource is stored in the cluster's etcd database.

Setting `storage: true` means that instances of this version of the custom resource are persisted to etcd. This allows Kubernetes to store and manage resources of this version in a manner similar to built-in resource types like Pods or Deployments.

When you set `storage: true`, Kubernetes ensures that the API server can store, retrieve, and manage instances of this version of the custom resource. This enables operations such as creating, updating, deleting, and querying resources of this version through the Kubernetes API.

Conversely, if you set `storage: false`, instances of this version of the custom resource are not stored in etcd. This might be useful in scenarios where you have a version of the resource that is only meant for serving data to clients via the Kubernetes API but does not need to be persisted in the cluster's data store.

In summary, setting `storage: true` allows instances of the specified version of the custom resource to be stored in the cluster's etcd database, enabling Kubernetes to manage them like built-in resource types.

When a Custom Resource Definition (CRD) in Kubernetes has multiple versions, with only one version marked as `storage: true`, and a resource of a different version is created, Kubernetes handles it differently depending on the configuration and behavior of the API server.

Here's what typically happens:

1. **Resource Creation**: When a resource of a non-storage version is created, the Kubernetes API server will still accept and validate the request, assuming the request adheres to the schema of the non-storage version.
2. **Storage**: Despite the resource being of a non-storage version, the API server still writes the resource's data to etcd. However, it may not be fully indexed or searchable in the same way as resources of the storage version.
3. **Compatibility**: The API server usually tries to maintain compatibility by storing the data in a way that allows it to serve requests for both storage and non-storage versions. However, the behavior may vary depending on the specifics of the Kubernetes version and configuration.
4. **Limitations**: While the resource may be stored, it might not have the same level of management and indexing as resources of the storage version. For instance, it may not be subject to the same consistency guarantees, or it may not be included in certain types of queries or operations.
5. **Considerations**: It's essential to carefully consider the implications of having multiple versions with different storage configurations. Mixing storage and non-storage versions can lead to complexities in managing and querying resources, and it may impact performance or consistency guarantees.

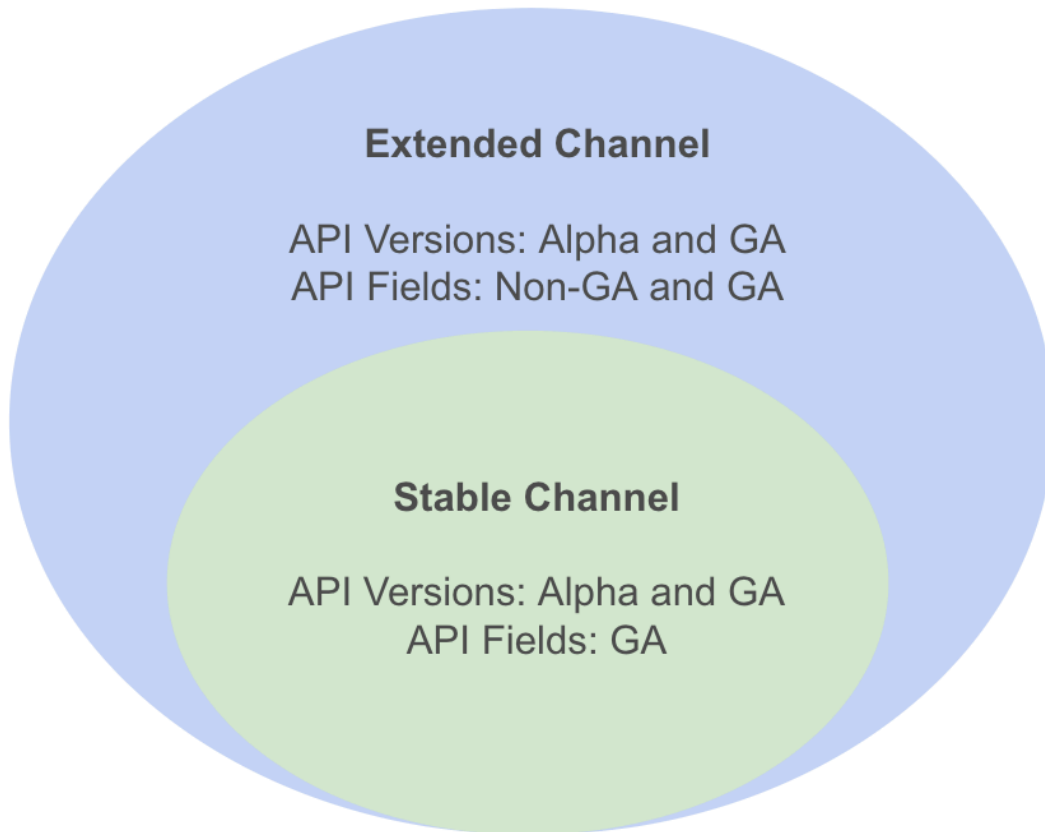
In summary, when a custom resource version isn't the storage version, the Kubernetes API server still stores the resource's data in etcd, but the behavior and management of that data may differ from resources of the storage version.

Design Considerations

1. Extended features can only be promoted to Stable when new Stable API versions are released. We may need to be prescriptive on the criteria and timeline for new Stable API versions to facilitate the reasonable batching of the extended features to be promoted.
2. End users might be hesitant to embrace extended features due to the absence of support guarantees, making the process of obtaining feedback and conducting testing more challenging.
3. Does this actually solve converting between API versions that have differences in their schema? What else is needed to solve this?
4. [Graduation criteria](#) for promoting a Extended feature to Stable must be clearly defined, easily measured and adopted across Istio.
5. How unstable is the Extended Channel?
 - a. Backwards incompatible Schema changes
6. As the API versions don't change (always v1 or v1alpha1), how will users know what iteration of the API Version they are using?
7. As Istio is an existing project, there are migration challenges.
8. There are implicit semantic version assumptions that need to be considered. What are they? Is it cross cutting and can be solved with [Compatibility Versions](#)?
9. How do we track features/behaviors that are cross cutting functionality touching multiple CRDs? Re: [comment](#).
10. Would installing just one Extended Channel CRD to test a new field on a v1 stable/extended resource be an allowed/expected user workflow? (This is a bit awkward in the Gateway API last I checked, where installing all CRDs from the same Channel is the behavior facilitated by the documentation.) See comment [here](#)
 - a. Based on allowing Istio users more flexibility, yes we should allow and streamline installing CRDs from a different Channel regardless of the default Channel used for all APIs.
11. Is the stability of Extended Channel worse than the stability of Alpha and below APIs in the legacy model?

Alternative Approaches

Multiple CRDs



API Versions

Typically, an API version is the **only** semantic reference needed to inform the user of the expected features and behavior of that iteration of the API. This is the case for the **v1** API versions in all Release Channels.

However, we need to extend this reference to effectively allow the flexibility of experimenting with new features while still delivering a stable API. In this regard, the [Extended Channel](#) **v1alpha1** API version differs from the [Stable Channel](#) **v1alpha1** API version, as the non-GA features in the **v1alpha1** API version are not included in the [Stable Channel](#). This is a workaround for the operational inadequacy of Kubernetes CRD version management and our users must be clearly aware of this.

We will guide users with:

- Thorough documentation and easy association of CRD to target Release Channel
 - An annotation to the CRD that denotes which Istio version it was released in and channel it is designed to work with

istio.io/istio-version: v1.2.2

istio.io/channel: stable

- Error feedback for known issues that users may encounter when attempting to use a [Extended Channel](#) **v1alpha1** API in the [Stable Channel](#). For instance, if the user is attempting to use a non-GA field in the [Stable Channel](#).

Istio Releases

As **v1** and **v1alpha1** resources can change between Istio Releases, installed CRDs are tightly coupled to the installed Istio version. For example, "I'm using the Extended Channel **telemetry/v1alpha1** CRDs from Istio 1.21".

As such, users using the [Extended Channel](#) must pay close attention to the Release Notes for guidance on steps to take before upgrading to minimize data loss.

User Flow

1. User installs Istio specifying the target release channel, the [Extended Channel](#) will be installed by default.
 - a. We can explore eventually making the [Stable channel](#) the default so new users can explicitly opt into using more robust Istio features, by switching to the [Extended Channel](#)
 - b. This is implemented in [Helm](#)
2. CRDs for target release channel are installed. These CRDs contain all supported versions, **v1alpha1** and **v1**, but with different schemas based on the release channel.
3. User creates CRs that conforms to the schema of the installed CRD versions for target release channel.

Scenarios

- When a non-GA field is removed from a resource in the [Extended Channel](#)
 - Users must remove their usage of the field before upgrading. An analyzer should be created for this. Only once usage is removed, the user should upgrade their Istio base helm chart. The order is important here because the latest CRD version must be the stored version to prevent data loss (e.g. in the case of an added field). If usage isn't removed before upgrading to a CRD version where the field does not exist, all of the CRs in etcd will have that field dropped creating unpredictable runtime behavior (imagine all your timeouts being deleted for example).
 - Recommend upgrading with canary revision strategy
<https://istio.io/v1.16/blog/2021/revision-tags/>
- When a user wants to migrate from the [Extended Channel](#) to the [Stable channel](#)
 - Changing from [Stable Channel](#) to [Extended Channel](#) is relatively easier than going the reverse way. Similar to above, the user will be required to update their

usage of non-GA fields before switching to the [Stable Channel](#) to prevent data loss.

- However, Kubernetes will actually persist unknown fields in CRs if you change to a different schema. [This](#) behavior is controlled by `x-kubernetes-preserve-unknown-fields: true`. At this time, we have `x-kubernetes-preserve-unknown-fields: false`
- Removal of a resource in the [Extended Channel](#) as it is not widely used and not working well
- When a User wants to use the [Stable Channel](#) with a specific extended/non-GA resource/field.
 - Would installing just one [Stable Channel](#) CRD to test a new field on a v1 stable resource in the [Extended Channel](#) be an allowed/expected user workflow?
 - A custom CRD that can be created and used in the Base Helm Chart

Release Channels

Represented as an OpenAPI generated YAML file that contains all CRDs and its relevant API versions for each channel.

As of now, a [customresourcedefinitions.gen.yaml](#) containing all Istio APIs and API versions exists and is used to install all CRDs.

We are proposing generating one for each channel, for example, *extended.gen.yaml* and *stable.gen.yaml*.

Upon installation ([helm](#) or `istioctl`), one of these YAML files will be used to install Istio CRDs according to the set channel. The [base Helm chart](#) values will be modified to facilitate selecting the right YAML based on user configuration.

In summary, each CRD will be configured based on the below table:

	Stable Channel	Extended Channel
YAML File	stable.gen.yaml	extended.gen.yaml
Stored version of CRDs:	v1alpha1 without non-GA fields	v1alpha1 with non-GA fields
Served version of CRDs:	v1alpha1 without non-GA fields and v1	v1alpha1 with non-GA fields and v1

All channels will be aware of all API versions. However, the v1alpha1 API versions in the [Stable Channel](#) differ from the ones in the [Extended Channel](#) by the exclusion of non-GA fields. A user will need to be explicitly aware that they are using a v1alpha1 [Stable Channel](#) CRD vs v1alpha1

[Extended Channel](#) CRD. We will add relevant tooling to surface typical errors that may occur based on this.

The below table shows a more granular view of the end state of what happens to each CRD in Istio, factoring the removal of versions may take several releases.

	Stable Channel	Extended Channel
YAML File	stable.gen.yaml	extended.gen.yaml
CRDs		
AuthorizationPolicy	Stored version: v1 Served version: v1 Proposed changes: v1beta1 is dropped	
DestinationRule	Stored version: v1 Served version: v1 Proposed changes: v1beta1 is promoted to v1 . v1beta1 is deprecated v1alpha3 is deprecated	Proposed changes: v1alpha3 is deprecated and will not be introduced to Extended Channel
EnvoyFilter		Stored version: v1alpha3 Served version: v1alpha3
Gateway	Stored version: v1 Served version: v1	Proposed changes: v1alpha3 is deprecated and will not be introduced to Extended Channel

	<p>Proposed changes: v1beta1 is promoted to v1.</p> <p>v1beta1 is deprecated.</p>	
PeerAuthentication	<p>Stored version: v1</p> <p>Served version: v1</p> <p>Proposed changes: v1beta1 is promoted to v1.</p> <p>v1beta1 is deprecated.</p>	
ProxyConfig	<p>Stored version: v1beta1</p> <p>Served version: v1beta</p>	<p>Proposed changes: v1alpha3 is deprecated and will not be introduced to Extended Channel.</p>
RequestAuthentication	<p>Stored version: v1</p> <p>Served version: v1</p> <p>Proposed changes: v1beta1 is deprecated.</p>	
ServiceEntry	<p>Stored version: v1</p> <p>Served version: v1</p> <p>Proposed changes: v1beta1 is promoted to v1.</p> <p>v1beta1 is deprecated.</p>	<p>Proposed changes: v1alpha3 is deprecated and will not be introduced to Extended Channel.</p>
Sidecar	<p>Stored version: v1</p> <p>Served version: v1</p> <p>Proposed changes: v1beta1 is promoted to v1.</p> <p>v1beta1 is deprecated.</p>	<p>Proposed changes: v1alpha3 is deprecated and will not be introduced to Extended Channel</p>

Telemetry API	<p>Stored version: v1alpha1 without non-GA fields</p> <p>Served version: v1 as proposed here, v1alpha1 without non-GA fields</p>	<p>Stored version: v1alpha1 with non-GA fields</p> <p>Served version: v1alpha1 with non-GA fields, v1</p>
VirtualService	<p>Stored version: v1</p> <p>Served version: v1</p> <p>Proposed changes: v1beta1 is promoted to v1.</p> <p>v1beta1 is deprecated.</p>	<p>Proposed changes: v1alpha3 is deprecated and will not be introduced to Extended Channel</p>
WasmPlugin		<p>Stored version: v1alpha1</p> <p>Served version: v1alpha1</p>
WorkloadEntry	<p>Stored version: v1</p> <p>Served version: v1</p> <p>Proposed changes: v1beta1 is promoted to v1.</p> <p>v1beta1 is deprecated.</p>	<p>Proposed changes: v1alpha3 is deprecated and will not be introduced to Extended Channel</p>
WorkloadGroup	<p>Stored version: v1</p> <p>Served version: v1</p> <p>Proposed changes: v1beta1 is promoted to v1.</p> <p>v1beta1 is deprecated.</p>	
Theoretical v1 CRD with Extended Fields, no v1alpha1 exists	<p>Stored version: v1 without non-GA fields</p> <p>Served version: v1 without non-GA fields</p>	<p>Stored version: v1 with non-GA fields</p> <p>Served version: v1 with non-GA fields</p>

CRDs

Each API is defined as Proto Definitions in the [istio/api](#) as **v1alpha1** and/or **v1** versions.

Annotations in the form of comments will dictate CRD generation per channel.

Extended resources will be annotated to inform the generator of its target, [Extended Channel](#).

```
// +cue-gen:EnvoyFilter:releaseChannel:extended
```

Extended fields in the **v1** and **v1alpha1** resources will be denoted with a `releaseChannel:extended` annotation.

```
// +cue-gen:Telemetry:releaseChannel:extended  
map<string, CustomTag> custom_tags = 5;
```

Istiod

Based on the proposed design above, Istiod will be configured to read the [Extended Channel v1alpha1](#) of each CRD, which will be the superset of all supported API features.

Helm

Helm provides several ways to manage Custom Resource Definitions (CRDs) in your deployment:

1. By placing CRDs in the [CRDs](#) directory of the chart.
 - a. Helm will install CRDs from the ``crds`` directory before installing the rest of the chart. This ensures that the CRDs are installed in time for any resources that might need them.
 - b. Helm does not apply any templating to files in the ``crds`` directory. This means you can't use Helm's templating features to customize the CRDs based on values.
 - c. Helm does not manage the lifecycle of CRDs installed this way. It will not upgrade or delete them when you upgrade or delete a release.
 - d. The `--skip-crds` flag is effective in this scenario.
2. By managing CRDs as a Helm Template in the ``templates`` directory.
 - a. Helm treats CRDs in the *templates* directory like any other resource. This means they are included in the Helm release and can be installed and upgraded along with the rest of the resources in the chart.
 - b. The `--skip-crds` flag is ineffective in this scenario.
 - c. You can use Helm's templating features to customize the CRDs based on values.
 - d. If a CRD is installed as a template, it might not be installed in time for other resources that need it. This can cause errors if other resources are created before the CRD is installed.

3. By managing CRDs as a Pre-Install Hook in the `templates` directory.
 - a. A pre-install hook is best if the CRDs are tightly coupled with your application and are always needed before installing or upgrading the application.
 - b. Pre-install hooks run before any other templates are loaded. This ensures that your CRDs are installed and ready before any other resources are created.
 - c. Resources created via hooks are not managed as part of the Helm release lifecycle. This means they won't be upgraded or deleted when the Helm release is upgraded or deleted.
4. By using a separate chart for the CRDs
 - a. Use a separate chart if the CRDs are used by multiple applications or charts. This allows you to manage and version your CRDs independently.
 - b. However, managing CRDs in a separate chart can add complexity, as you'll need to ensure that the CRD chart is installed before any charts that use those CRDs.

Our Design:

- CRDs will be managed as a Pre-Install Hook in the `templates` directory.
- CRDs will be skipped on *helm upgrade* as upgrading CRDs should be done manually, or through an Istio provided tool to ensure existing resources aren't broken. This is mainly an issue when upgrading in the [Extended Channel](#) and when moving from the [Extended Channel](#) to the [Stable Channel](#).

Istiod Schema Collections

In Istiod, there is a concept of collections, which represents different permutations of APIs to be consumed by Istiod based on user configuration. For instance, if the user [enables Gateway API](#), Pilot will [consume a collection](#) that includes the stable Gateway API.

Extending on this concept, a Stable Collection will be created to represent the Stable Channel which will be enabled using an environmental variable/feature flag on installation.

The collection for the Stable Channel may look like the following:

```
// PilotStableChannel contains only stable collections used by Pilot including the
full Gateway API.
pilotStableChannel = collection.NewSchemasBuilder().
    MustAdd(AuthorizationPolicy).
    MustAdd(DestinationRule).
    // unstable. remove or mark as exception due to it being legacy
    MustAdd(EnvoyFilter).
    MustAdd(Gateway).
    MustAdd(GatewayClass).
    MustAdd(HTTPOutlet).
    MustAdd(KubernetesGateway).
```

```
MustAdd(PeerAuthentication).
MustAdd(ProxyConfig).
MustAdd(ReferenceGrant).
MustAdd(RequestAuthentication).
MustAdd(ServiceEntry).
MustAdd(Sidecar).
// need to be graduated to stable
MustAdd(Telemetry).
MustAdd(VirtualService).
// unstable. remove or mark as exception due to it being legacy
MustAdd(WasmPlugin).
MustAdd(WorkloadEntry).
MustAdd(WorkloadGroup).
Build()
```

The environment variable/feature flag may look like the following:

```
EnableStableChannel = env.Register("PILOT_ENABLE_Stable_CHANNEL", true,
    "If this is set to true, support for Stable Channels will be enabled. In
    addition to this being enabled, the Stable Channel CRDs need to be installed.").Get()
```

Notes from 4/11 TOC meeting

Keith Mattix

9:08 AM

<https://github.com/istio/istio/pull/50358>

Eric Van Norman

9:08 AM

<https://github.com/istio/istio/pull/50358>

Costin Manolache

9:19 AM

We already have the v1

That's clear

The only issue is new stuff we might add to v1

Costin Manolache

9:20 AM

For new CRs that we may add - assuming we follow the pattern in Gateway (and we should since new CRs should work with ambient and gateway) - no problem

Whitney Griffith

9:20 AM

Its captured in Scenarios here

<https://docs.google.com/document/d/1onST4-swbZE1UPCDQM1c7T5rzYEn7Wo4uFNShz0kNw/edit#heading=h.p4568phsujd6>

John Howard

9:21 AM

I thought we were planning to move everything to v1

Costin Manolache

9:21 AM

We should stop 'moving without change'

Keith Mattix

9:21 AM

The only non-v1 things are WasmPlugin and EnvoyFilter. A (large) subset of Telemetry is v1 but it is a strict subset

Costin Manolache

9:22 AM

That's the fundamental problem. Experimental is not reviewed as a v1, and no changes are possible

John Howard

9:22 AM

@Costin I don't think GW is going with the "Separate name for experimental" btw. It was a proposal by Rob that got -1'd by most of the community

Mike Morris

9:23 AM

we _could_ reintroduce (or add later if needed) a separate Experimental channel to allow more flexibility in breaking changes

John Howard

9:24 AM

...

matchResources:

namespaceSelector:

matchLabels:

istio.io/rev: bar

...

Costin Manolache

9:25 AM

Revisions are not really for users but vendors

John Howard

9:27 AM

If only GCP's mesh supported revisions :-)

Costin Manolache

9:27 AM

That doesn't mean we should keep the alpha approach.

We can still have a separate CRD for each experimental feature

Mike Morris

9:28 AM

agree that revisions are important - im trying to understand more where the conflict between revisions and CRD channels would be, i dont think i quite get the issue there yet

Costin Manolache

9:28 AM

For example EnvoyFilter or Wasm could be in a different space instead of istio.io

For revision you need canary to have the new stuff

Costin Manolache

9:30 AM

It's really not just upgrade - for example you may want envoyfilter or wasm for a very small controlled set of workloads

Keith Mattix

9:31 AM

We need some messaging for Telemetry but I think I'm cool with that

Costin Manolache

9:37 AM

Not so novel. ServiceExport

Mike Morris

9:39 AM

@mitch <https://github.com/kubernetes-sigs/gateway-api/pull/2912>

Justin Pettit

9:39 AM

Sorry, I need to drop for another meeting.

Mike Morris

9:39 AM

<https://github.com/kflynn/k8s-versioning> has some good context too, which covers similar ground as a blog post

John wrote a while back <https://blog.howardjohn.info/posts/crd-versioning/>

John Howard

9:40 AM

FWIW I disagree with a lot of the opinions part from Flynn there ^. But good info

Keith Mattix

9:43 AM

Decision for 1.22:

1. Add support for revisions to my draft PR
2. Add helm value (under experimental or something) for enabling channels (opt-in_
3. Create docs for users indicating that channels will be default in the future

Whitney Griffith

9:43 AM

Thanks for all the feedback!

Keith Mattix

9:44 AM

4. Add Alternatives Considered section to Release channels doc

3* - Create docs for users indicating that we hope channels will be default in the future pending feedback

Whitney Griffith

9:45 AM

I agree with that as well Mitch! Thank you!

References

[Thinking about alpha/beta/GA in k8s \(public\)](#)

[\[SIG-NETWORK\] Phasing Out Beta From Gateway API](#)

[CRD Upgrades are Easy to Get Wrong](#)