Google Summer of Code

PROPOSAL

Improve Third Party Resources

By
Nikhita Raghunath

CLOUD NATIVE COMPUTING FOUNDATION

kubernetes

TABLE OF CONTENTS

1. Abstract

ThirdPartyResources are already available but the implementation has languished with multiple outstanding capabilities that are missing. They did not complete the list of requirements for graduating to beta.

Hence, there are multiple problems present in the current implementation of ThirdPartyResources. This project aims at working towards a number of known shortcomings to drive the ongoing effort toward a stable TPR release forward.

2. Background about TPRs

2.1. Why TPRs are needed

Kubernetes comes with many built-in API objects. However, there are often times when one might need to extend Kubernetes with their own API objects in order to do custom automation.

ThirdPartyResource objects are a way to extend the Kubernetes API with a new API object type. The new API object type will be given an API endpoint URL and support CRUD operations, and watch API. One can then create custom objects using this API endpoint.

2.2. Why this project is needed

Currently, TPRs are being used heavily in multiple production environments. A list of known users using TPRs can be found here (not exhaustive). However, there is a list of known issues that cause heavy crashes and breakages. Tracker issue: kubernetes/features#95.

This project proposal aims at working towards many of the known issues, some of which may require breaking API changes. TPRs are of great significance since they help in extending the Kubernetes API and thus the major goal of this proposal is to make sure that TPRs can be used safely without any inconsistencies.

3. Project Details

This proposal aims at working towards a fix for the following list. The list defines the issues with proposed solutions that will be the basis for work in the project.

3.1. REST API Semantics

Goals:
TPRs need to have the following API semantics:

- TPRs are intended to look like normal kube-like resources to external clients. In order to do that effectively, they should respect the normal get, list, watch, create, patch, update, and delete semantics.
- In a kube-like API, different versions of the same resource.group are related, so `resource.v1.group` is GET-able via `resource.v2.group`. This provides a seamless migration between versions. This same convention could be applicable to TPRs as well.

Issues:

- Multiple versions are not supported.
- Only the first TPR created in a group version is installed.

Proposed solution:

- Have a single version for a TPR and hardcode it to `tpr`. This allows for easy identification, prevents mistakes, allows for TPR authors to migrate resources, and keeps client tools working.
- Implement this in the add-on server. See this document for more details.
- Show an example of how it should work: see comment for further details.

Related Links:

- PRs: #36977, #24299, #28414, #29724
- Issues: #25570, #23831

3.2. Finalizers

Goals:

- If there is a non-empty "finalizers" field in the TPR, the instances of the TPR are deleted before the TPR itself is deleted.

Issues:

- Even if a non-empty finalizer field is present, the object gets deleted.

Proposed solution:

- Delete all the `thirdpartyResourceData` when the `thirdpartyResource` is deleted.
- Need to get this done since things will break when there is a recreation of the same TPR.
- Use synchronous garbage collection.
- Have a strong control of an unconditional finalizer.

Related Links:

- Issues - #40715, #35949

3.3. Garbage Collection

Goals:

- The Garbage Collector should be able to get REST Mapping for ownerRefs to TPR and add-on APIs (basically pointing to any arbitrary unknown kind).

Issues:

- The Garbage Collector does not know how to look up ownerRefs to anything other than compiled in types.
- It currently uses static REST Mapping.
- Was hot-lopping the error message but fixed by PRs mentioned below (prints error only once now).

Proposed solution:

- Use a dynamic REST Mapper so that it picks up types the server knows about, not just compiled-in types.
    - discover types at startup

    - since it is referencing a dynamic kind (either a kind contributed by an add-on server or by a thirdpartyresource), it should re-discover on some interval to recognize the kind.
- Object with ownerReferences referring non-existing tpr objects could be deleted by the garbage collector.

Related links:
- PRs - #40497, #42862, #42615
- Issues - #39816

3.4. Self-links

Goals:
- The self-link for TPR resources should contain the missing forward slash.

Issues:
- Currently, when you do a `$ kubectl get thirdpartyresource -o json`, you get a missing forward slash in the self-link: `"selfLink":"/apis/extensions/v1beta1/thirdpartyresourcesfoo.stable.example.com"`

Proposed solution:
- Refer the line of code here.
    - the root cause should be no forward slash between pathPrefix and name of rootScopeNaming, not related with pathSuffix.
    - There is a forward slash already present in the suffix.

Related Links:
- PRs - #41011, #37686
- Issues - #37622

3.5. Validation and Defaulting

Goals:
- Prevent invalid objects to be stored from the server side in the first place and also give useful feedback on what is wrong with it.
- Allow the TPR author to supply the validation code.
- Current implementation includes:

    - ThirdPartyResource: validates name (3 segment DNS subdomain) and version names (single segment DNS label)
    - ThirdPartyResourceData: validates objectmeta (name is validated as a DNS label)
    - removes ability to use GenerateName with thirdpartyresources (kind and api group should not be randomized)

Issues:
- Currently to validate a particular resource, the only way is to
    - tear everything with the resource down
    - halt for the user to solve
- Basically, hardly any validation at all.

Proposed solution:
- Use a webhook to build a HTTP validation endpoint to the designated/configured operator.
- Build a HTTP service for each use case. Will this lead to breaking changes or introduce unwanted dependencies to:
    - external service
    - introducing local cluster validation
- Make sure to reduce the amount of special-case handling of ThirdPartyResources so that from the outside, they work similarly to federated API groups.
- Alternate: Provide JSON schema specifications with each TPR.

Note:
- It's probably easier to make a generic API server that loads TPRs and can apply custom admission code than increasingly complex externalization hooks.
- Run all TPRs from a separate API server - it would be easier to write that way.

Related Links:
- PRs - #25007
- Issues - #38117

The first major milestone is to create a design proposal for validation and then implement it.

3.6. Sub-resources

Goals:

- Having something like a `/status` field in the custom controller to inform about the current status of whatever is being managed by the TPR's controller. Have a `/scale` field too.
- There should be an option to have sub-resources in TPRs (just like Deployments, StatefulSets, etc. have) which is modified independently through its own API endpoint.

Issues:
- Providing a `/status` subresource requires semantic knowledge of the type being stored. By their nature, TPRs do not provide that semantic knowledge to the API server hosting them.
- The main reason that `/status` has value is the ability to control which bits of a resource a "normal" client can update and which bits of a resource a "privileged" client can update. Doing this means interpreting and stomping out certain parts of an object.
- `/scale` is even more tenuous. Each resource decides which part of its object gets controlled. Its different for different resources. See Jobs versus Deployment.

Proposed solution:
- If you want to the API server to interpret the various fields inside of a TPR mean (barring perhaps metadata), write your own API server.
- TPR can be made to "opt-in" to the common patterns. spec, scale, and status are the obvious three. Define clear rules about what checking the relevant box says about the schema of the TPR.
- Note: It might be easier to implement TPRs from a separate API server.

Related links:
- Issues - #38113, #36885.

The second major milestone is to create a design proposal for sub-resources and then (optionally) implement it.

3.7. Add unit and integration tests

Goals:
There is a lack of unit tests for TPRs. We should at least have the following unit tests:
- create object with unknown group/version (covers TPR creation scenario)
- changing patch of object with unknown group/version (covers TPR apply scenario)
- no-op patch (idempotent apply) object with unknown group/version (covers TPR re-apply scenario)

- changing patch of known object (like deployments) in an unknown version (will need to add the unknown version to the fake discovery doc) - makes sure kubectl is resilient to future versions of known objects being introduced.
- Write tests for all concrete bugs in kubernetes/features#95.

Related links:
- Issues - #40841
- PR - #40956

3.8. Posting YAML

Goals:
- TPRs should support YAML. It works with `$ kubectl create -f manifest.yaml` because the YAML manifest is converted client-side to JSON before calling out to the API server.
- TPRs should also support YAML while doing a POST request

Issues:
- Currently, the content type is always set as the json serializer no matter which content type is sent - JSON or YAML.
- Hence, user is unable to send a POST request using a YAML file.

Proposed solution:
- Refactor the ThirdPartyResource serializer negotiation part so that it supports application/yaml is also supported.

Related Links:
- Issues - #37455

4. Schedule of Deliverables

- There are two main milestones. Namely, coming up with:

    1. A design proposal for Validation and implementing validation.
    2. A design proposal for Sub-resources and (optionally) implementing sub-resources.

- Point 2 mentioned above (implementing sub-resources) is *optional* and will be completed if time permits.

- If more things come up or plans change, they can also be incorporated.

- The schedule of deliverables has been designed keeping in mind the two milestones. Please note that the design proposals will be a matter of community discussions. Community discussions take a long time and will be stretched throughout the GSoC period. Thus, the community discussions, implementation of tests and tackling of less-controversial issues will go on in parallel.

    - For first evaluation - Publish design proposal for validation and sub-resources and implement tasks mentioned in the table below.
    - For second evaluation - Implement validation as per the design proposal.
    - For the final evaluation - Start work on the implementation of sub-resources.

- The ground work to tackle the REST API semantics issue is on the way. My work will be based on that and would be supporting these efforts.

- A few days before the evaluation period have been left free for completing any remaining work (if any). This provides sufficient cushion for making sure that the timeline is followed.

Key Dates	Task	
Community Bonding Period begins		
5 May - 14 May	Draft validation proposal and publish it in the community	
15 May - 30 May	Draft sub-resource proposal and publish it in the community	
Community Bonding Period ends		
31 May - 10 June	Address comments and Prototype ideas around validation	Add unit and integration tests for API semantics
11 June - 20 June		Add tests and fix behaviour of controllers

21 June - 30 June		Add YAML support
First Evaluation		
1 July - 25 July	Implement validation as per design proposal	
Second Evaluation		
29 July - 21 Aug	Implement sub-resources as per design proposal	
21 Aug - 29 Aug	Submit final code and project summaries	
Final Evaluation		

Obligations:

- I will be available to work full time (a *minimum* of 40 hours per week) during the GSoC period.
- My university classes begin on 15 July. However I will not be having any exam sessions until 15 August so working for the project should not be a problem.

5. General Notes

I believe that communication is a vital aspect of GSoC and to ensure that the status of the project is communicated properly, I will be undertaking the following steps:

- Publish a blog post every week detailing:
  - Actionable items delivered that week.
  - Hurdles faced while tackling the issues.
  - How I overcame the hurdles.
- Try to maintain a google doc with *daily* updates to the work done for the day. (Even if I am stuck at some issue, I'll be writing that down). Previous experience taught me that this was a very efficient way to maintain accountability and I'd like to continue this.
- Contact the mentor daily to keep him in the loop of how the work is progressing.
- Participate in bi-weekly SIG meetings.
- Maintain a github meta-tracker repository containing the following, so that everything is available at one place:

    - A notes folder containing all notes that I took while getting to understand the task at hand.
    - A link to the google doc with daily update.
    - Link to blog posts (weekly updates).
    - List of issues that I am currently working on/have closed.
    - List of Pull Requests that I have opened for the project.

6. About Me

6.1. Personal Information and Contact Details

Name: Nikhita Raghunath

Email: nikitaraghunath@gmail.com

Slack/IRC nick: nikhita

Github username: nikinath

University: VJTI, Mumbai.

Timezone: GMT +05:30 (India)

Website: https://nikinath.github.io/

Twitter: https://twitter.com/TheNikhita

6.2. Why Me

I have been working with Kubernetes for that past 2 months. I have experience in deploying applications to Kubernetes clusters and lately, I have been intrigued by the development at Kubernetes.

I have looked at the code base and opened the following Pull Requests until now:

a) https://github.com/kubernetes/kubernetes/pull/43591
b) https://github.com/kubernetes/kubernetes/pull/43606
c) https://github.com/kubernetes/kubernetes/pull/43573

While going through the issues (and exploring TPRs), I found that most of them were fixed but still open. So I added comments mentioning how it was fixed and referenced to the code that fixed it. (links added for the sake of completeness)

a)https://github.com/kubernetes/kubernetes/issues/10309#issuecomment-288196121
b) https://github.com/kubernetes/kubernetes/issues/6703#issuecomment-288221149
c) https://github.com/kubernetes/kubernetes/issues/37554#issuecomment-289778842
d) https://github.com/kubernetes/kubernetes/issues/37278#issuecomment-289780021
e) https://github.com/kubernetes/kubernetes/issues/19317#issuecomment-289780972
f) https://github.com/kubernetes/kubernetes/issues/3184#issuecomment-285935360

I have started working on more issues and will be sending in more PRs in the coming weeks.

Experience in Go:

- I have previous experience in programming in Go - mainly while contributing to https://github.com/taskcluster/taskcluster-cli. TaskCluster is Mozilla's internal CI system and the project was to build a CLI tool for them.
- I also presented a talk at GopherCon India.
- I am fairly comfortable programming in Go.