Google Summer of Code

Proposal

Add Plugin Mechanism to the Dashboard

By
Ajat Prabha

CLOUD NATIVE
COMPUTING FOUNDATION

kubernetes

Table of Contents

Basic Information

Name	Ajat Prabha
Major	Computer Science And Engineering
University	Indian Institute of Technology Jodhpur, India
Github	@ajatprabha
Slack	ajatprabha

Email	ajat.prabha.leo@gmail.com, prabha.1@iitj.ac.in
Phone	(+91) 86302-96147
Blog	https://ajatprabha.in
Twitter	@ajatprabha
Postal Address	Room No. 307, Hostel B1, IIT Jodhpur Hostels, IIT Jodhpur, Rajasthan, India - 342037
Timezone	Indian Standard Time (UTC +5:30)

Project Details

Add plugin mechanism to the Dashboard

Synopsis

This project aims to introduce a plugin mechanism to the Kubernetes Dashboard. It shall deal with defining the plugin framework architecture, its scope, how it could enhance the Dashboard UI and make it possible to utilize third-party APIs to extend its functionality.

Why this project?

Dashboard is a great tool when visualization of the cluster is concerned. It provides useful information from the cluster in a good UI. But one major concern is that whenever a new feature is needed, the core code of the Dashboard has to be modified. This hinders many people from getting their feature baked into Dashboard easily. The scope of this project lies around tackling this problem and enabling the developers to integrate their own features into the Dashboard.

Issue k/d#1832 started the discussion around a plugin mechanism and there are issues opened on GitHub which require changes to the Dashboard, for example:

1. kubernetes/ingress-nginx#109 raised a need for CRUD operations on Ingress resources.
2. kubernetes/ingress-nginx#2480 requires enhanced experience with Ingress resources.
3. kubernetes/dashboard#1577 proposed addition of Weave Scope to the Dashboard.

All of the above issues are suitable candidates that will benefit after the Dashboard supports a generic plugin mechanism. The above features can be developed individually as separate plugins without the need to make changes to the Dashboard's core code.

Another area that can benefit from this plugin mechanism is the CustomResourceDefinitions. The dashboard will support the CRDs in a generic UI as it is in the roadmap but these CRDs can benefit even more in terms of enhanced control, analytics, etc. with the plugin mechanism.

Brief Overview

I propose the following architecture where it shall broadly be thought in terms of:

1. Plugin Manifest
2. Plugin lifecycle hooks
3. API Interaction layer

Note: This proposal is not a definite outline of the final outcomes and further improvements will be made as part of GSoC contributions.

Plugin Manifest

This idea revolves around the declaration of the details about the plugin, the idea is very similar to what we see in other frameworks such as package.json, Android Manifest, etc. This manifest shall help in terms of:

1. **Metadata**: Useful metadata such as plugin's name, author, description, license, etc can be specified.
2. **Versioning:** Developers can specify the plugin version, specify minimum Dashboard version for the plugin to work and even release beta versions of the plugin. This will help in packaging and distribution of the plugin.
3. **Security**: It can specify what type of permissions are required by the plugin to function viz. storage access, K8s API interaction, network interaction, etc.

Plugin Lifecycle Hooks

The plugin mechanism shall provide lifecycle hooks. A lifecycle hook is something which the core will execute based on its definition. Some examples of potential lifecycle hooks:

1. **Installation Hook**: This shall deal with the installation of the plugin, it will be executed when the plugin is being installed for the first time. It can be used to do certain things which are required for proper functioning of the plugin.
2. **Uninstallation Hook**: This shall deal with the uninstallation process, can be used for cleanup tasks, etc.
3. **Activation/Deactivation Hooks**: This can be used if the plugins can be activated/deactivated temporarily with uninstalling the plugin.
4. **Mounted Hook**: This hook shall be executed when the plugin is mounted in the dashboard's UI and it becomes visible to the user.

There can be more such well-defined hooks which the plugin mechanism should define and document. If accepted we'll need more evidence to define the hooks with their execution point in the lifecycle state diagram which will depend on the use cases and flexibility provided by the plugin mechanism.

API Interaction Layer

A well defined API should expose the core features/functionalities to be utilized by the plugin. It shall define a set of interfaces such as:

1. **PluginInterface**: This interface should be implemented by the plugin and later when the plugin is installed, lifecycle hooks will use methods in this interface to integrate the plugin with the Dashboard, such as mounting, etc.
2. **APIInterface**: This interface will be responsible to expose the K8s API to be used by the plugin. Basic CRUD operations can be permitted to the plugins. Permission management is one area which needs proper validation.
3. **NetworkInterface:** This can be used to expose services like the *HttpClient* or a custom client to the plugin so that the plugin can make calls to external APIs and the calls can also be proxied by the backend server to avoid CORS issues.

A package similar to @angular/core etc. should be written which will contain a set of these plugin interfaces.
Note: Names such as PluginInterface should be reconsidered after further discussion in the KEP.

Why me for this project?

I was introduced to Kubernetes last summer when I was interning with an Indonesian decacorn GO-JEK. I was amazed by the potential Kubernetes has and the scale at which it operates. I joined the Kubernetes developer community back in November 2018 and my experience with such a huge community has been great.

Related Work

I've opened a few PRs since then, most of which are accepted. I've read the code base and worked towards fixing a few issues:

1. Align action-bar actions with context actions.
    1.1. Issue: k/d#3440
    1.2. PR: k/d#3514
2. Add WorkloadStatusComponent for workload status overview.
    2.1. Issue: k/d#3440
    2.2. PR: k/d#3667

Issues Opened: k/d#3665, k/d#3609
I have also opened a KEP in which discussion regarding the architecture shall continue further.

Relevant Past Experience

While at GO-JEK, I worked on a highly performant and resilient on-the-fly image processing micro-service called Darkroom. It is written in **Golang** and deployed in production using Kubernetes. It usually serves 500,000 image requests/minute at its peak usage. I also worked

on an internal merchant facing frontend portal written in **React** for their on-demand food delivery service.

Open Source Experience

I started a student group called Devlup Labs at my institute to promote the use of FOSS which hosts some projects I've worked on along with other people. My GitHub profile has projects like a Central Authentication Service web app, wireless payments handling web app and some course projects. I've contributed to organisations such as ReadTheDocs.org, Systers, etc in the past.
I have good working experience with frameworks like Django and VueJS with projects deployed in production. One such project I'd like to mention is WoC (Winter of Code), it uses RESTful APIs extensively, the backend and the frontend are decoupled. This project helped me to get the gist of *writing modular code with modern frontend frameworks like React, Angular, VueJS using* ***JavaScript/TypeScript***.

Education

I'm a Computer Science and Engineering undergraduate that has enabled me to learn the core fundamentals which I can put to good use in this project. I strive to write code clean enough that it is pleasant to read and understand while following best practices.

Contributions

Apart from the above-mentioned contributions which are closely related to this proposal, I have also worked on issues outside kubernetes/dashboard repository.

1. Add ResourceVersion as a precondition for delete in order to ensure a delete fails if an unobserved change happens to an object.
    1.1. This contribution is a part of the v1.14 release.
    1.2. Issue: k/k#73648
    1.3. PR: k/k#74040
2. Minor updates to the Kubernetes e2e tests.
    2.1. Issue: k/k#34059
    2.2. PR: k/k#72440

Deliverables

There are two major milestones viz. the architecture proposal of the plugin mechanism in the form of a KEP and the implementation of the same through code. Since the architecture decisions may take quite some time I'm willing to work post-GSoC on the implementation part whatever is out of the scope of the timeline that GSoC permits. This being said, the following are the major deliverables:

1. **A plugin mechanism architecture enabling any external developer to**

    1.1. Add custom functionality to the dashboard by registering their own plugin with the Dashboard.
    1.2. Ability to interact with the K8s API server
    1.3. Ability to interact with external APIs
2. An abstraction of the above said functionalities as part of code contribution:
    2.1. Interface definitions that should be available to the developers to interact with the core components of the K8s Dashboard.
    2.2. Tests for the production code.
3. Detailed documentation of the architecture and code samples to help new developers dive in quickly.
4. Fortnightly blogs on developmental advances and milestones.

Timeline

Brief

- (Phase 0) Till May 6: Pre-GSoC Period
- (Phase 1) May 6 - May 27: Community Bonding Period
- (Phase 2) May 27 - June 24: Coding Period 1
- (Phase 3) June 24 - June 28: Phase 1 Evaluations
- (Phase 4) June 28 - July 22: Coding Period 2
- (Phase 5) July 22 - July 26: Phase 2 Evaluations
- (Phase 6) July 26 - August 26: Coding Period 3 and Mentor Evaluation Submission
- (Phase 7) August 26 - September 2: Final Evaluation

Detailed

Dates	Task
Community Bonding Period begins	
May 6 - May 17	Define the project's outcomes more clearly and validate it with mentors.
May 18 - May 27	Gather more evidence and discover potential implementation strategies.
Community Bonding Period ends	
May 27 - June 7	Draft KEP and publish it for review in the community.
June 8 - June 23	Work towards improving the KEP and take it to the point where it is accepted to be implemented.

June 24 - June 28	First Phase Evaluations
June 29 - July 7	Structure the new plugin mechanism and start its implementation along with integration into the existing codebase.
July 8 - July 22	Work on the implementation along with tests.
July 22 - July 26	Second Phase Evaluations
July 27 - August 17	Complete the implementation and work on the documentation with code examples for plugin developers.
August 18 - August 26	Submit final code and project summaries
August 26 - Sept. 2	Final Phase Evaluations

Time availability during GSoC

I will ensure that I put in 40 hours per week in the project. I have exams from April 29th to May 7th. So I will not be able to do much from April 22nd to May 7th. Although I'll be available on Slack occasionally. My summer vacation starts on May 8th and ends on July 21st. During this period I shall work full time on the project. I will not be having any exams until August 21st, so working on the project after vacation is not a problem and I'll work more on weekends to compensate for reduced time on weekdays during August.

Post GSoC

I would like to follow up on this project even after the GSoC program is over. I'll be happy to maintain this part of the project after the GSoC period.