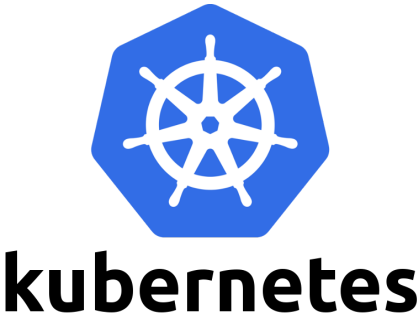
Google Summer of Code

PROPOSAL

Cluster Addons: Package all things!

By

Onyekwere Somtochi



Mentors:

Justin Santa Babara(@justinsb)

Leigh Capili (@steathlybox)

Table of Contents

1. Abstract

Cluster Addons are resources that are considered inherently part of the Kubernetes cluster as they help extend the functionality of the cluster. Over time different addons have surfaced with increasing complexity while the tools for managing these addons have not progressed as much.

The aim of this proposal is to build and package operators for different popular addons that are easy to use and follow best practices in various clusters.

2. Background

In simplest terms, addons are used to extend the functionality of a cluster. They are resources like Services and Deployments (with pods) that are shipped with the Kubernetes binaries. They are important and add to the extensibility of Kubernetes and perform various functions in the cluster.

Cluster Addons are considered an inherent part of the Kubernetes clusters. They are tied to other components and their lifecycle is managed alongside the cluster lifecycle. Managing these addons very well is of the essence.

Early add-ons existed in the k/k addons directly and were managed with the addon manager. The addon-manager had a lot of shortcomings.

Currently, there is no unified way of managing these add-ons. The bash addon-manager is not an efficient solution and isn't widely accepted.

Operators, on the other hand, are custom controllers and custom resource definitions that encode application-specific knowledge and are used for managing complex stateful applications. It is a widely accepted pattern.

Managing addons via operators, with these operators encoding knowledge of how best the addons work, introduces a lot of advantages while setting standards that will be easy to follow and scale.

Using add-on operators enables us to watch for the health and status of the resource object, reconciling the observed state with the desired state in the manifest.

The addon operator for CoreDNS has already been created and the Kube proxy-operator is also underway. This project proposal aims to build upon these existing operators and create more add-on operators for different popular addons.

3. Project Implementation

Goals

The goals of the project are to create, build and package a set of real-world add-on operators for a set of selected addons. The addons I have selected to work on are:

- Local Node DNS
- Calico
- Flannel
- Weavenet

I also aim to bring these two add-ons to completion. There are already pull requests open for them in the cluster addons repository.

- Kubernetes Dashboard (See #44)
- Kube Proxy (See #35)

The list is definitely open to modifications by the GSoC mentors, members of the cluster-addon project or the community. These add-ons were chosen because of their popularity and their functions to encourage faster adoption by the community.

Structure

A detailed list of the functionality for the add-on operators has been listed in the Addons via Operators KEP and this will be a guide for the implementation during the course of the project. The addon operators will have all of the needed items on the list. The most important ones are listed below.

- A custom resource definition
- An Application CRD instance
- They will follow a common structure
- Operators will be declaratively driven

The addon operators are generated from the same code using the kubebuilder declarative pattern. From here, encoding the addons' specific knowledge continues by adding functions to the controllers and custom fields to the custom resource.

Each add-on operator will also have end-to-end smoke tests to ensure that it works as intended and for testing different scenarios like creation, updating, deletion (CRUD) operations, configuration change . Golden file tests will be included to ensure that the addon is configured as expected based on the input of the custom resource.

Details

This section lists out some details for implementing the different addon operators to be created in this project. It outlines some additional specs for the custom resource and functionality the controller may require to manage the add-on properly.

Local Node DNS

Stable version: 1.15.10
This addon improves DNS performance by running a DNS caching agent on cluster nodes as a Daemonset The local node DNS is among the legacy Kubernetes add-ons and the YAML file can be found in this k/k addons folder. The different variables needed for the local node DNS like the `DNSDomain` which will default to cluster.local, `DNSServer` and `DNSServerIP` can be included in the spec of the custom resource of the addon-operator with the controller providing the default values if these fields in the YAML are empty.

The Local Node DNS will need to reuse a function[findDNSClusterIP] that the CoreDNS operator has to find the DNS Cluster IP. It could be worthwhile to have a utils folder in the cluster addons repository so that different operators can make use of the same function. Because the Local Node DNS needs an existing kube-dns service on the cluster to perform efficiently. The users can also change the DNS by adding a patch to the instance of the custom resources definition to modify the DaemonSet and Config.

Flannel

Latest version: 0.10.0
Flannel allocates subnets to different hosts in a Kubernetes cluster for use. It creates a virtual overlay network. Flannel requires some env variables to be set. The Flannel custom resource can have two fields in its spec - `Backend` and `MasqueradeCIDR`. The backend will default to `VXLAN` when not specified.

Calico

Latest versions: 3.13
Calico is an open-source networking and network security solution for containers, virtual machines, and native host-based workloads. This is one of the complex addons listed so far. We would need to set the environment variables and override the apiserver since we cannot depend on kube-proxy. Some part of this logic is already implemented in the kube-proxy API.

There is an existing operator for the Calico add-on in progress [here] fromTigera using the operator-SDK that is worth looking over.

Weave Net

Latest version: 2.6.2
Weave Net provides networking and network policy, will carry on working on both sides of a network partition, and does not require an external database.
The operator needs to check that the TCP port 6783 and UDP 6783/6784 aren't blocked. The installation docs give the manifest for the weave net addon(from Kubernetesv1.6 upwards) based on the kubectl version. This is a case where sourcing manifest through https is beneficial but for now we can include the different versions of the manifest.

Kubernetes Dashboard

This add-on is fairly simple and has already been added to the addons repository. The task left to be completed is to ensure that the operator works within and outside the cluster and the addition of end to end smoke tests. Golden file tests already exist for the addon operator.

Kube-Proxy

The pull request of the Kube-proxy already has a checklist of items to be done and this will be a guide for the work to be done on the operation. There is already base code and manifests. The two tasks left for the kube-proxy is to add golden file tests for the operator and ensure the operator can be run within the cluster. For the golden file test. We will have a folder containing both the output manifest files and the custom resources inside the controller folder. The reconciler logic might need to be separated from the setup manager function to prevent calling `watch` on the mock during tests. It will be modeled after the golden file tests for the dashboard operator[here].

Investigations and Finding

Packaging and Distribution

Manifest Bundle

While the Manifest Bundle KEP presents a good solution to standardize the way we package manifests and metadata associated with operators with Docker and OCI images. The enhancement proposal is yet to be approved and still has some work to be done before it can be fully used by existing operators. There are also other solutions like using git repositories, helm charts.

The AddonInstaller provides a nice way to use whichever solutions work best or are available at the moment while integrating nicely with tools such as kubeadm, kops, etc and other cluster management tooling. The manifest files can be pulled from the local filesystem, git repository, or even for docker or OCI manifest. A pull request for integration of the AddonInstallerConfiguration into kubeadm exists here. It adds an AddonCofiguration feature gate to kubeadm so users can optionally enable it and have more control over how addons are installed. A current solution is to get the AddonInstaller working in various tooling using the manifest as kustomize directory in a git repo. So in the future, the AddonInstaller can be easily extended to use Docker images.

Granularity of RBAC

Granularity in RBAC has to do with giving the operator only the least privilege that it needs to perform. Operators require RBAC privileges to carry out some functions. Clear and granular roles and role binding should be written for different components and the operator although there is no way to enforce this, it should be strongly advised and documented. There is a need for tools to analyze and generate role bindings to further automate the process.

The Operator Lifecycle Manager is an example of a tool that supports granular RBAC and does this by allowing the creator of the operator to define the privileges.

UX for cluster management tooling

This deals with how cluster management tooling like kops, kubeadm, etc can provide a smooth experience for users that want to install addons to their clusters and how easy it should be to opt-in and out of this feature.

Currently, for kops, add-ons are installed using the kubectl (same with kubeadm) or specifying it in `spec.addons` when creating the cluster from a YAML file. This should change when using operators to manage add-ons. Work is already being done on integrating the addon installer for kops and kubeadm.

First, I think not all addons should be installed automatically. Some important and popular addons like kube-proxy and CoreDNS should be installed (with the

custom resource and the operator) automatically while others should be left to the users to opt-in for both the custom resource and operators.
For everything that is installed automatically, there should be an option for the users to override.

Information meant to be on the cluster

Information about various addons and their components should be accessible to users. The deployment of an addon sets up owner references on the Kubernetes objects created pointing to the custom resource. These owner references can be used to find the objects and get their information but this can quickly become cumbersome.

The Addon discovery makes this easier by setting up a controller that watches for a label of the format `'discovery.addons.k8s.io/< name-of-addon >'` and aggregates references to resources. It creates a custom resource definition called addon and users can execute a command `` `kubectl get addons` `` and see all the addons installed listed. All the resources associated with a particular addon are also aggregated and their conditions nested in a YAML file. This makes it easier to check the status of the resources and gather information on addons.

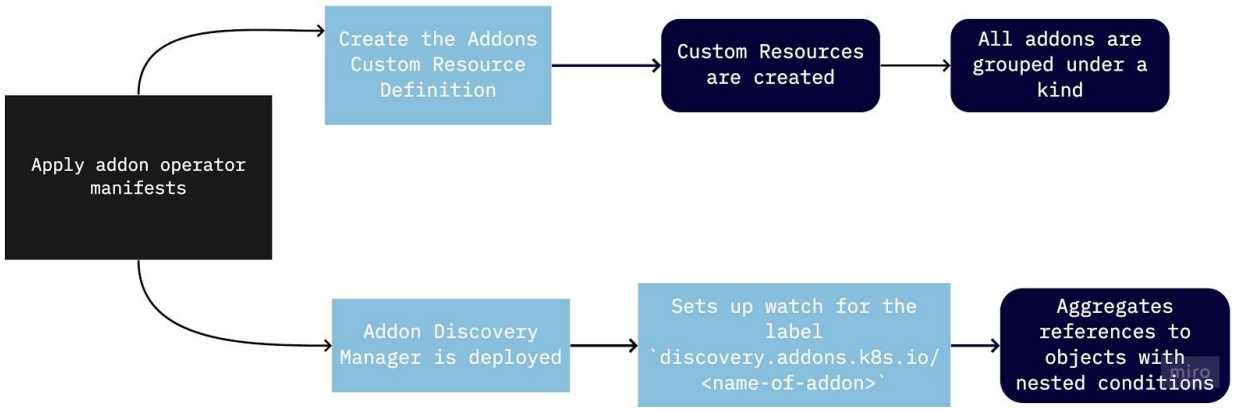


Fig 1. Flow of the Addon Discovery

Upgrades and Rollback between versions

Addons more often than not have different versions and for an operator to manage an addon well, it has to be able to handle upgrades and rollback of versions by the users seamlessly. Currently, the user can update the version of the addon in the custom resource for the operator to install and because we are using controllers, we have the opportunity to add more complex updates. For cluster addons that are

closely tied to kubernetes, the operator could check the version of kubernetes and update it to the correct version before updating the version. Encoding specific knowledge of an addon in operator also involves knowing how best to update the addon and what other changes within the cluster that should be made in order for the upgrade/rollback to be successful. There should definitely be tests around how operators update and rollback versions of an addon to ensure it works as intended.

Shared Operators between two addons

Sharing code between two generated addons that need complex functionality and don't change could help reduce the resources consumed by the controllers. Could this be extended for three to four controllers. Decoupling the addons into different operators as complexity increases would be no problem.

Addon health/status

The health/status of addons is very important to users of these addons as it helps to give insight and inform them if there is anything wrong with the addon. Currently, with the operators, The operators sets up watches for the different objects that it creates and creates an aggregate health that you can see in the `CommonStatus` that has `healthy` and `errors` (An array of errors from the different objects) fields. For some cases, this is sufficient but we can give more insight into the status of these addons by adding more fields to the `CommonStatus` or using a tool called kstatus.

Kstatus is a project under cli-utils that gives a more verbose information on the status of resources. The library defines five status of a resource: InProgress, Failed, Current, Terminating, NotFound, Unknown and two conditions - Reconciled and Stalled. The kstatus project is divided into two packages: status and a Higher level API `polling` (This computes the status of a set of resources by polling the resource state of the cluster). Integrating this package with the addon-operator would give better information of the status of the set of resources that it creates.

---

*Note: There is still a lot of research to do around cluster addons but I have only included the ones I thought most relevant to prevent a verbose proposal. I aim to continue to research in different areas and discuss with the community. Most of the suggestions here are personal conclusions drawn from research. My expertise is limited and each investigation is open to correction and modification by the mentors.

References

- https://kubernetes.io/docs/concepts/cluster-administration/addons/#infrastructure Official Addon Documentation
- https://github.com/kubernetes-sigs/cluster-addons/issues/39 GSoC Proposal Tracking Issue
- https://github.com/kubernetes/enhancements/blob/master/keps/sig-cluster-lifecycle/addons/0035-20190128-addons-via-operators.md - KEP for Addons via Operators.
- https://docs.google.com/document/d/1tpayzZ4teTKOXNUEsA2HK73eWzz9MhZOFTKYz_rRI_8/edit?usp=sharing - Kubernetes Addon Management
- https://github.com/kubernetes/enhancements/pull/1481 Manifest Bundle proposal

4. Schedule of Deliveries

The main milestones for completing this project is to have all six addon operators fully tested and production-ready.

Dates	**Tasks**
May 4	Community Bonding Period begins
May 4 - June 1	- Work with the community on selecting a final list of addon operators and deciding out the design and implementation details. - Continued study of each of the operators.
June 1	Community Bonding Period ends
June 1 - June 28	- Completing the Kubernetes-dashboard, Kube-proxy, metric-server operators - Writing smoke tests for all created add-ons
June 29 - July 3	First Evaluation
July 3 - July 27	- Completing the Local Node DNS, Calico, and Flannel operators - Writing smoke tests for all created add-ons
July 27 - July 31	Second Evaluation
August 1 - August 30	- Packaging all created addons - Extensive Documentation on how to package addons - Submit final code and project summaries
August 31 - September 7	Final Evaluation

Wishlist

Six addon operators have been set as the goal for this project. If they are completed before the end of the Google Summer of Code. I plan to add more operators to the list and continue to work on them. The addons that I have in mind are:

- Ingress controller
- Canal

Obligations

- I will be able to work full-time (a *minimum* of 40 hours a week) for all three months of the internship.

General Notes

Communication is very vital to the successful completion of GSoC, as well as coordination with the Kubernetes community. The efforts I will be making in this regard are:

- Participate in all cluster-addons bi-weekly meetings, and any other relevant SIG meetings.
- Keeping my mentor(s) informed about my progress on daily tasks.
- Weekly blog posts about project progress, including:
    - Deliverables for the week.
    - Hurdles (if any) encountered while delivering for the week.
    - How the hurdles were surmounted.
- Communicate with the community on Github and Slack.
- Maintain a public Google Doc with daily progress.
- Create a public Github tracker repository with relevant information about project progress.

Post-GSoC

After the Google Summer of Code is over, I plan to continue to contribute to the cluster addons project and create more addon operators!

5. About Me

5.1. Who am I

Name: Somtochi Onyekwere

Email: somtochionyekwere@gmail.com

Github: https://github.com/SomtochiAma

Slack (Kubernetes): SomtochiAma

Twitter: https://twitter.com/AmaOnyekwere

University: Department of Electrical Electronics, The Federal University of Technology, Owerri

Timezone: UTC+01:00 (West Africa Time)

5.2. Why me.

I am greatly intrigued by the deployment of applications using Kubernetes and how addons operators can refine this process. I already have a merged pull request for the Kubernetes dashboard operator and I am constantly testing the existing operators for bugs and extra functionality that it might need.

I've been actively contributing to Kubernetes for the past three months, and to date, I've engaged in testing, code refactoring, documentation, bug fixes and feature implementations in the Kubernetes and cluster addons repository.

I have listed some of these contributions below:

I have been involved in improving and updating the documentation for building an example operator readme in the CoreDNS folder.

- Adds docs on how to run operator locally
- Updates CoreDNS readme for kubebuilder 2.0

I created an addon operator for Kubernetes dashboard to better understand how addon operators work.

- Operator for the Kubernetes dashboard

Merged Pull requests to the Kubernetes Repository

- More unit tests for scheduler #87565
- Adds unit test on Bind extension for the scheduler #87455
- Refactors MakeSecPods function #88218

  This pull request refactors the MakeSecPods function to take in a struct instead of a large number of arguments

Merged pull requests to the Kubernetes Community Repository

- Updates contribex meetings in sigs.yaml #4499

Open pull request in Kubernetes

- Uses ParsePort util function for parsing ports #88216
- Limits request until api is healthy #88172

  This pull request limits the request hitting the apiserver until it becomes healthy for the first time
- Adds additional documentation for job status #87919

Open pull request in the Kubernetes utils repo

- Adds nested traces to the trace library #139

THE END