



Google Summer Of Code 2025 [@kro](#) Project Proposal

By Dhairya Majmudar

Project Name

CLI for kro Resource Management

~CLI Implementation Plan For kro~




What is kro?

-> kro stands for Kube Resource Orchestrator. kro is a Kubernetes-native project that allows to define custom Kubernetes APIs using simple configurations. kro is developed in open source, jointly by the developer teams of Amazon AWS, Google Cloud & Microsoft Azure.

CLI Project setup

-> Currently, kro has a very basic CLI setup present in the file [cmd/kro/main.go](https://github.com/kro/kro/blob/main/cmd/kro/main.go).

The required packages for CLI development are already installed in kro. Here is the list showing them all.

```
kro on  main [] via  v1.24.0  
> go get -u github.com/spf13/cobra  
go get -u k8s.io/client-go  
go get -u k8s.io/apimachinery  
go get -u sigs.k8s.io/yaml
```

-> The project will be started by first making a new directory structure for the kro CLI. Here is the tree view of that. It consists of a new *cli* package into *pkg* folder that will consist of CLI files required to perform validations, code generation, monitoring, etc. All the CLI commands will lie in the *cmd/commands* folder.

```
1  |— cmd  
2  |   |— kro  
3  |       |— commands  
4  |— pkg  
5  |   |— cli  
6  |       |— client  
7  |       |— validator  
8  |       |— generator  
9
```

-> Now comes the main CLI file, which is supposed to be executed. This file will consist of all command groups and global flags. Below a code sample about how it can be implemented.

```

1 func main() {
2     rootCmd := NewRootCommand()
3     if err := rootCmd.Execute(); err != nil {
4         os.Exit(1)
5     }
6 }
7
8 func NewRootCommand() *cobra.Command {
9     cmd := &cobra.Command{
10         Use: "kro",
11         Short: "kro - Kubernetes Resource Orchestrator CLI",
12         Long: `kro CLI helps developers and administrators manage
13 ResourceGraphDefinitions (RGDs) and their instances in Kubernetes clusters.`,
14     }
15
16     // Define global flags
17     kubeConfigPath := ""
18     if home := homedir.HomeDir(); home != "" {
19         kubeConfigPath = filepath.Join(home, ".kube", "config")
20     }
21     cmd.PersistentFlags().String("kubeconfig", kubeConfigPath, "Path to kubeconfig file")
22     cmd.PersistentFlags().String("context", "", "Kubernetes context to use")
23     cmd.PersistentFlags().Bool("verbose", false, "Enable verbose logging")
24
25     // Add command groups
26     commands.AddValidateCommands(cmd)
27     commands.AddGenerateCommands(cmd)
28     commands.AddInstallCommands(cmd)
29     commands.AddPublishCommands(cmd)
30     commands.AddPackageCommands(cmd)
31     commands.AddViewCommands(cmd)
32     commands.AddMonitorCommands(cmd)
33     commands.AddTroubleshootCommands(cmd)
34
35     return cmd
36 }

```

-> This main CLI file will create the **kro** command, which will further have different commands and can be used as **kro validate [...arguments]** OR **kro generate**, etc.

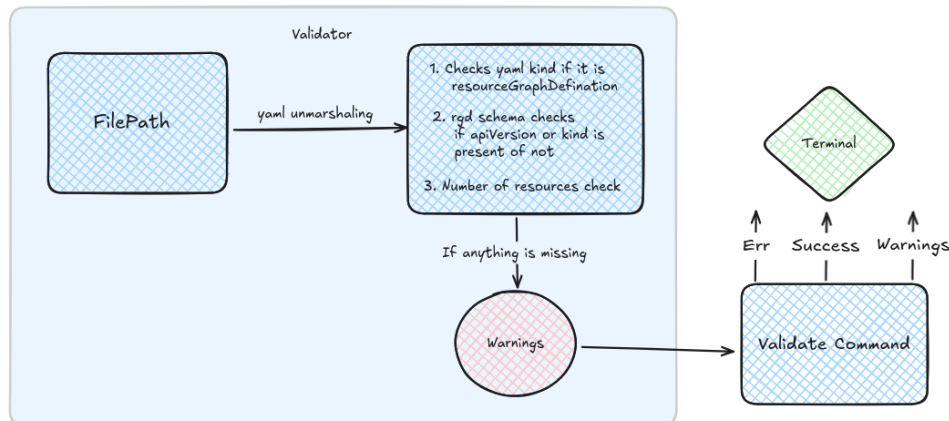
-> After completion of this it comes the part to define several other subcommands which are made to perform tasks like validate, generate, package etc.

-> These commands will be further added into the main CLI file shown above

For Egs:- `commands.AddValidateCommands(cmd)`

Validator Command

-> The validator implementation will take place in *pkg/cli/validator*.

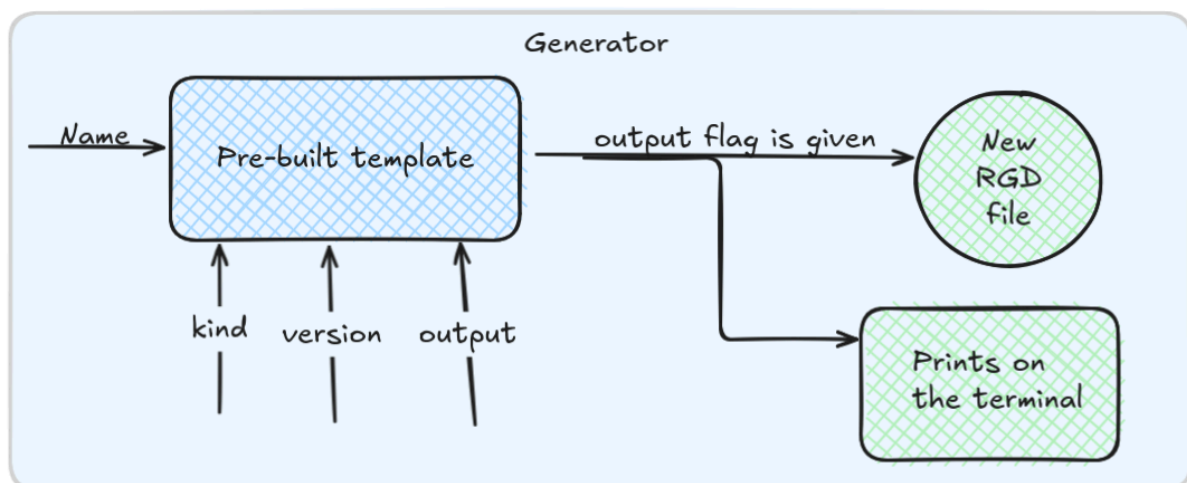


-> Above is an architectural diagram validator command which takes file path as an argument and unmarshals the yaml. The validator consists of several different checks which are as mentioned into the diagram. If any of these validation check fails it returns a warnings these warnings will be further processed by the validate command function and prints, warnings (if any), errors (if any), or validation success message on the terminal.

Usage: kro validate rgd file_name.yaml

Generator Command

-> The validator implementation will take place in *pkg/cli/generator*.



-> Above is an architectural diagram generator command which takes name as an argument for naming the metadata other than these; several optional flags are also passed to the generator command which are:

- 1) kind - used for *spec.schema.kind* (By default is kept empty),
- 2) version - used for *spec.schema.apiVersion* (By default, it is kept as v1alpha1)

3) output - output file path (If it is provided, a new file will be created at that path, or otherwise, the generated schema will be printed in the terminal)

-> The generator uses “**text/template**” which is a built-in package of go lang to generate the RGD file.

Usage: kro generate rgd example_name --kind kind_name -- output example.yaml

View & Monitor Command

-> For making view commands, we need to first create a kro client into the *pkg/cli/client*.

1. kro view rgds: This command will be useful to list all the RGDs of the cluster.
2. kro view rgd rgd_name: This command will take rgd_name as the argument and will be used to view details of a specific RGD.

-> The monitoring commands also use the kro client, but here additionally, we'll watch the changes taking place in the RGD along with also displaying it on the terminal. This command will be using `k8s.io/apimachinery/pkg/apis/meta/v1`, `k8s.io/apimachinery/pkg/runtime/schema`, & `k8s.io/apimachinery/pkg/watch` to watch the changes.

-> Monitoring command will also be capable of showing users the events like modification, deletion, error, and addition.

Usage: kro monitor rgd rgd_name

Install & Troubleshoot Commands

-> Similar to monitoring and view commands, we need to use kro client for the install command. Here, things will be a bit simpler as it will be taking a yaml file input, which is unmarshaled and fed into the kro client.

Usage: kro rgd install file_path

-> In the same fashion, the troubleshoot commands will be created leveraging the kro client. Troubleshoot commands will be useful to get the diagnostic view of instances and pods.

Usage: kro troubleshoot diagnosis rgd_name -> This command will show all the detailed information required for rgd diagnosis which includes status, instances, age, namespaces etc.

-> For getting logs from kro controller pods a separate command will be good to be created, something like *kro troubleshoot log*. This command will list kro controller pods and the pod logs into the terminal.

Package & Publish Commands

-> This command will be useful to package the RGD files from a given directory location. It will take two arguments, directory and the package output file name. Golang's built-in package "*compress/gzip*" can be used to convert output files into a zip folder.

Usage: kro package rgd directory_name output_file

-> kro publish command will be useful to publish the RGD into a given configmap. Two arguments can be taken from the user are directory and config map. This command takes all the yaml file and adds them into configmap data.

Usage: kro publish directory_name configmap_name

Building & Testing CLI

-> Test cases for every command and its related functions will be included to check each and every possible edge case for the cli. For building the cli the command *go build -o bin/kro cmd/kro/main.go* will be wrapped into the make command *make cli* into the makefile.

Documentation

-> The necessary docs updates will be added in the present readme file. Other than these how-to-use cli and about its different command usage will also be reflected on the docs [website](#) of kro. To help users use the cli without any hassle.

~About Me & Tentative Timeline~

Basic Information

Name	Dhairya Majmudar
GitHub username	DhairyaMajmudar
Email	majmudar777@gmail.com
Contact no	+91 9023170440
Linkedin	Dhairya Majmudar
TimeZone	Indian Standard Time (GMT+05:30)
Institute/College	IIIT Kota

A little Background about myself

-> I am Dhairya Majmudar, a third-year student at Indian Institute of Information Technology, Kota. I am an enthusiastic open-source developer with a knack for learning and exploring. I started my programming journey early in the first year of my institute. I have been using open-source software for a long time, even before I knew what it was. Whether it's using VLC media player to watch videos or browsing the web with Mozilla Firefox, I've always tried to contribute to open-source software in any way I can.

-> Over time, this passion turned into active contributions when I started contributing to open source, full time along with managing my institute studies. I actively take part in open-source mentorship programs because as a student, these mentorship programs give us a great opportunity to learn from people working in the actual wild.

Prior open source experience

-> The current year's spring started with contributing to **CNCF Headlamp**, where I got into the **LFX mentorship program**. My project includes instrumenting the project Headlamp using **OpenTelemetry**.

-> Last year during spring, I got the opportunity to contribute to Linux's KDE software under their mentorship program **Season of KDE (SoK'24)**. Where I collaborated with my mentor to enhance the mathematical properties and view of Cantor CAS project.

-> During the summer, I started contributing to the JSON Schema, where they later selected me as **Google Season of Docs (GSoD'24)** mentee. During my mentorship, I

collaborated with my mentor and technical writers to improve and implement new UI-related features to the JSON Schema website, built on the top of the Next.js + TypeScript stack. I also helped to improve the JSON Schema's DevOps CI/CD pipeline and wrote unit. and e2e tests. Further on, after completion of my mentorship, I was promoted as one of the **core JSON Schema maintainers**.

-> **Gitlab's April hackathon** is organised by Gitlab for contributors to contribute to their core codebase. My DevOps and frontend contributions to the Gitlab core codebase made me be in the top 10 winners of the hackathon. Other than these, I've also contributed code to several CNCF projects like **Headlamp, Vitess, and Keptn**.

-> Link to all my notable open source contributions: [Github Gist Link](#).

kro Contributions

	Pull Request Title	PR Number	Status
1.	Tests: adds unit tests for `pkg/graph/crd`	#402	Merged
2.	Cleanup: Removes unused function GetResourceTopLevelFieldNames	#393	Merged
3.	Fix: makefile: deploy-kind command	#400	Merged
4.	Tests: pkg dynamiccontroller: increases test coverage	#414	Open
5.	pkg simpleschema: adds SimpleSchema enum support	#424	Merged
6.	Tests: adds tests for cel/environment.go	#448	Mereged
7.	Enhance error handling for MarkerTypeMinimum & MarkerTypeMaximum	#468	Merged

Tentative Timeline

Time Frame	Duration	Tasks
8 May - 1 June	3 Weeks	<input type="checkbox"/> Connect with the kro team and mentors and get on-boarded. <input type="checkbox"/> Discuss the proposal with the team and look for possible improvements and changes.

		<input type="checkbox"/> Kickstart the project.
2 June - 8 June	1 Week	<input type="checkbox"/> Start developing the CLI main file and kro client. <input type="checkbox"/> Writing corresponding test cases.
9 June - 25 June	2.5 Weeks	<input type="checkbox"/> Start developing the validator and the validation command for the same. <input type="checkbox"/> Writing corresponding test cases and making it sync with the main CLI entry point.
26 June - 12 July	2.5 Weeks	<input type="checkbox"/> Development of generator cli command and its corresponding test cases. <input type="checkbox"/> Taking reviews from mentors and community and resolving the open PRs. <input type="checkbox"/> Start preparing for Mid Term Evaluation.
14 July - 18 July	½ Week	<input type="checkbox"/> Submitting the Mid term evaluation. <input type="checkbox"/> Writing and publishing blog posts for the work done till now.
21 July - 28 Aug	1 Week	<input type="checkbox"/> Start focusing on kro client development, which will be further used in other commands. <input type="checkbox"/> After the completion of kro client development of monitoring and viewing commands will be done
29 Aug - 12 Aug	2 Weeks	<input type="checkbox"/> During this time period the other cli commands like packaging, install, and publish will be developed. <input type="checkbox"/> Alongside it, their corresponding test cases will also be added
14 Aug - 20 Aug	1 Week	<input type="checkbox"/> Updating docs website and readme file to show how to use the cli tools. <input type="checkbox"/> Final project completion.
21 Aug - 1 Sept	1.5 Weeks	<input type="checkbox"/> Buffer week for final iterations and testing. <input type="checkbox"/> Writing blog posts about the work done

		and experience gained from the project. <input type="checkbox"/> Final project report submission
--	--	---

-> In addition to that, my work each week would be a blend of the following:

1. Breaking down issues into smaller modular tasks.
2. Interacting and getting feedback from mentors.
3. Team meetings with mentors and other community members.

Post GSoC

-> Even after GSoC ends, I plan to continue contributing to kro and strengthening my bond with this amazing community. For the next upcoming major releases of kro, there's a [milestones section in github](#). I would love to contribute further to developing those features.

References

1. <https://cobra.dev/>
2. <https://github.com/spf13/cobra>
3. Argo CD CLI: <https://github.com/argoproj/argo-cd/blob/master/util/cli/cli.go>
4. <https://dev.to/aurelievache/learning-go-by-examples-part-3-create-a-cli-app-in-go-1h43>

Google Doc Link:  [Copy of kro Opentelemetry Instrumentation](#)