Security Validator for Jenkins Kubernetes Operator

Google Summer of Code Program 2021 Project Proposal

Pulkit Sharma
pulkit.sharma.mst17@iitbhu.ac.in
Github handle

Project Abstract. Some of the plugins have security warnings that are not properly presented to the end-users. This may result in introducing security risk in existing CI/CD infrastructure.
To ensure that the given resource being deployed does not have plugins containing security issues, add a validation step using Kubernetes Admission Controller.

Project Description. Jenkins instances on a Kubernetes cluster running the operator are deployed using declarative YAML configuration files; hence some of the plugins declared in these files may contain security warnings.
So there is no way for the user to know other than manually checking for each on the site. Hence to solve this problem, I am going to implement a validation webhook that checks whether the given set of plugins have security issues or not. If it does, then display warnings to the user and ask his/her permission to continue the deployment or not.

This will prevent some serious security hazards before happening. I am interested in this project as I am interested in learning about Kubernetes and operators in general. This project also gave me the opportunity to learn Jenkins and CI/CD Pipelines.

Project Deliverables

- May 17: I will discuss my approach with community members. I would have set up my development environment and installed the required tools.

- June 4 (coding begins) - I would have a clear roadmap and set strict deadlines for my deliverables. I will start implementing my project.

- July 12 - July 16 (Phase 1 Evaluation)
  - A new validation webhook is added to the operator and should be able to receive validation requests for Jenkins resources.
  - The validation function should be able to fetch security warnings from the server.

- August 16 -August 23(Final Evaluation)
  - Webhook should be to validate Jenkins resources.
  - Unit and e2e tests are written
  - Completed Documentation

- August 31(Final Results Announced) -
  - Update documentation and clean code.
  - Write a blog post about my GSOC experience.

- Post GSoC - The operator has a lot of issues that need to resolve. I also have an idea of extending the implemented security validator to solve some other issues.

March 29 - April 13 (Application Period)

- I will do the required literature survey
- I will work on my proposal and make sure that it clearly conveys what I am trying to implement.
- I will study the Jenkins operator codebase.

April 14 - May 16 (Acceptance Waiting Period)

- I will work on trying to develop a proof of concept.
- I will study the codebase more, increase my engagement in the community, discuss potential ideas, solve issues and report bugs.

May 17 - June 3 (Community Bonding Period)

- I will discuss my approach with the community members and will try to get their feedback.
- Based on their feedback, I will make necessary adjustments in my approach.
- I will make a thorough roadmap and strict deadlines for my deliverables.

June 7 - July 12 (Coding period 1)

Week 1 June 7 -June 13

- I will scaffold a new validation webhook
- I will update markers and generate manifests for webhook configuration and service and update them to integrate them with the project.
- I will add a new spec to Jenkins custom resource to enable/disable validation checks.

Week 2 June 14 - June 20

- I will generate resource definitions for cert-manager writing YAML files for Issuer and certificate.
- I will update the Makefiles to build and run the operator locally.
- I will build the container image locally and deploy it locally to check if it is getting the validation request from the API server.
- I will start implementing the validator interface.

Week 3 & 4 June 21 - July 4

- I will implement the fetching of security warnings from the Plugin site.
- I will start implementing the validation logic for creating and updating objects.

Week 5 July 5 - July 11

- I will finish implementing the validator interface.
- I will structure the code for best practices.

July 17 - August 23 (Coding period 2)

Week 6 July 17 - July 23

- I will focus on improving the code.
- I will be studying godocs of testing framework

- I will start writing unit tests for internal functions.

Week 7 July 23 - July 29

- I will finish writing unit tests for internal functions.
- I will start writing code for e2e tests.

Week 8 July 30 - Aug 5

- I will finish writing e2e test cases.
- I will take care to extract out repeated code and reduce boilerplate code.

Week 9 Aug 6 - Aug 12

- I will start writing documentation.
- For best development practices, I'll also run the code through goreportcard , aiming for A+ code rating.
- I will demonstrate the working of the webhook to my mentors and will try to get their feedback.
- I will try other solutions than cert-manager for managing tls certificates.
- If things worked well, I will plan on extending the validator for other validations.

Week 10 Aug 13 - Aug 16

- I will finish writing documentation.
- I will work on the feedback provided by community members.

Final Week Aug 17 - Aug 23

- For the final week, I'll focus on the reviews given by my mentors and discuss with the community members on what more things can be added to the validating webhook.
- I'll clean up the documentation and code.
- I'll write a blog post on my GSoC experience.

Detailed Plan and Implementation

Webhook Scaffolding

I will use operator-SDK to scaffold a new validation webhook.

```
$ operator-sdk create webhook --programmatic-validation --group jenkins.io
--version v1alpha2 --verbose --kind Jenkins
```

I will only have to implement the Validator interface. Operator-SDK will create a new webhook and take care of adding it to the manager and creating handlers.
It also creates functions for writing tests.

After executing the command, two files are generated in api/v1alpha2 jenkins - webhook.go and webhook_suite_test.go

Managing certificates

Webhooks communicate to the API server over HTTPS and use TLS. To serve webhooks, we need to have certificates. By default, the webhook looks in /tmp/k8s-webhook-server/serving-certs for certificates.

Proposed Solution -

I will be using **cert-manager** to manage certificates. It will ensure certificates are valid and up to date and attempt to renew certificates at a configured time before expiry.

CA Injector can be used to configure CA certificates for webhooks. It populates the caBundle field using CA data from a cert-manager Certificate.
By Populating the caBundle, the webhook is able to recognize the certifying authority.

A self-signed issuer with a private can generate signed certificates and provide automatic certificate renewal and management.

Other Alternatives -

1 Self-signed certificates with infinite long expiry date -

Creating a certificate and a private key with openssl with a very long expiration date so that there is no need for renewal.
Populate the cabundle field and place the tls certificates in the required directory using an init container or a shell script.

Pros -
1 Minimalist approach, adds very little changes to existing infrastructure.
2 No external dependencies.

Cons -
1 Renewing certificates frequently is considered as a good security practice.

2 Cert-controller

It is a go package specifically designed for webhooks. It scaffolds certificates for webhooks and adds them to a secret and populates the cabundle field.
It also watches the certificates and makes sure if they are okay and regenerate them if they are not.

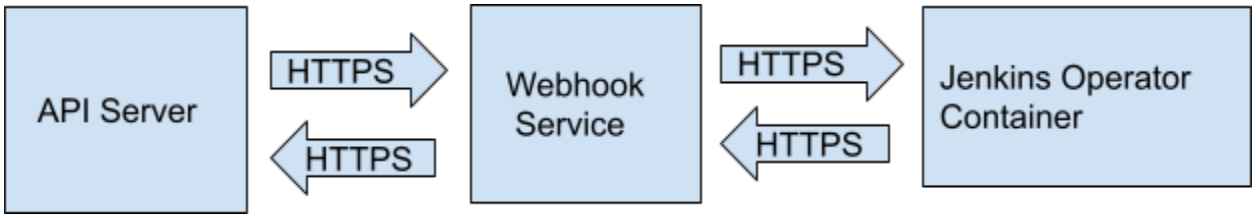
Pros -
1 Automatic certificate provisioning.
2 Very small in comparison to cert-manager, the whole package consists of ~700 lines of code, also will have very less memory consumption and cpu usage.

Cons -

1 Still have to rely on a third party library.
2 Not very well maintained

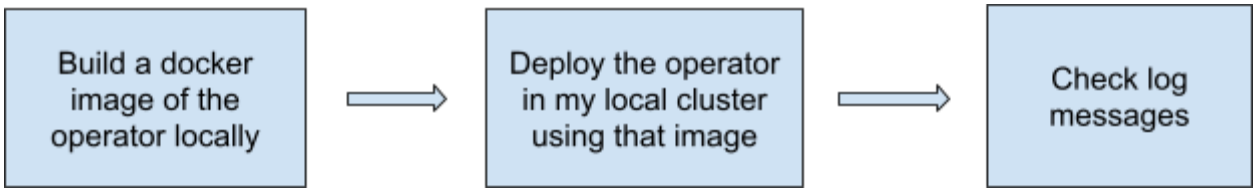
Development workflow

In Kubernetes, admission webhooks works the following way -



So, the service exposes the operator to the API. Hence, the operator cannot be deployed locally by using the make run command for local development purposes.

So, I will make a docker image of the operator using the make container-runtime-build command and deploy it locally and check logs messages for development purposes



Issues in local development -

Because the webhook is configured to run in such a manner, it cannot be tested locally using make run.

Also, we can't run the operator locally because after running the manager as executable it will check for certificates in /tmp/k8s-webhook-server/serving-certs and if not found it will break.

Proposed solution -

Create a directory and place files by the name of tls.crt and tls.key in them and define the certDir key while deployment.

Make a new target : run-with-webhook

Every time build a docker image locally and then run the image.

Deploying the operator

The recommended way is to use the make deploy command, but this is not used by the community and is outdated and full of bugs.

Although the operator uses markers to scaffold manifests and uses the make manifests command, the operator generates the webhook configuration(config/webhook/manifests.yaml) and specifications for generating webhook service(config/webhook/services.yaml) based on markers.
But these files have to be updated so that they integrate with the operator.

Also, Jenkins-operator Deployment manifests(config/manager/manager.yaml) needs to be updated to accommodate the generated Webhook service and mount a volume for certificates.

I will update these files along with config/all_in_one_v1aplha2.yaml, as this is used for deployment purposes.

Implementing Validator Interface

Kubebuilder scaffolds webhook functions and binds the webhook with the manager.

If the webhook. Validator interface is implemented, a webhook will automatically be served that calls the validation.

In this case, the webhook is only required to validate for updating and creation of Jenkins deployments.

So, I will implement validation logic for ValidateUpdate() and ValidateCreate() .

And since this logic for their validation is identically the same, I am going to implement a single function ValidateJenkins(), to validate both of them.

```
// TODO(user): change verbs to "verbs=create;update;delete" if you want to enable deletion validation.
// +kubebuilder:webhook:path=/validate-jenkins-io-jenkins-io-v1alpha2-jenkins,mutating=false,failurePolicy=fail,sideEffects=None,groups=jenkins.io.jenkins.io,r

var _ webhook.Validator = &Jenkins{}

// ValidateCreate implements webhook.Validator so a webhook will be registered for the type
func (r *Jenkins) ValidateCreate() error {
    jenkinslog.Info("validate create", "name", r.Name)

    // TODO(user): fill in your validation logic upon object creation.
    return nil
}

// ValidateUpdate implements webhook.Validator so a webhook will be registered for the type
func (r *Jenkins) ValidateUpdate(old runtime.Object) error {
    jenkinslog.Info("validate update", "name", r.Name)

    // TODO(user): fill in your validation logic upon object update.
    return nil
}
```

I can easily extract the Plugin Name and version.

Using the plugin name, security warnings can easily be fetched using Jenkins [Plugin site API]

I will send a get request to https://plugins.jenkins.io/api/plugin/<name-of-plugin> to receive an HTTP response containing all information about the particular plugin like build date, its dependencies, list of maintainers, etc.

For a particular plugin, like google-login it can be implemented in go something like this -

```go
client := &http.Client{}
req, err := http.NewRequest("GET", "https://plugins.jenkins.io/api/plugin/google-login/", nil)
if err != nil {
    fmt.Print(err.Error())
}
req.Header.Add("Accept", "application/json")
req.Header.Add("Content-Type", "application/json")
resp, err := client.Do(req)
if err != nil {
    fmt.Print(err.Error())
}
fmt.Print(resp.Body)
defer resp.Body.Close()
bodyBytes, err := ioutil.ReadAll(resp.Body)
if err != nil {
    fmt.Print(err.Error())
}
securityWarnings := Warnings{}
jsonErr := json.Unmarshal(bodyBytes, &securityWarnings)
if jsonErr != nil {
    log.Fatal(jsonErr)
}
```

To extract security warnings from all the JSON encoded information present in the response body, I will implement a custom data structure.
With the help of the encoding/json package and this data structure, security warnings can easily be extracted.

```go
type Warnings struct {
    SecurityWarnings []Warning `json:"securityWarnings"`
}

type Warning struct {
    Versions []Version `json:"versions"`
    Id       string    `json:"id"`
    Message  string    `json:"message"`
    Url      string    `json:"url"`
    Active   bool      `json:"active"`
}
type Version struct {
    FirstVersion string `json:"firstVersion"`
    LastVersion  string `json:"lastVersion"`
}
```

Validation Logic

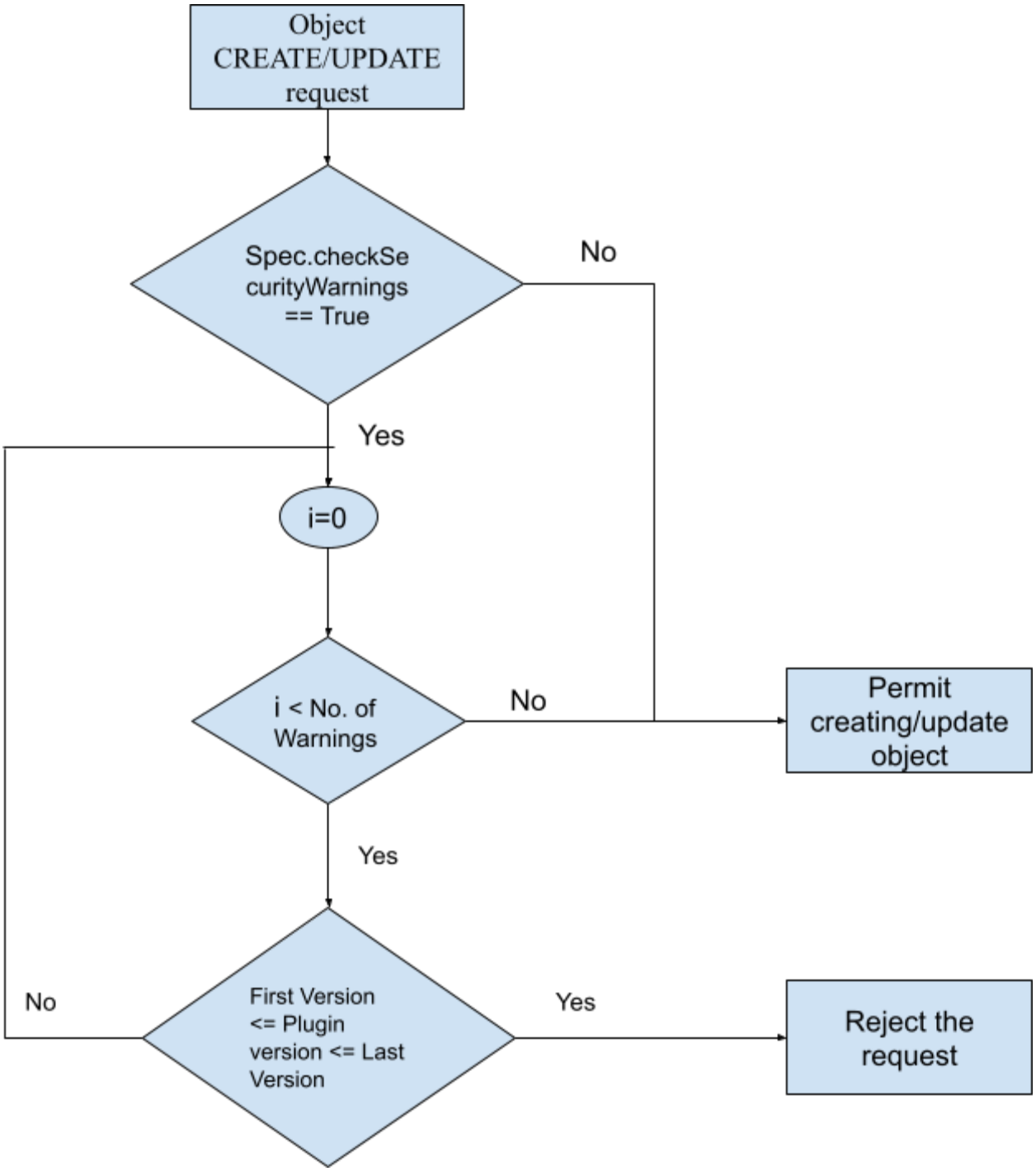
I propose to add a new field in the Jenkins spec which takes a boolean input and will determine if we want to go through the whole validation process.

```go
type JenkinsSpec struct {
    // Master represents Jenkins master pod properties and Jenkins plugins.
    // Every single change here requires a pod restart.
    Master JenkinsMaster `json:"master"`

    // SeedJobs defines list of Jenkins Seed Job configurations
    // More info: https://jenkinsci.github.io/kubernetes-operator/docs/getting-started/latest/configuration#confi
    // +optional
    SeedJobs []SeedJob `json:"seedJobs,omitempty"`

    CheckSecurityWarnings bool `json:"checksecuritywarnings,omitempty"`
```

```
            type: object
          checksecuritywarnings:
            type: boolean
          configurationAsCode:
            description: ConfigurationAsCode defines configuration of Jenkins
              customization via Configuration as Code Jenkins plugin
```

If the plugin version lies between the FirstVersion and LastVersion, the webhook will reject the request creating/updating Jenkins custom resource.

The whole thing can be summarized in the following block diagram -



Testing

It's vital to write automated tests at different levels:
- Unit Tests - To test each component or unit of code individually.
- End-to-End Tests - To test the behavior of the webhook, just as it would behave when running on a live cluster.
Internal code, without dependencies on K8s services and context, will be ideal for Unit Testing. These tests will run fast and can be written as standard go tests. However, the most crucial and yet slow tests will be E2E tests since they need to run on the cluster. It'll be simpler and efficient to use the testing framework provided by operator-sdk.

Although Operator-sdk scaffolds boilerplate code for getting environment etc. for writing e2e tests when a new webhook is created, the boilerplate code is also available in test/e2e/test_suite.go. After checking out other e2e tests, I found out that the community uses Expect() function in the Gomega package for testing purposes.

The test-specific code will involve testing the following scenarios (may consider adding more later)

1. Creating/Updating Jenkins Resource with Plugins having no security warnings - the Operator should succeed in creating/updating the resource.
2. Creating/Updating Jenkins Resource with Plugins having security warnings - The operator should warn the user about security warnings and ask if he wishes to continue the deployment.
   There are two scenarios in this case -
   1. The user wants to continue the deployment - The operator should create/update the resource.
   2. User does not want to continue- Operator should reject the request.

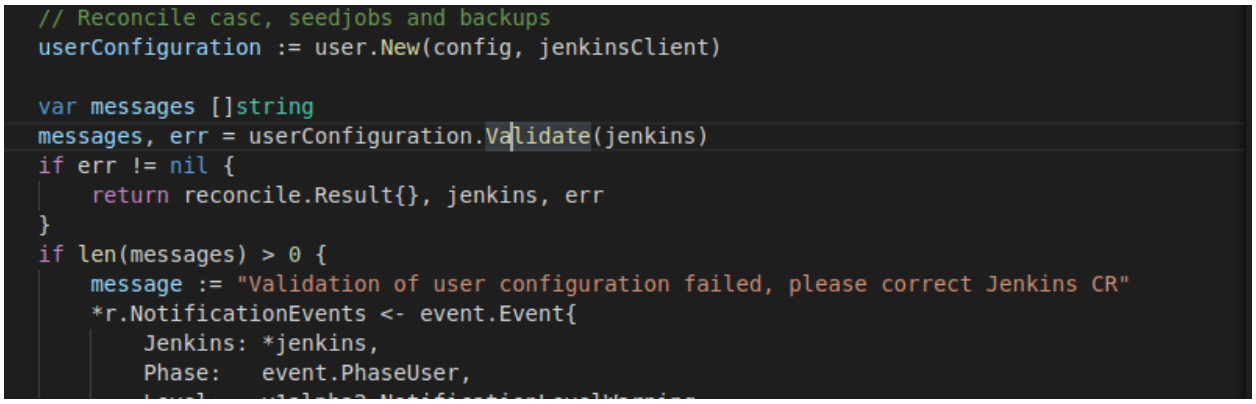
Future Improvements

Validating Required Core Version of the Plugin

I have planned to extend this validating webhook to solve a problem that can be solved in a similar fashion.
I observed that the operator fails to create a pod when the custom resource tries to install a plugin that requires a Jenkins core version higher than that of the master container. It should give a warning about this to the user before creating the object. Like security warnings, the required Jenkins version can also be fetched over the REST API.
I have raised this issue and discussed it with community members, and I am planning to implement it as an extension of this project.

Migrating Validating Jenkins custom resource spec section to webhook

Validation for Jenkins custom resource specs occur at reconciliation loop.



```
// Reconcile casc, seedjobs and backups
userConfiguration := user.New(config, jenkinsClient)

var messages []string
messages, err = userConfiguration.Validate(jenkins)
if err != nil {
    return reconcile.Result{}, jenkins, err
}
if len(messages) > 0 {
    message := "Validation of user configuration failed, please correct Jenkins CR"
    *r.NotificationEvents <- event.Event{
        Jenkins: *jenkins,
        Phase:   event.PhaseUser,
```

I was thinking of migrating this validation to the validation webhook. I could either extend this already implemented webhook or add another webhook since validation webhooks execute in parallel and this would speed up things.

All the logic for validating custom resource spec has already been implemented in package user so it won't be too tiresome to do.

Implementing a defaulting webhook for setting up default values -

Default values for Jenkins CR are also being set in the reconciliation loop.

```
func (r *JenkinsReconciler) setDefaults(jenkins *v1alpha2.Jenkins) (requeue bool, err error) {
    changed := false
    logger := logx.WithValues("cr", jenkins.Name)

    var jenkinsContainer v1alpha2.Container
    if len(jenkins.Spec.Master.Containers) == 0 {
        changed = true
        jenkinsContainer = v1alpha2.Container{Name: resources.JenkinsMasterContainerName}
    } else {
        if jenkins.Spec.Master.Containers[0].Name != resources.JenkinsMasterContainerName {
            return false, errors.Errorf("first container in spec.master.containers must be Jenki
        }
        jenkinsContainer = jenkins.Spec.Master.Containers[0]
    }
```

I propose to implement a defaulting webhook as it would the preferred way of setting up the default values.

If some other problem comes up that can be solved via webhooks , I would certainly like to work on it.

Continued Involvement

I would like to continue to contribute to this community as Devops is my field of interest. I would hold myself accountable for maintaining and upgrading the webhook. I would certainly like to explore other Jenkins repositories as well; some of the plugin ideas like the machine learning plugin, etc. seems interesting.
Also, there are a lot of problems in this operator that need to be solved.

Major Challenges foreseen

There are things customized according to this project and for proper integration, one should have knowledge of the codebase and how things bind together.

Relevant Background Experience

I have been working on this project for about a month and learning DevOps and Kubernetes for about two months. I have studied the codebase and the documentation extensively. Currently, I am working on developing a proof of

concept. I have done the required literature survey and have planned for implementation in detail.

Personal

I'm Pulkit Sharma, an undergraduate at the Indian Institute of Technology (BHU), Varanasi. I mainly started with coding two years back and have grown to like open source. I've basic knowledge of Golang but insufficient experience in it. I am an eager learner and ready to add more skills to my toolbox.

I have been part of quite a good number of projects, and I have developed a flair for tackling challenges.

I am attaching my CV in the links section below to look at my projects.

Availability and commitments

I don't have any commitments and will be available anytime between 1 p.m IST to 2 a.m IST on weekdays. On weekends, I would love to spend time communicating with the team to learn from them while working on whatever issues occur at that time.

I'm flexible with my schedule and have inculcated the habit of working at night, so time zone difference shouldn't be an issue.
Due to the COVID-19 situation, our college's situation is uncertain yet but I'll keep my mentor updated with any new happenings or if there's a conflict of this project with any changes in my academic schedule. I'll also responsibly keep my mentor updated in case of any emergency that occurs with suitable details

Free Software Experience/Contributions :

PR's merged

https://github.com/TheAlgorithms/Python/pull/4035

https://github.com/TheAlgorithms/Python/pull/3599

https://github.com/kubernetes/community/pull/5526

https://github.com/kubeedge/examples/pull/95

https://github.com/div-bargali/Data-Structures-and-Algorithms/pull/704

Language Skill Set

Python
C++
Javascript
Go
CUDA
Java

Reference Links and Web URLs:

Github
Linkedin
Twitter
Resume