ON-PREMISE APPLICATION DEPLOYMENT PROJECT REPORT

Ramprasath A – ramprasathasokan@gmail.com

This project aims at solving on-premise application deployments for the enterprise clients of a leading tech corp. The requirements are managing multiple microservices running across different applications and how to deploy and manage microservices individually. As an example we are going to deploy a containerized web application namely Parse Server Example.

In order to further increase the ease of operations we are going to take into account the following challenges faced in deployments too:

1. Ease of clustered enterprise level deployments
2. Incremental remotely triggered application updates
3. Easy remote debugging
4. Health Alerts and Monitoring
5. Application Security (with source code protection)
6. Disaster management

Before diving into the installation instructions, let us see what are the various tools used and what requirements does it satisfies.

Tools Used

1. Google Cloud Platform – for infrastructure requirements
2. Docker – for setting up Rancher and Harness delegate
3. Rancher – for managing Kubernetes cluster
4. GKE (Google Kubernetes Engine) – for provisioning Kubernetes cluster
5. Kubernetes – for deploying our parser server application
6. Prometheus and Grafana – for monitoring
7. ELK stack (Elasticsearch, Logstash, Kibana) – for log management
8. Slack – for notification of alerts
9. Harness – for CD pipeline
10. Helm – for providing helm charts for parse server application
11. Velero – for backing up and restoring data in the event of a disaster

Assumptions

I assumed it to be an actual production environment but in the interest of time, I used docker containers for running Rancher and Harness delegate. Also, I used a managed ELK stack by Bitnami Launchpad which may or may not go along with the privacy guidelines of the organization. But as far as Harness is considered, there is an option to opt for a self-hosted version.

So, essentially, I chose tools and services which can be easily run on premise so that we can ensure more security and privacy of data. Also, I assumed GCP as on-premise infrastructure.

Solving the challenges faced in deployments using the above tools

1. Ease of clustered enterprise level deployments

We are using Rancher which provides a rich GUI to provision, monitor, alert, notify, log a Kubernetes cluster. Also, we can rollback deployments and backup etcd data by taking snapshots.

2. Incremental remotely triggered application updates

In Harness, under workflow creation we can choose rolling update strategy i.e. incremental updates strategy and we can trigger the updates remotely i.e. instead of directly interacting with the cluster through Harness.

3. Ease remote debugging

We are using the ELK Stack i.e. Elasticsearch, Logstash and Kibana Stack for log management. Using the ELK stack we can retrieve a wide range of logs along with their timestamp to easily debug errors (if any) remotely. Also, updates regarding health of a node will be notified via Slack channel from Rancher which will further ease remote debugging. Also, we can use the insights from Prometheus and Grafana

4. Health Alerts and Monitoring

We are using Prometheus and Grafana for monitoring our Kubernetes cluster and Rancher has a set of Health Alert groups which can be configured with Slack for notification if required.

5. Application Security

Rancher and Harness both support enforcing of RBAC (Rule Based Access Control) and we are leveraging it to ensure security of our management and deployment. If we happen to use NGINX-Ingress as our load balancer we can enable OWASP ModSecurity WAF rules just by adding an annotation: nginx.ingress.kubernetes.io/enable-modsecurity: true. We can use code obfuscation tools to obfuscate our source code so it will be harder to reverse engineer to obtain the source code.

6. Disaster Management

Disaster management deals with unexpected failure. A standby Kubernetes cluster and an up-to-date standby database with the same version of the application deployed as in the currently running cluster will help in ensuring high availability and thereby reducing the downtime in the event of a disaster.

In the case of backup, the first and foremost priority for backing up should be given to the database of the MongoDB used by our parse server. Then, we should take backups of etcd. We should backup these data regularly once every hour or once a day based on client

requirements. Most importantly we should ensure to store the backups in a highly available storage like AWS S3. In order to achieve this, we can use Velero.

After a disaster, we can recover from a disaster by restoring the latest backup of our database and application state.

Installation / Usage Instructions

YouTube link: https://youtu.be/FMxPRegLg-A

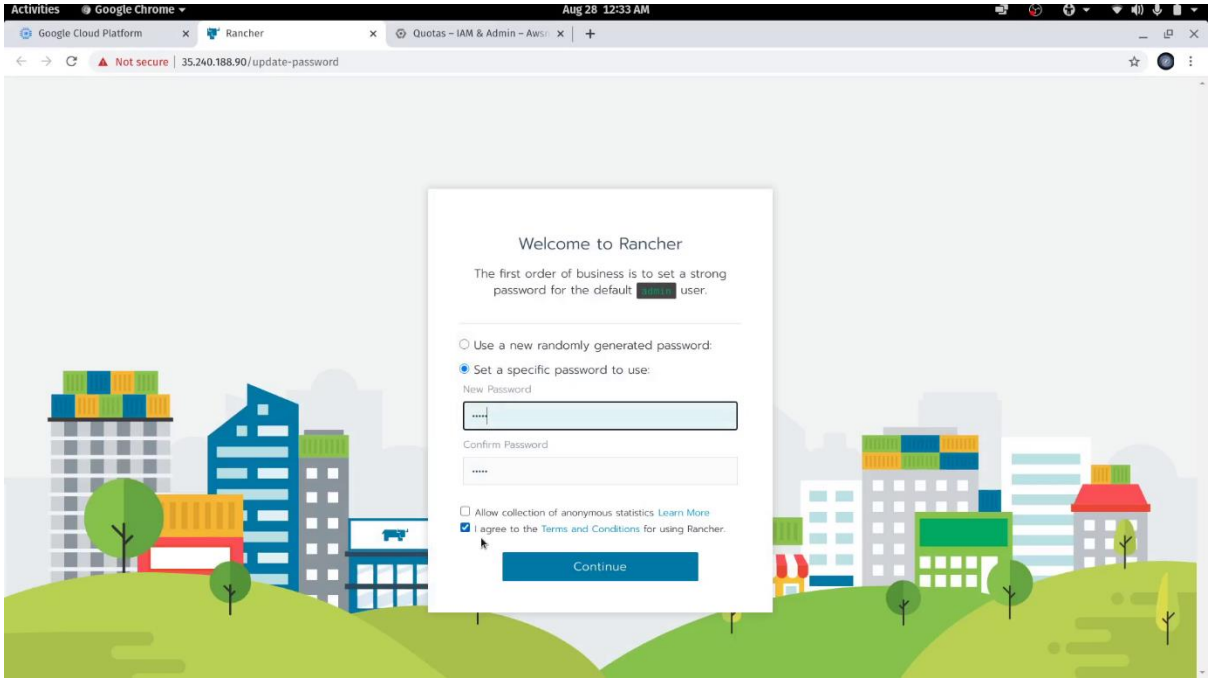
GitHub link: https://github.com/ramprasathasokan/On-Premise-Application-Deployment

Note: Before setting up VM instances we need to create two different projects in Google Cloud Platform because each region has a quota for vCPU about 8 vCPUs. In one project we will setup Rancher, ELK Stack and Harness delegate and we will use the other project solely for Kubernetes cluster.

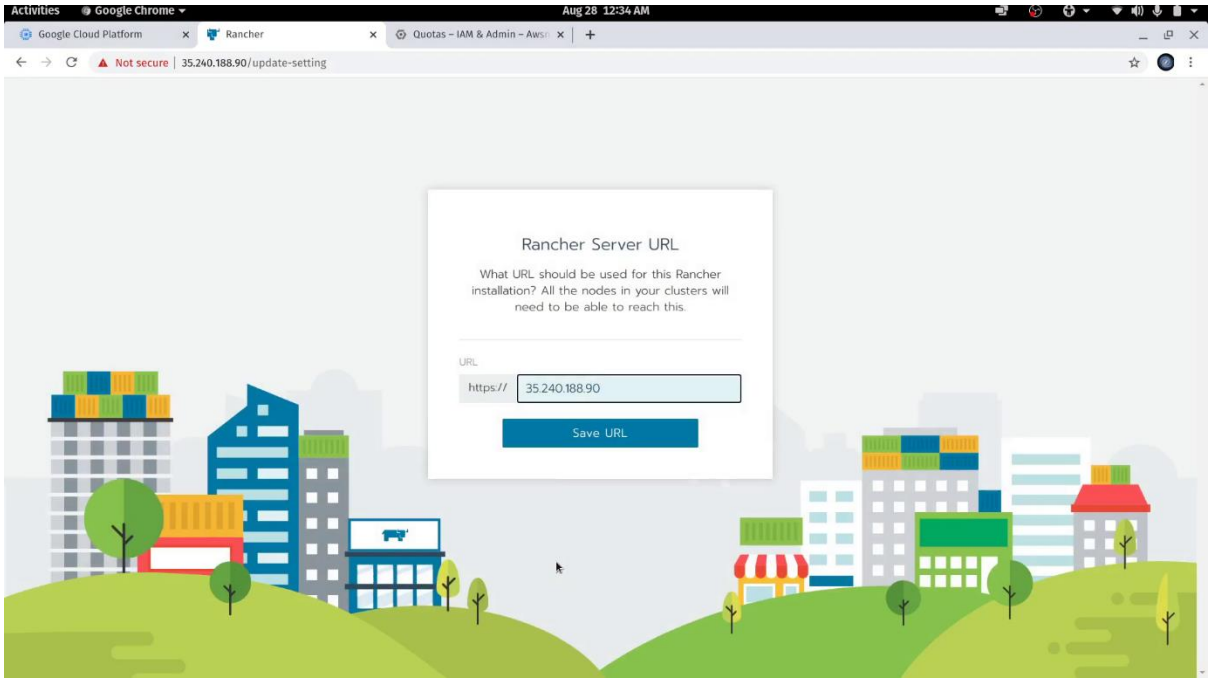
1. We need to setup our infrastructure, in this case, VMs. We are going to create 2 VM instance initially, one for Rancher and another one for Harness delegate. We will see about Harness delegate while setting up Harness CD pipeline. The following are specifications of each VM: n1-standard, 1vCPU, 3.75GB RAM, 50 GB Storage, Ubuntu 18.04 LTS. Select a desirable region and zone as per requirements. Enable both allow HTTP/HTTPS traffic.
2. Before clicking create instance we need to add our SSH credentials under Security tab of the instance creation page.
   - To generate SSH key: `ssh-keygen -t ecdsa -C username`
   - To copy securely: `xclip -sel clip < ~/.ssh/id_ecdsa.pub`
3. Now create instance and make sure to create another instance. Tip: we can use templates to create VMs.
4. SSH into both VMs using: `ssh -i ~/.ssh/id_ecdsa.pub username@external-ip`
5. We need to run: `sudo apt update && sudo apt upgrade -y` in both VMs and then perform a reboot: `sudo reboot`.
6. Then we need to install Docker-CE (Docker Engine) and verify installation in both VMs by running the following commands one by one:
   - `sudo apt update`
   - `sudo apt install apt-transport-https ca-certificates curl software-properties-common`
   - `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`
   - `sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"`
   - `sudo apt update`
   - `apt-cache policy docker-ce`
   - `sudo apt install docker-ce`
   - `sudo systemctl status docker`
   - `sudo usermod -aG docker ${USER}`

- *sudo su - ${USER}*
- *id -nG*
- *docker info*

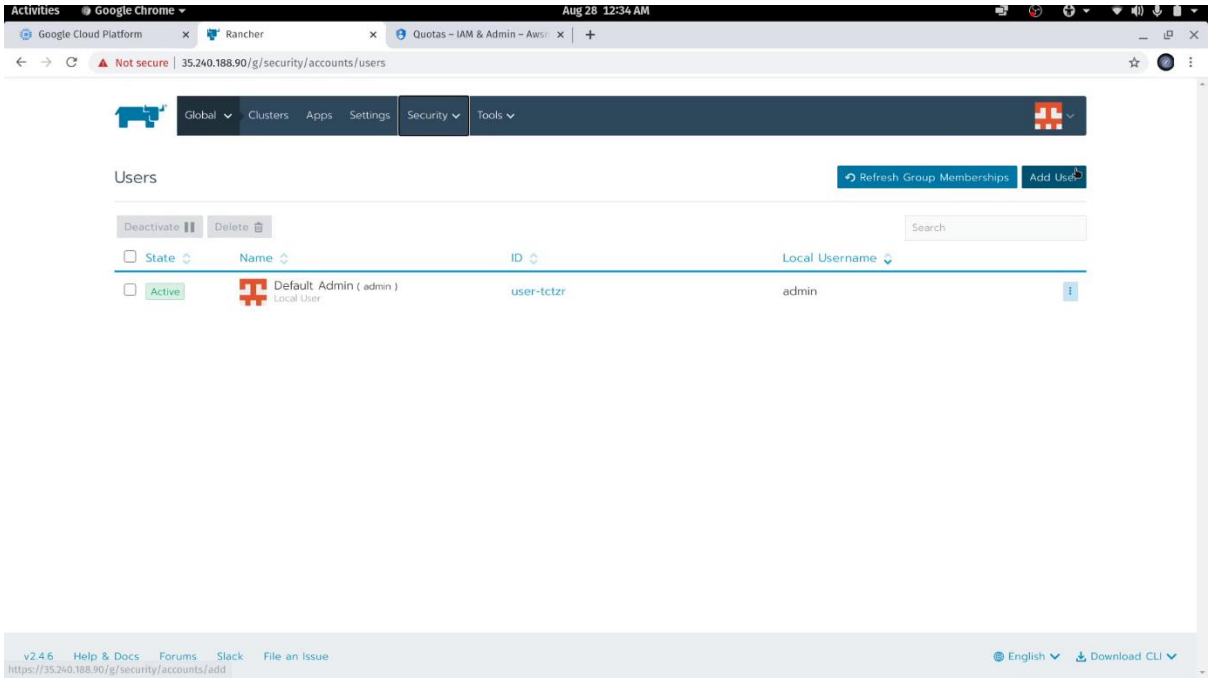
7. Now we need to install Rancher in its respective VM by running the following command:
   - *sudo docker run -d --restart=unless-stopped -p 80:80 -p 443:443 -v ~/.rancher-data:/var/lib/rancher rancher/rancher*
   - Note: We are mounting volume of Rancher to make the data persistent while performing reboot.
8. Then, we can access rancher using the external-ip of the instance where Rancher is installed. Then setup admin password, agree to T&C and continue.



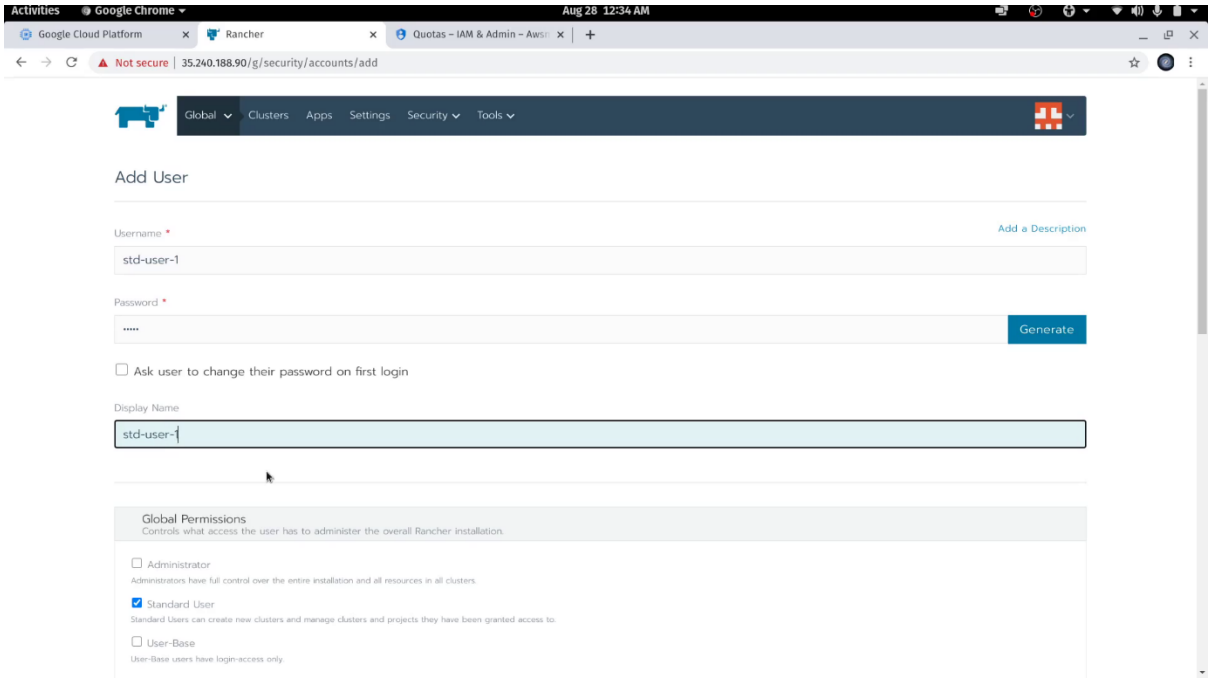
9. Save Rancher Server URL



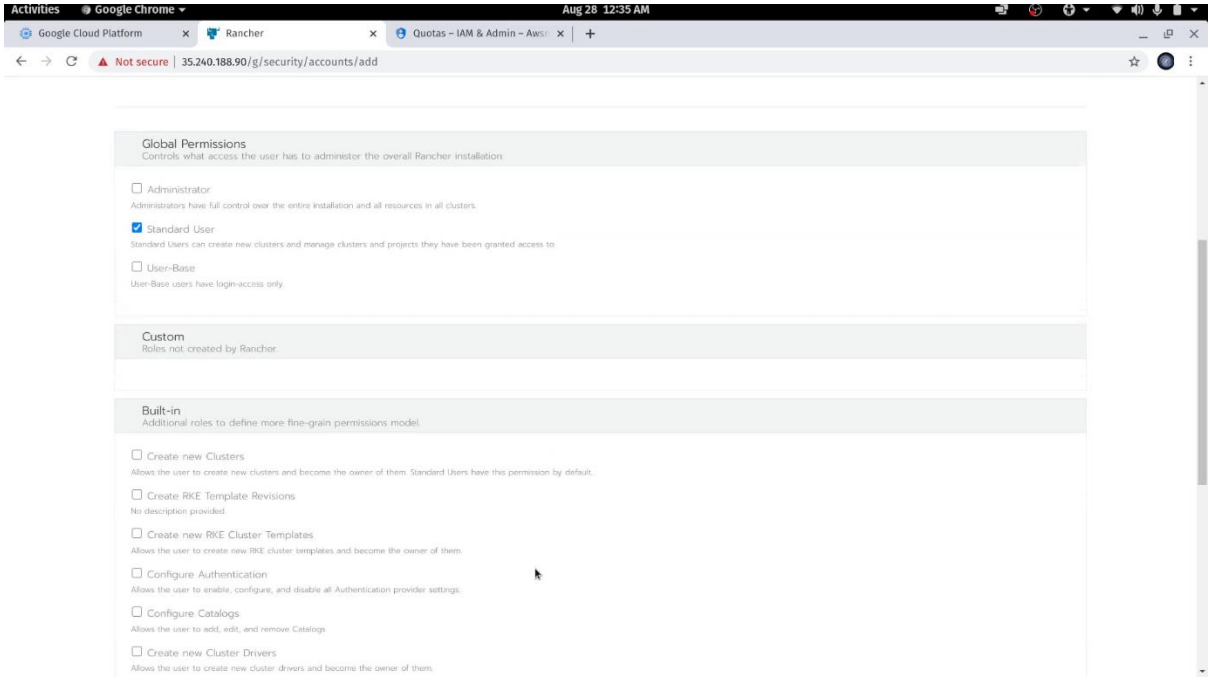
10. Now before we go ahead and create a cluster, let's see how to add a user and implement RBAC (Role Based Access Control) with Rancher.
    - Go to **Security > User > Add User**



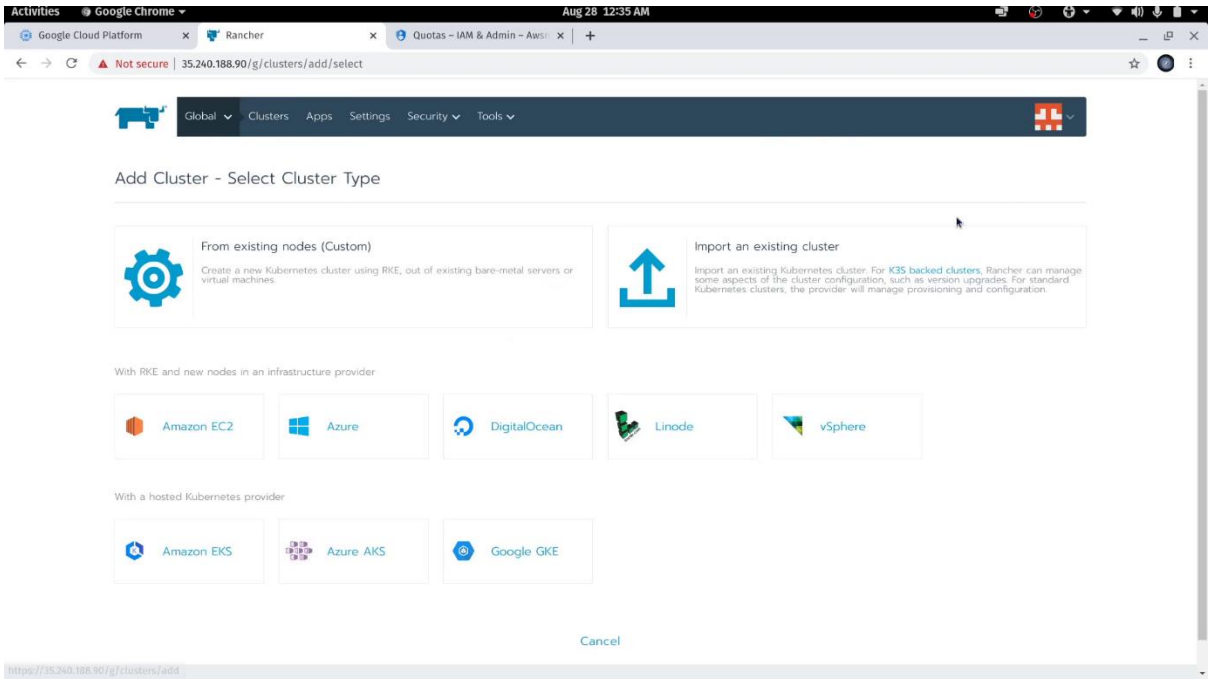
- Then enter username, password, display name and set global permissions for enforcing RBAC.



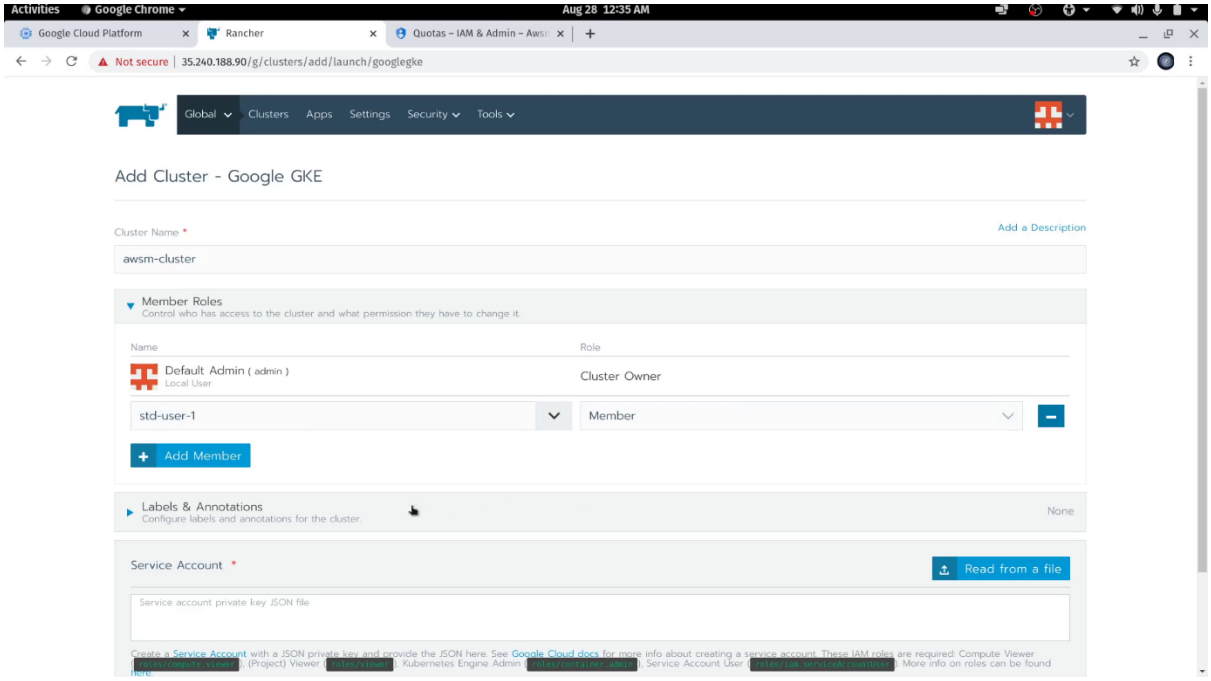
- Apart from the roles defined by Rancher, we can also add custom roles.



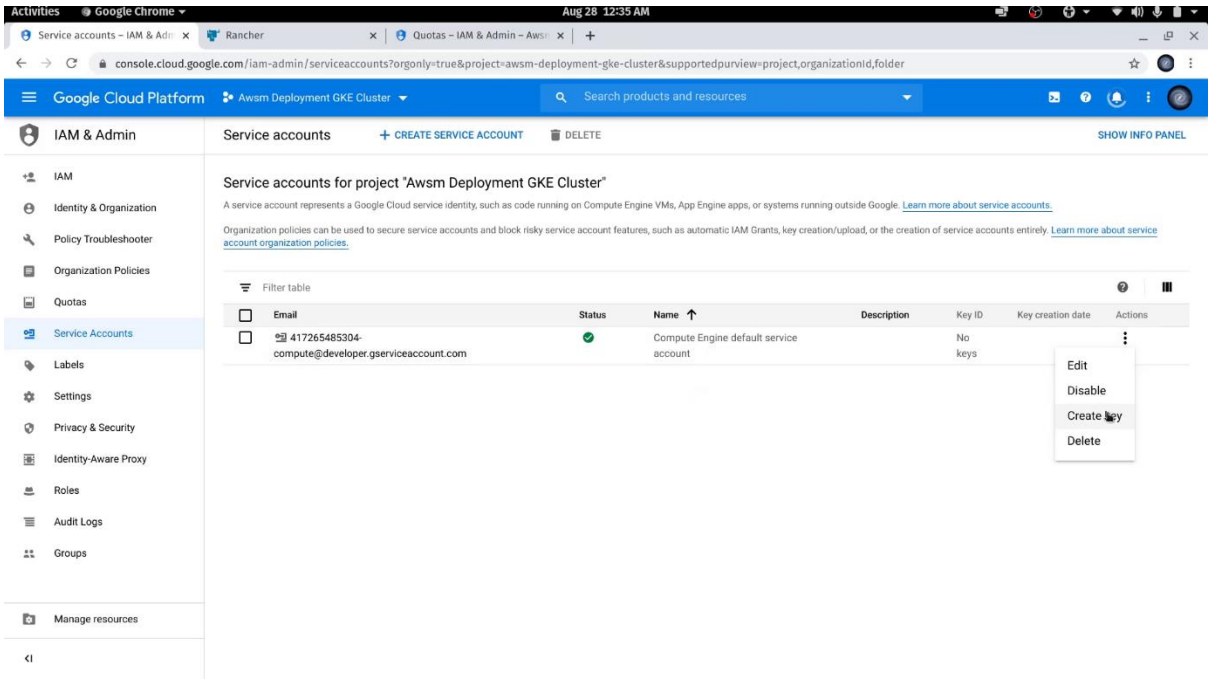
11. Now after creating a new user let's see how to add a cluster and provision it from GKE.
    - Go to **Cluster > Add Cluster**

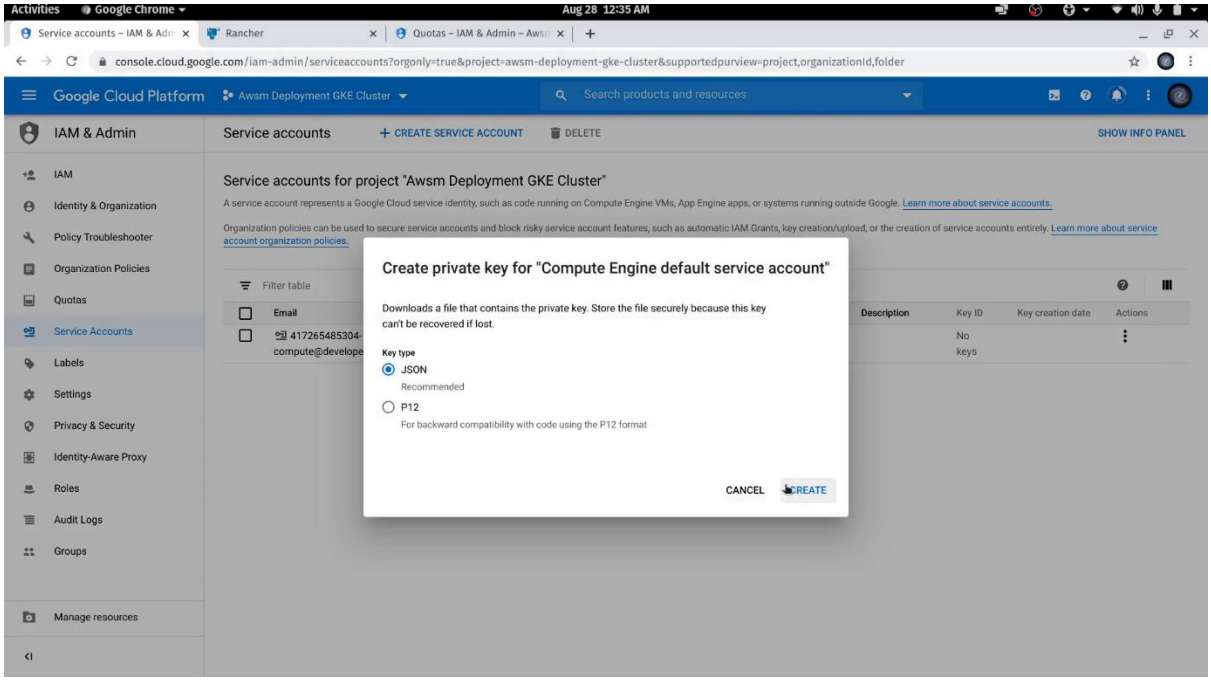


- Select **Google GKE**, enter **Cluster Name**, **Add Member** (if needed).

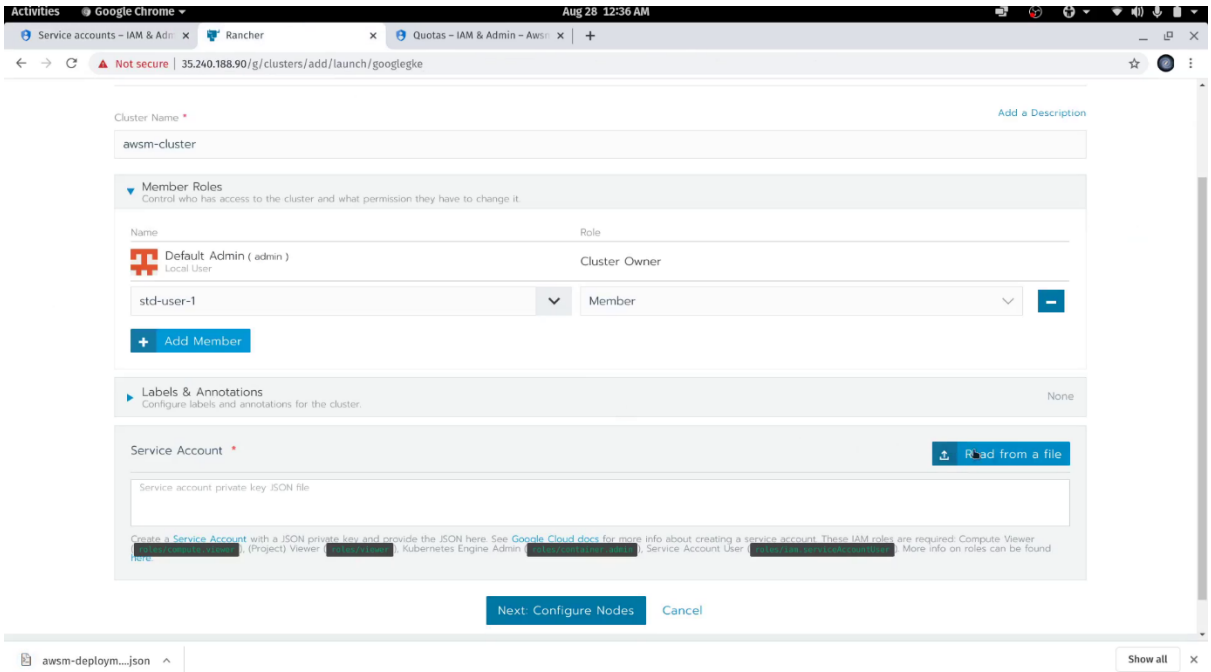


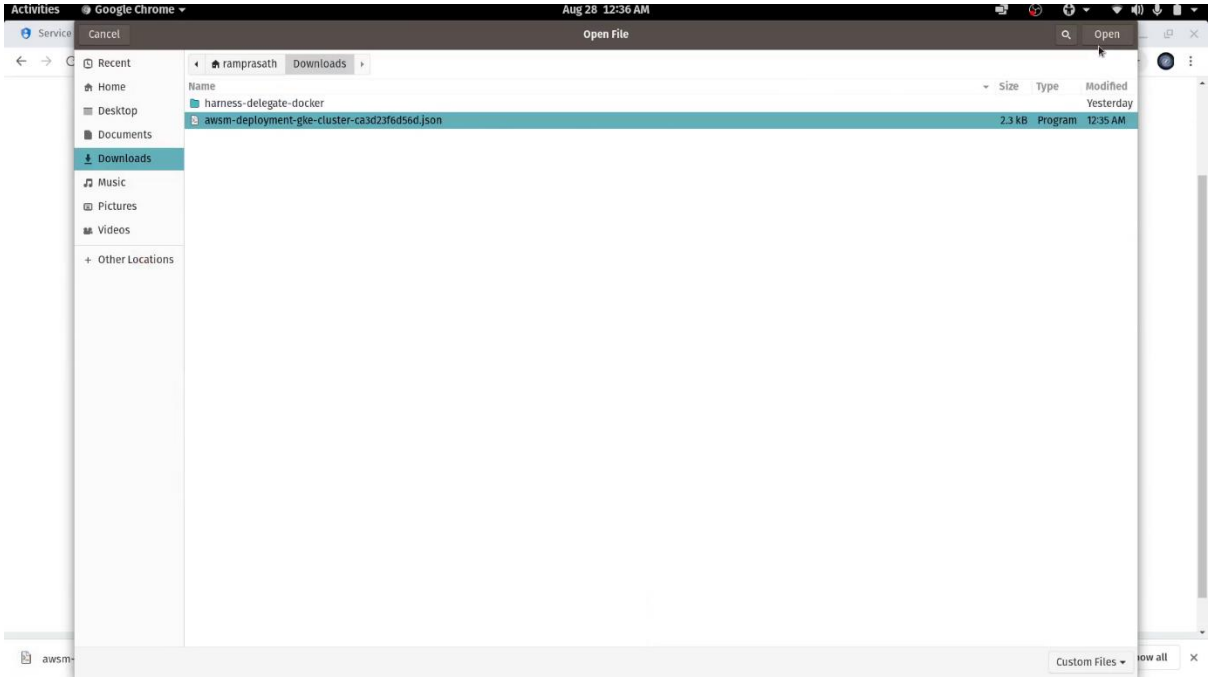
- Next, we need a JSON private key: **GCP > IAM & Admin > Service Account > Create key > JSON** under the project which was solely allocated for Kubernetes.



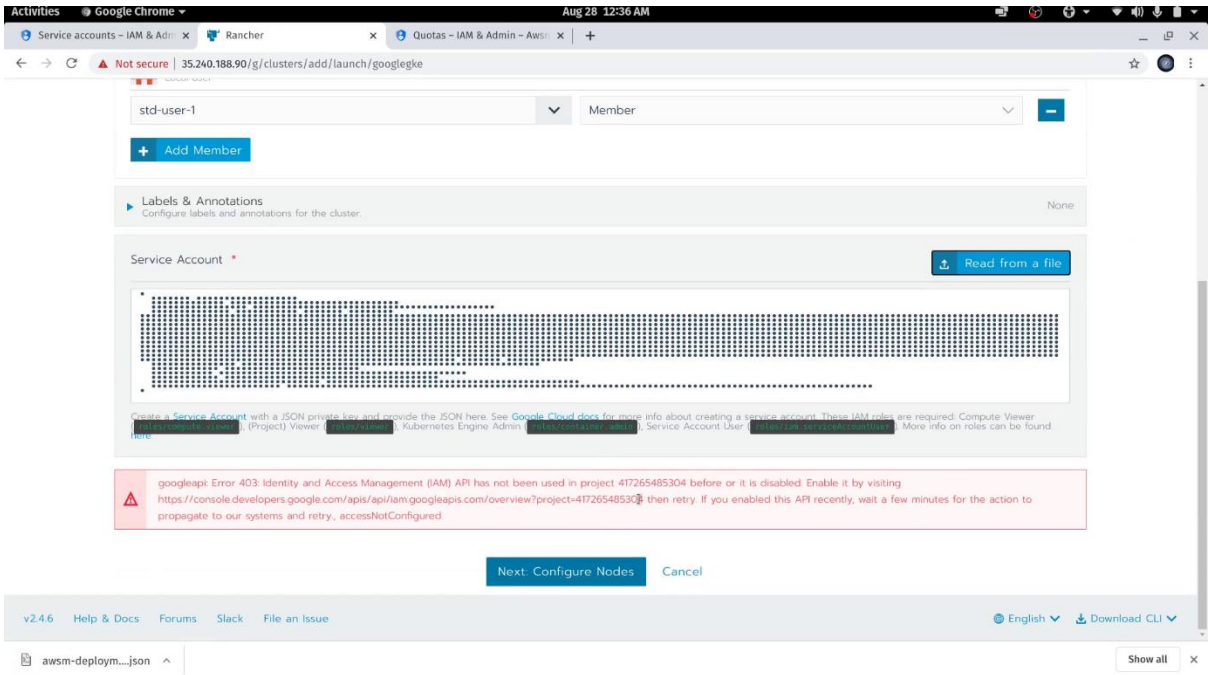


- After creating the key, we need to upload the key to Rancher.

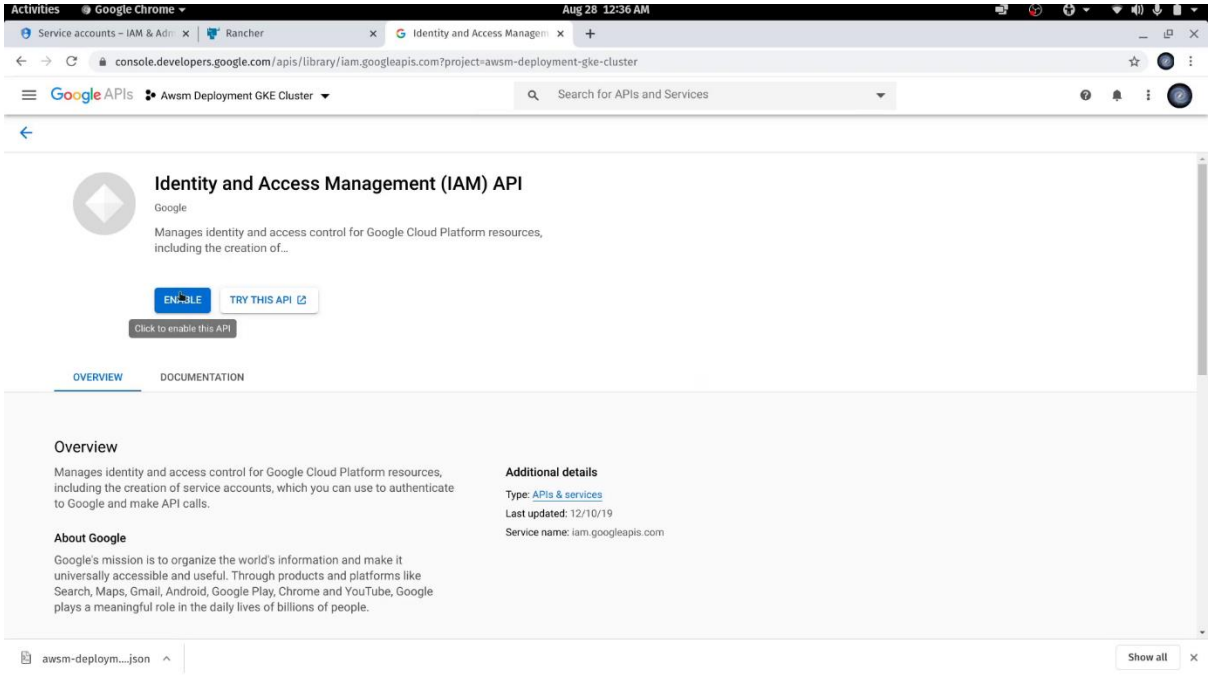




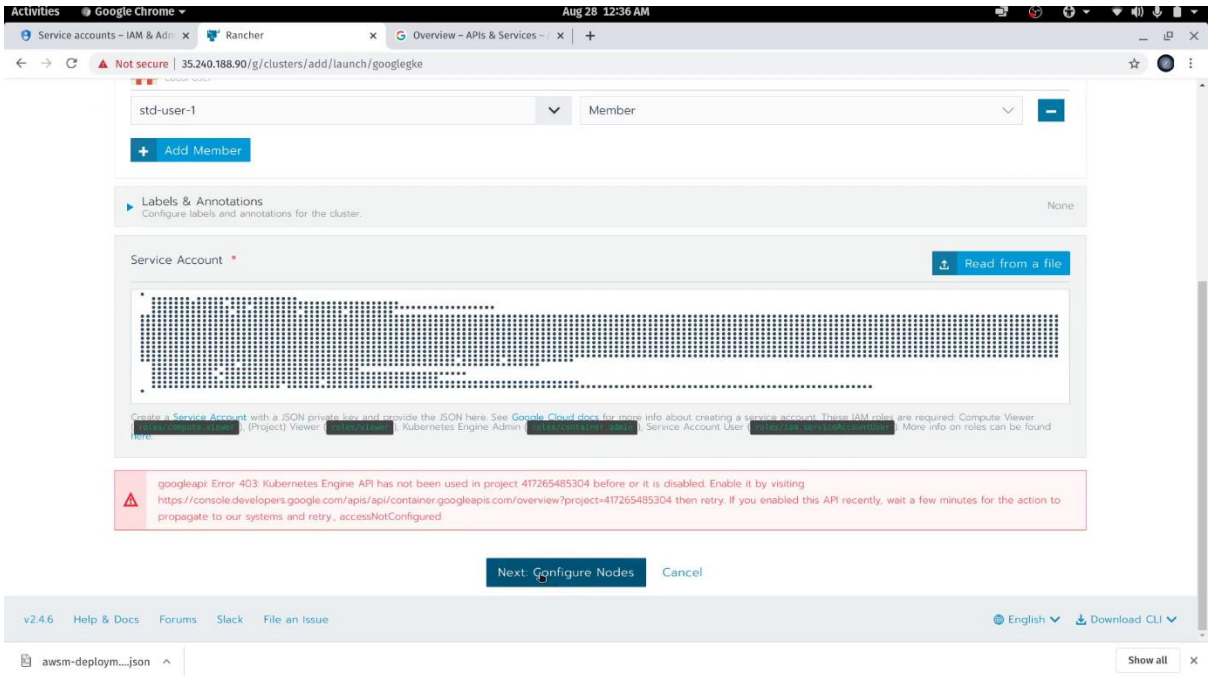
- Now Rancher will throw an error.



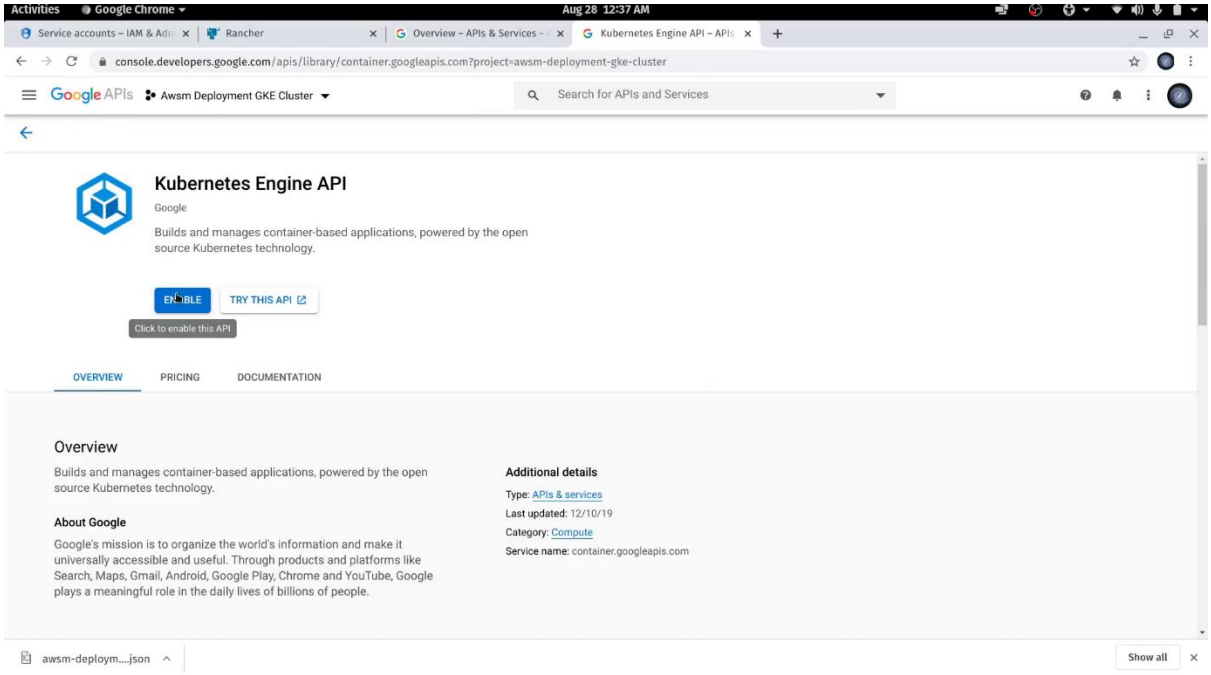
- To resolve the error, we need to enable IAM API in GCP's API Console.



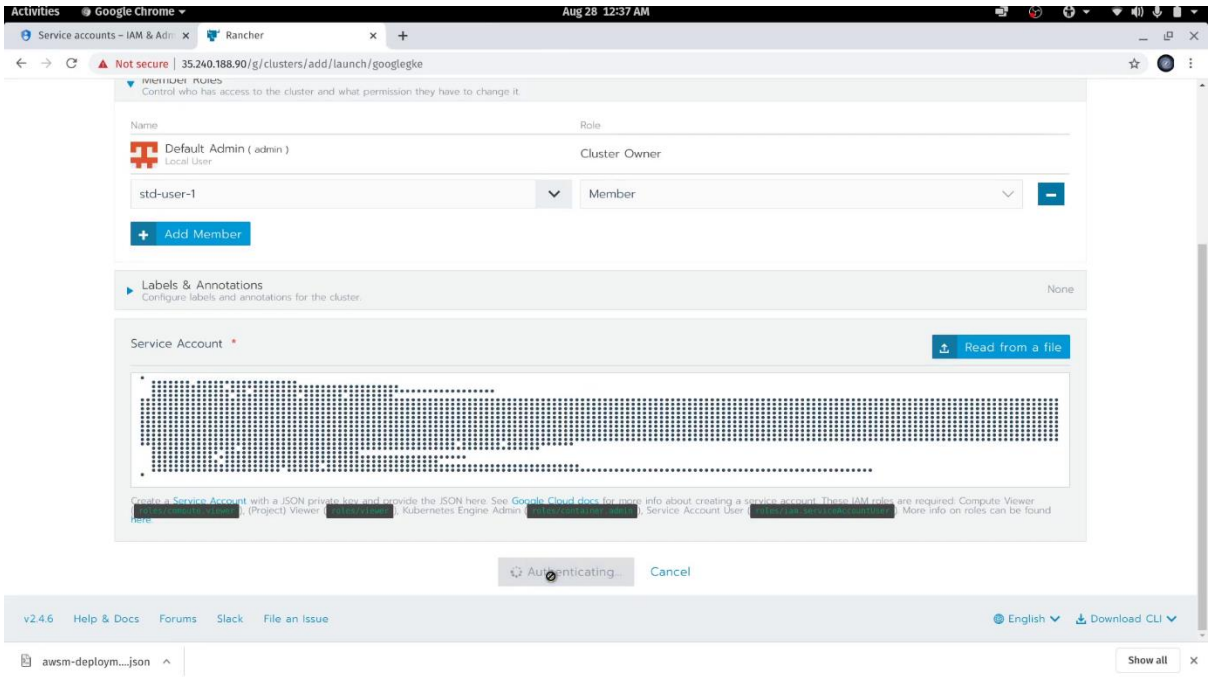
- After enabling the IAM API, an error will be thrown by Rancher to enable another API.



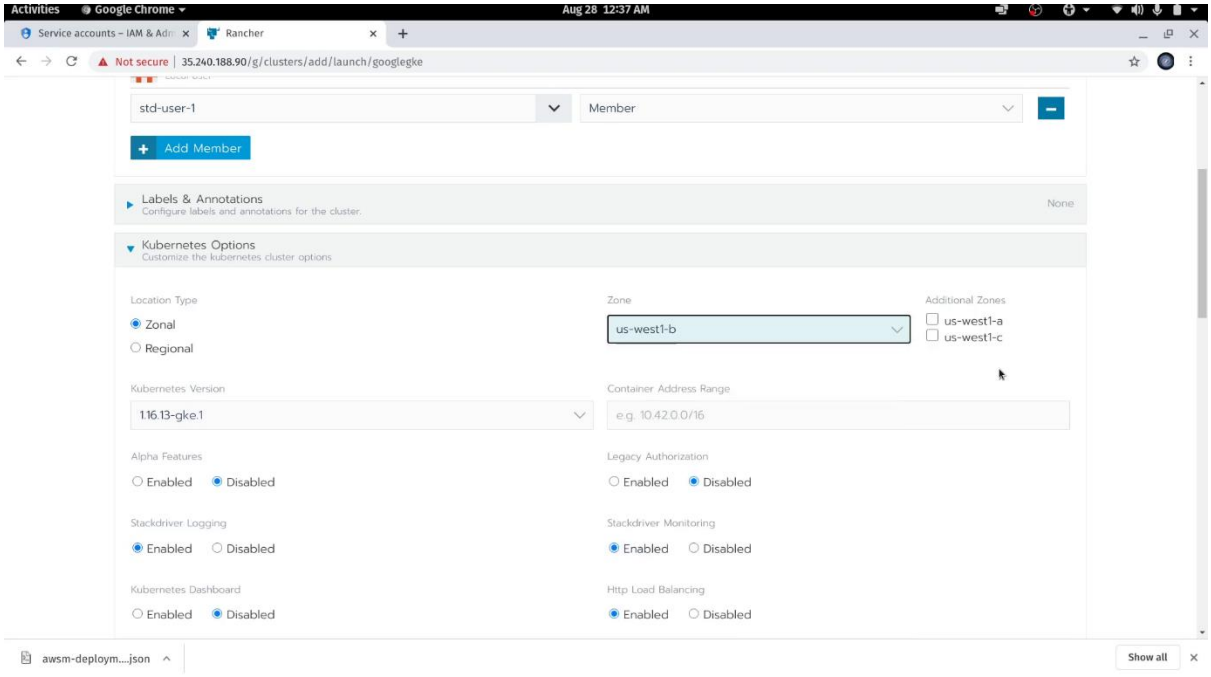
- To resolve the error, we need to enable Kubernetes Engine API in GCP's API Console.



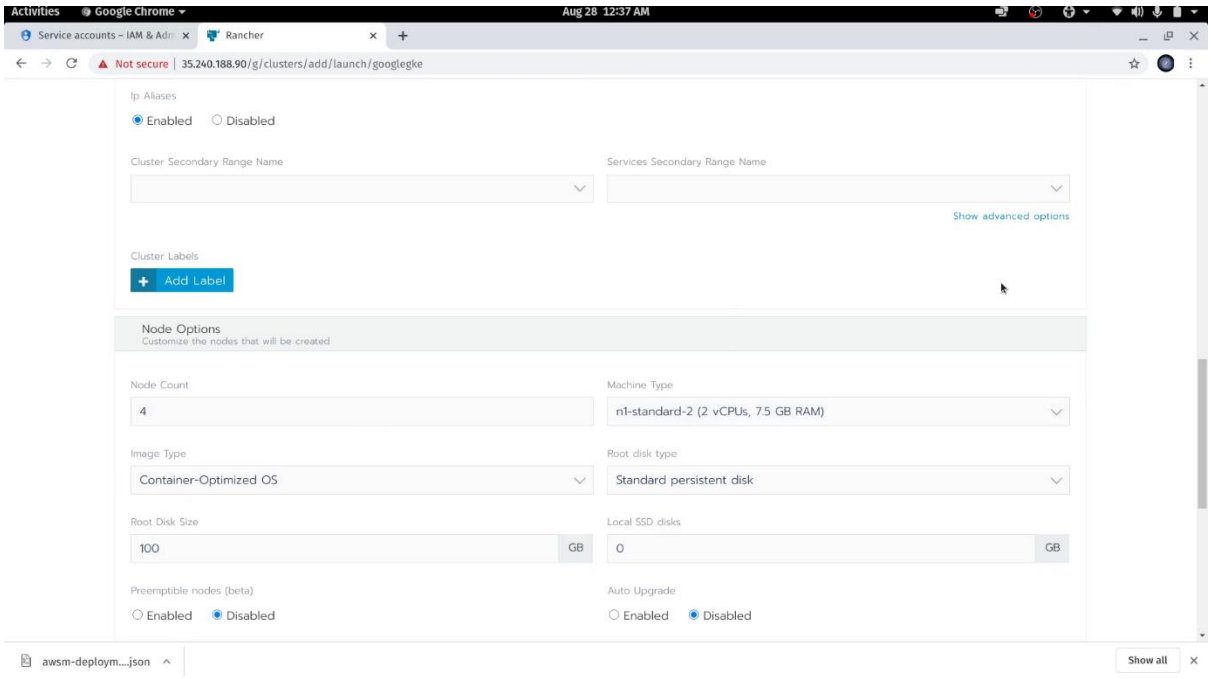
- Once the API is enabled, head back to Rancher and click **Configure Nodes**



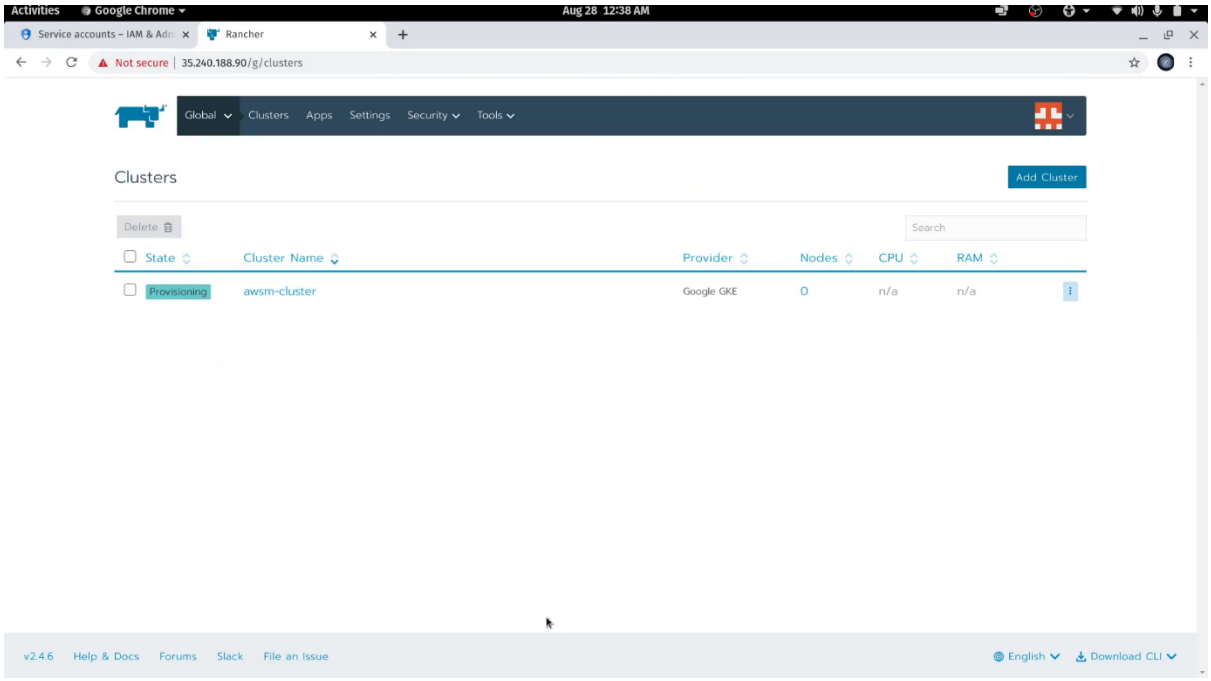
- Select a desirable zone as per requirements.



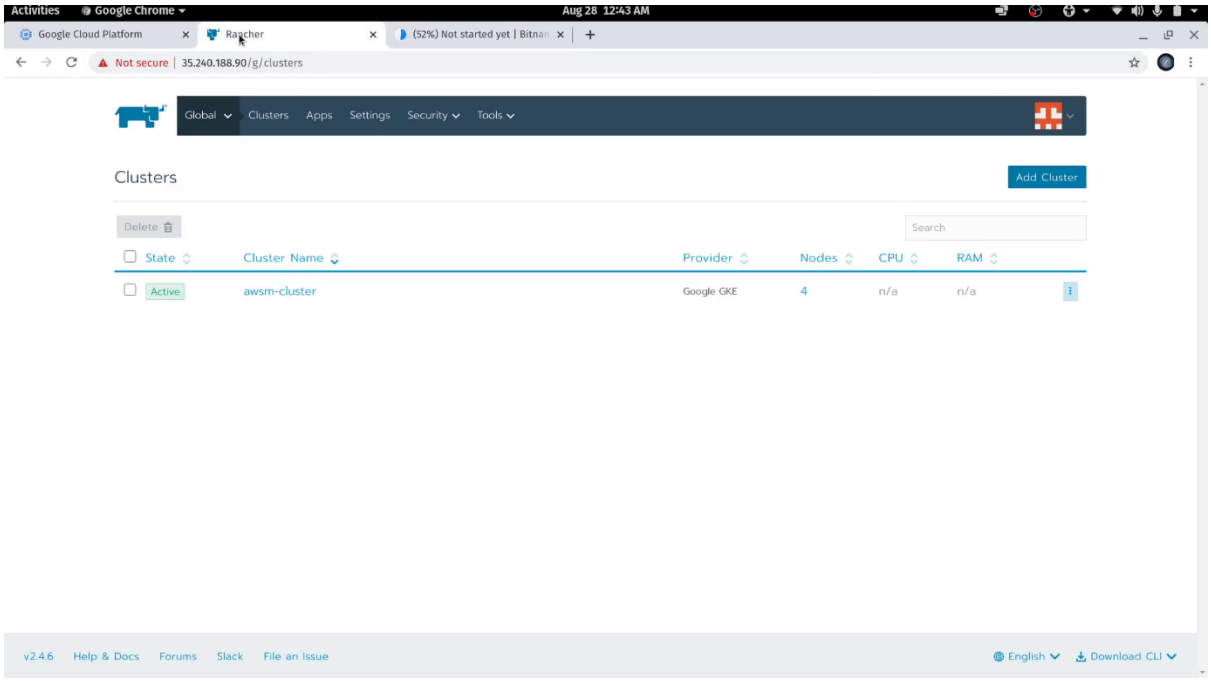
- Then configure node options



- Scroll down and click **Create**. The provisioning of our new cluster will take about 2-5 minutes.

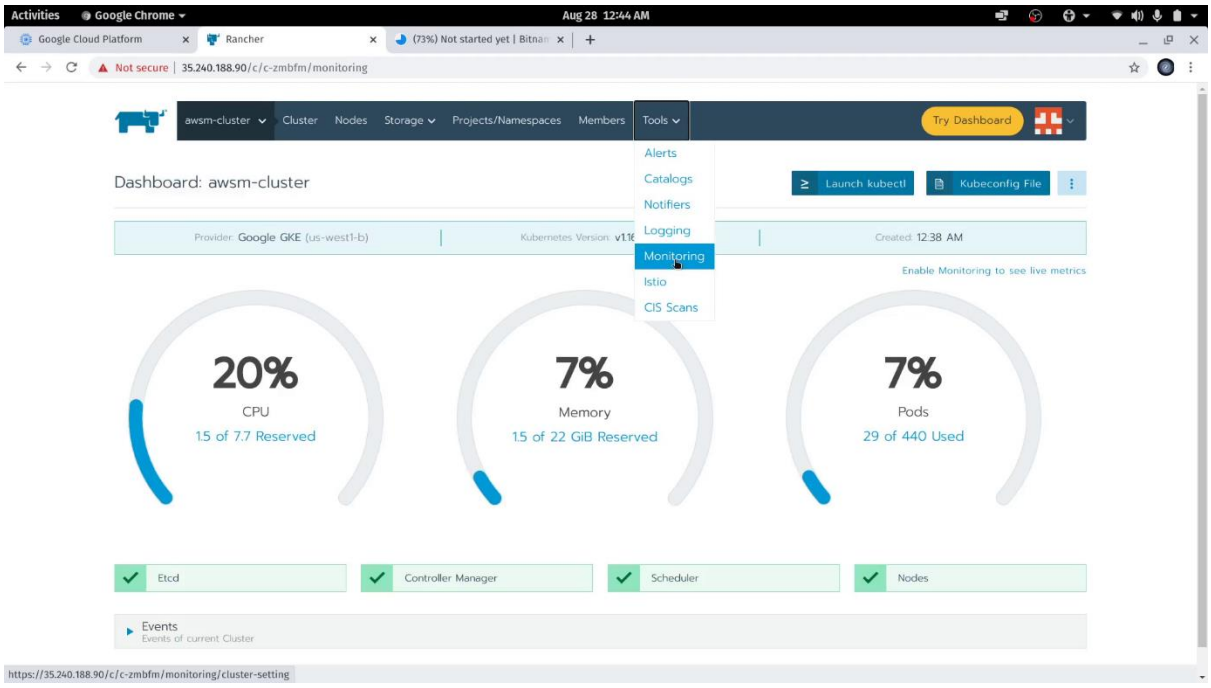


- Once the status of the cluster is active as displayed below. Then, we have successfully created a cluster using GKE.

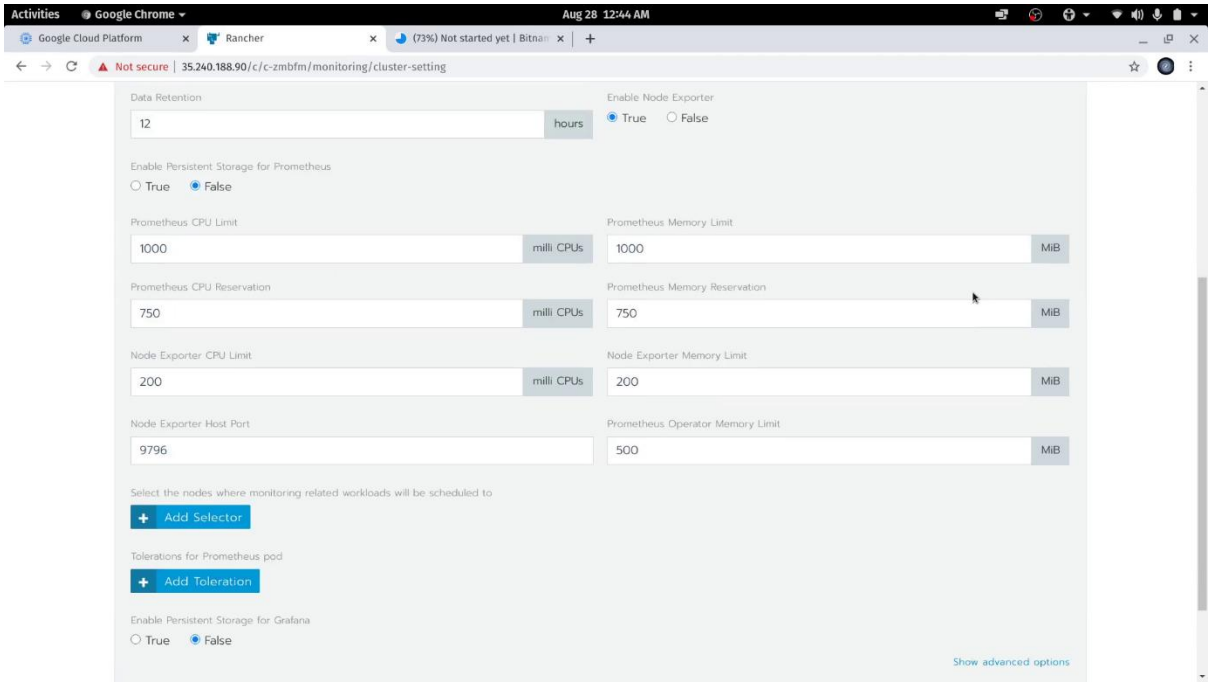


12. Once we have created the cluster, let's enable monitoring

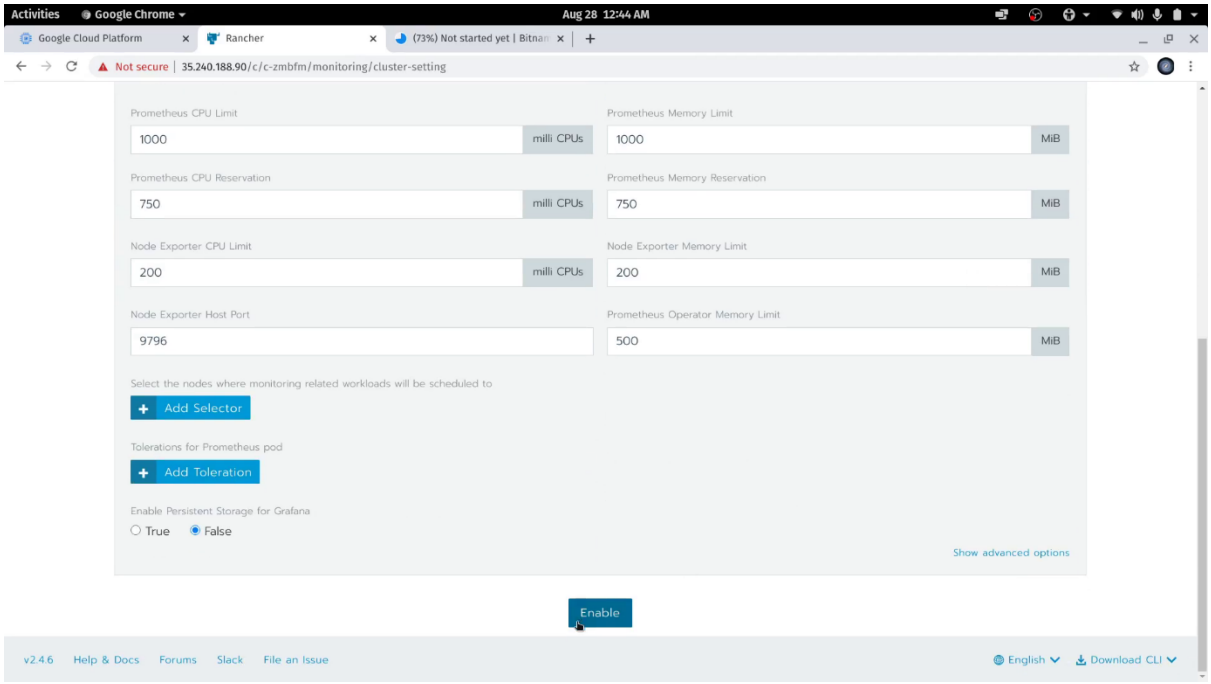
- **Clusters > awsm-cluster > Tools > Monitoring**



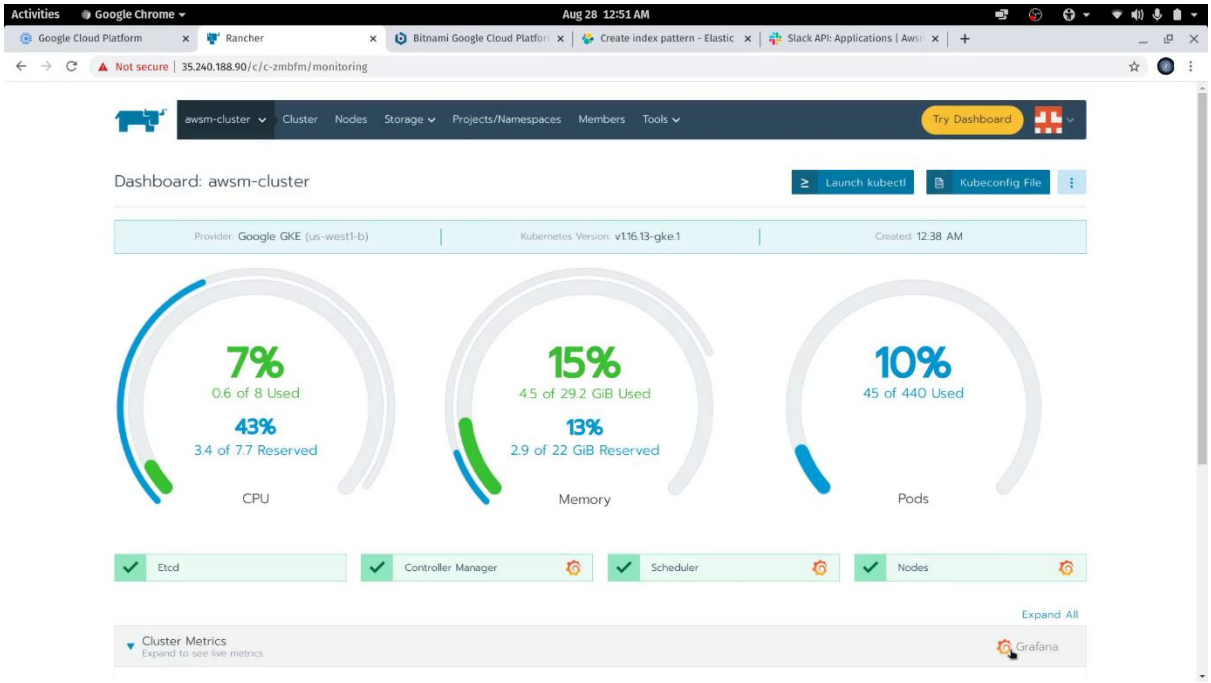
- **Enable Persistent Storage for Prometheus** for actual use case.



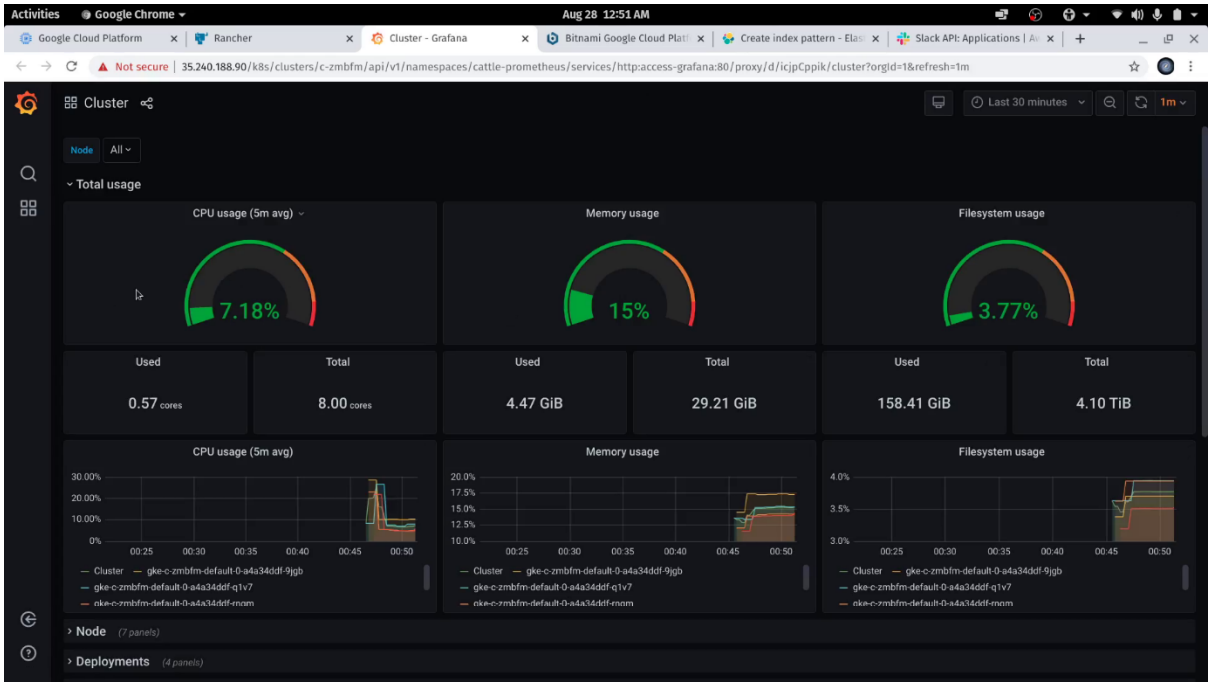
- Then scroll down and click **Enable**.



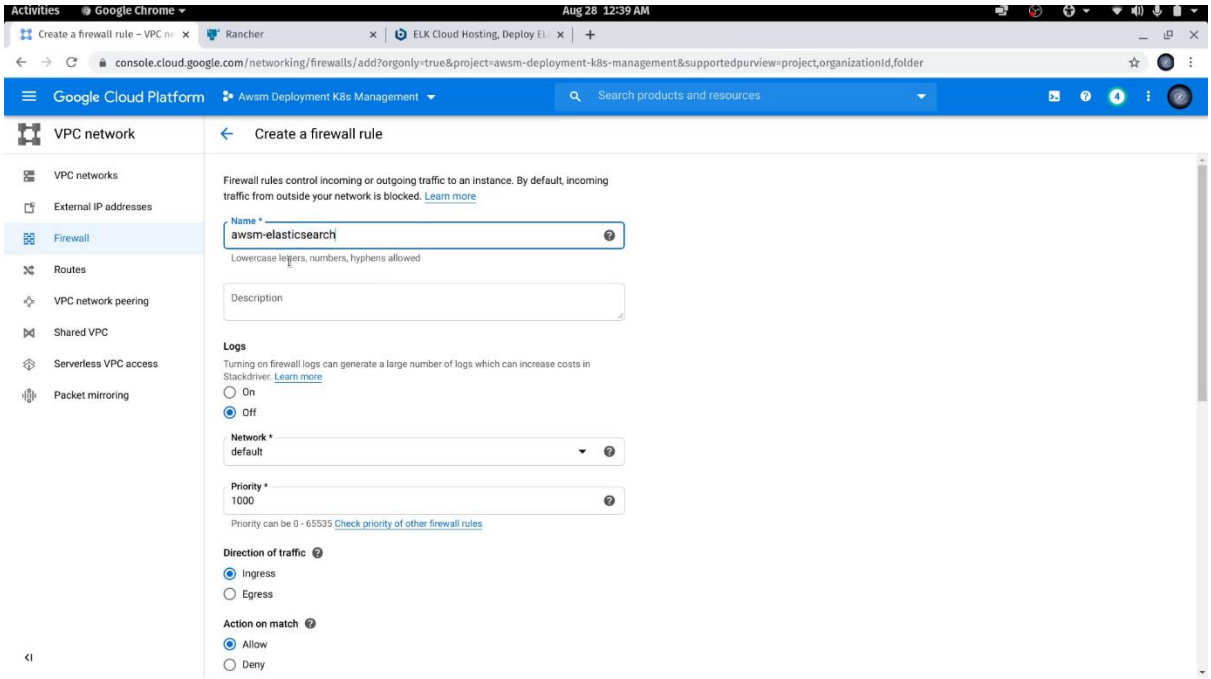
- Once monitoring is enabled and the monitoring API is ready, we can go to Grafana dashboard by clicking on the Grafana icon.



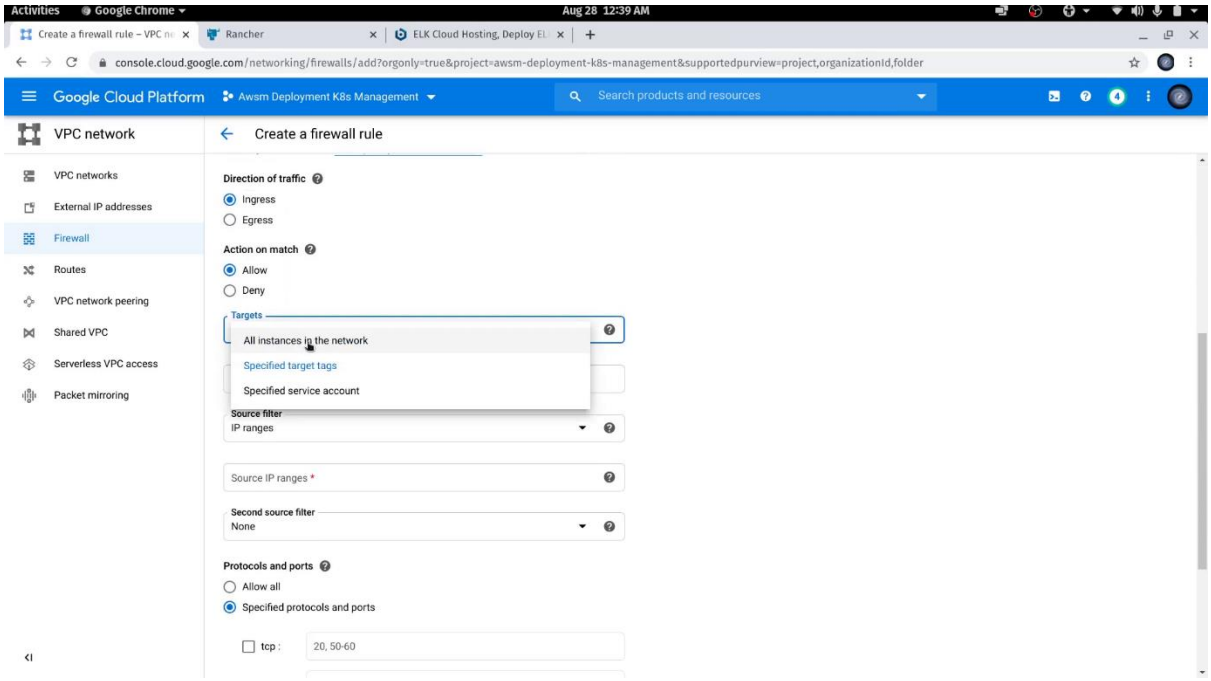
- We can now access various metrics from Grafana dashboard.



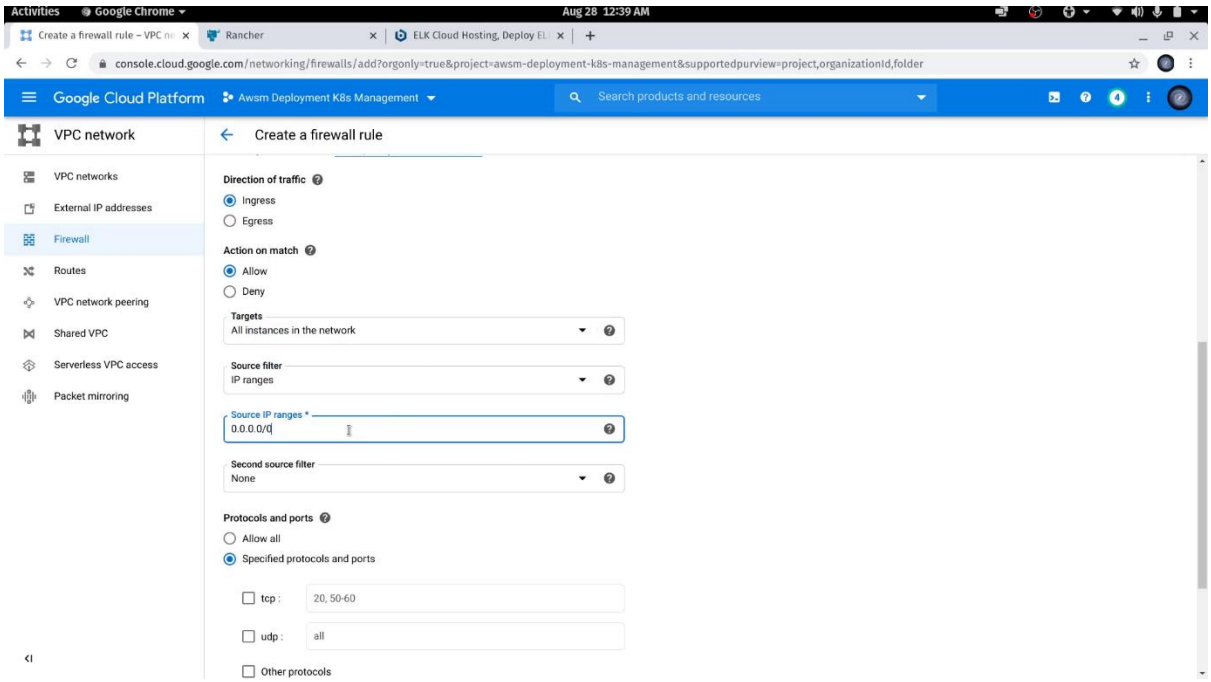
13. Now we will setup ELK Stack using Bitnami Launchpad for Google Cloud Platform.
    - Note: Before going to the link we need to add a firewall rule in GCP to allow TCP port 9200 for Elasticsearch.



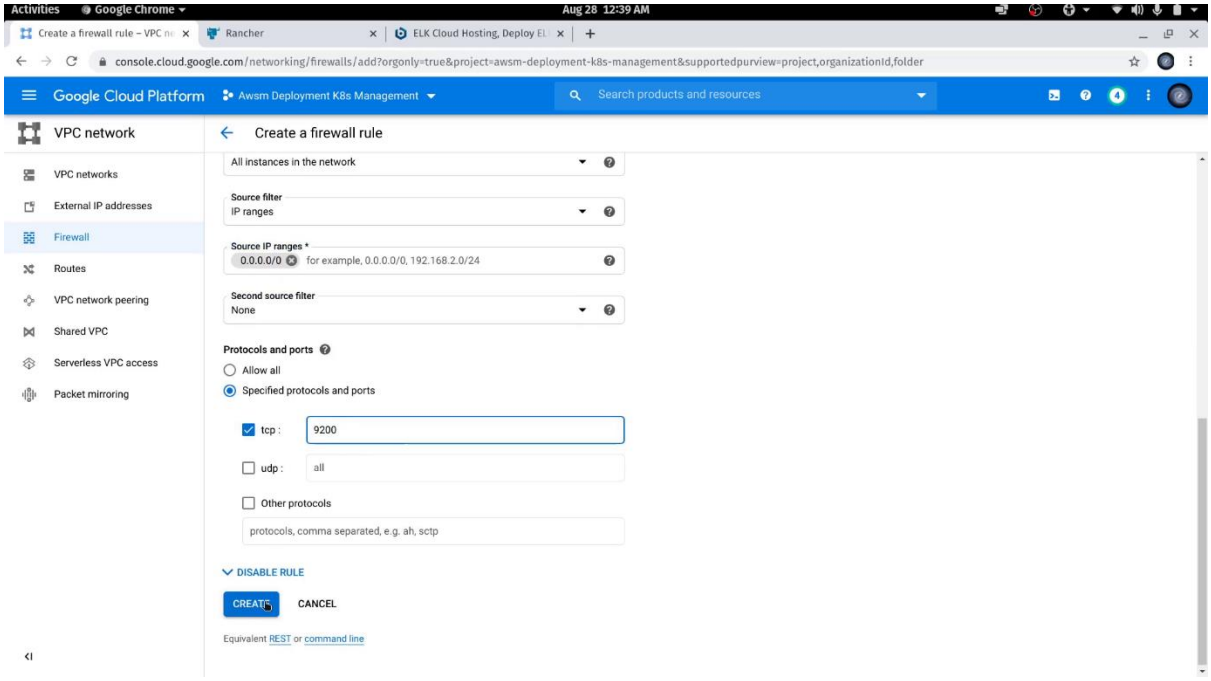
- Since this is a walkthrough, I am choosing **all instances in the network** but for actual use case we need to choose **specified target tags** (here, tag of ELK stack instance).



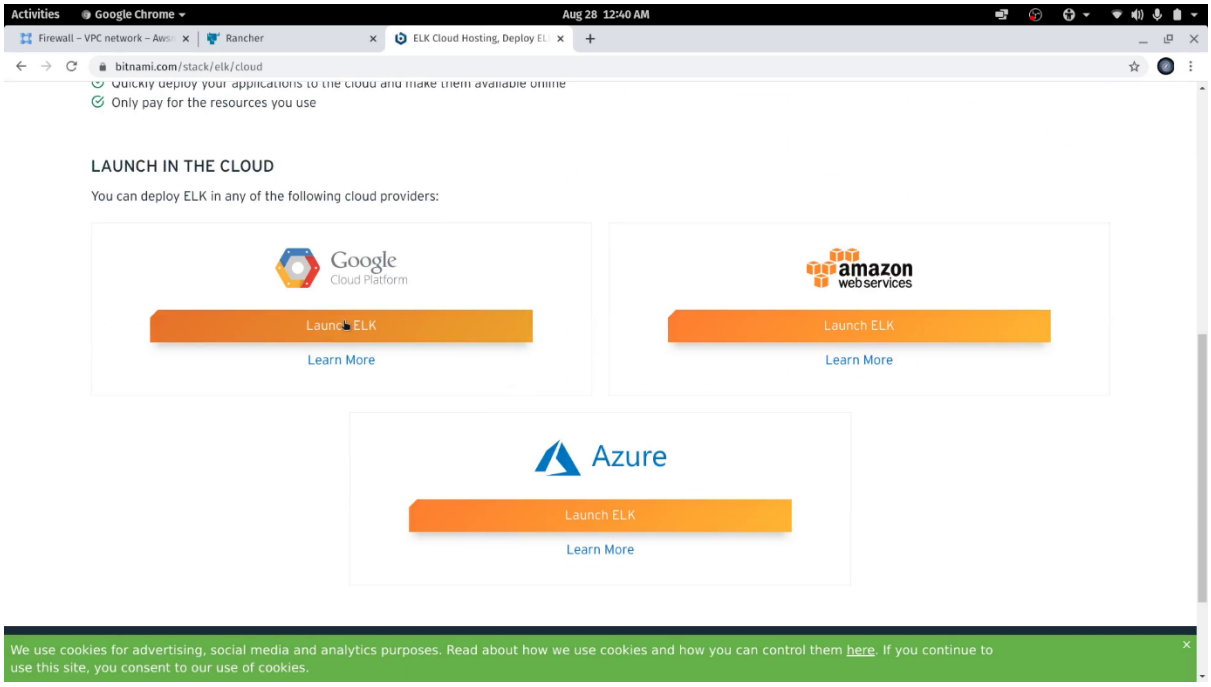
- For source IP ranges, enter **0.0.0.0/0**



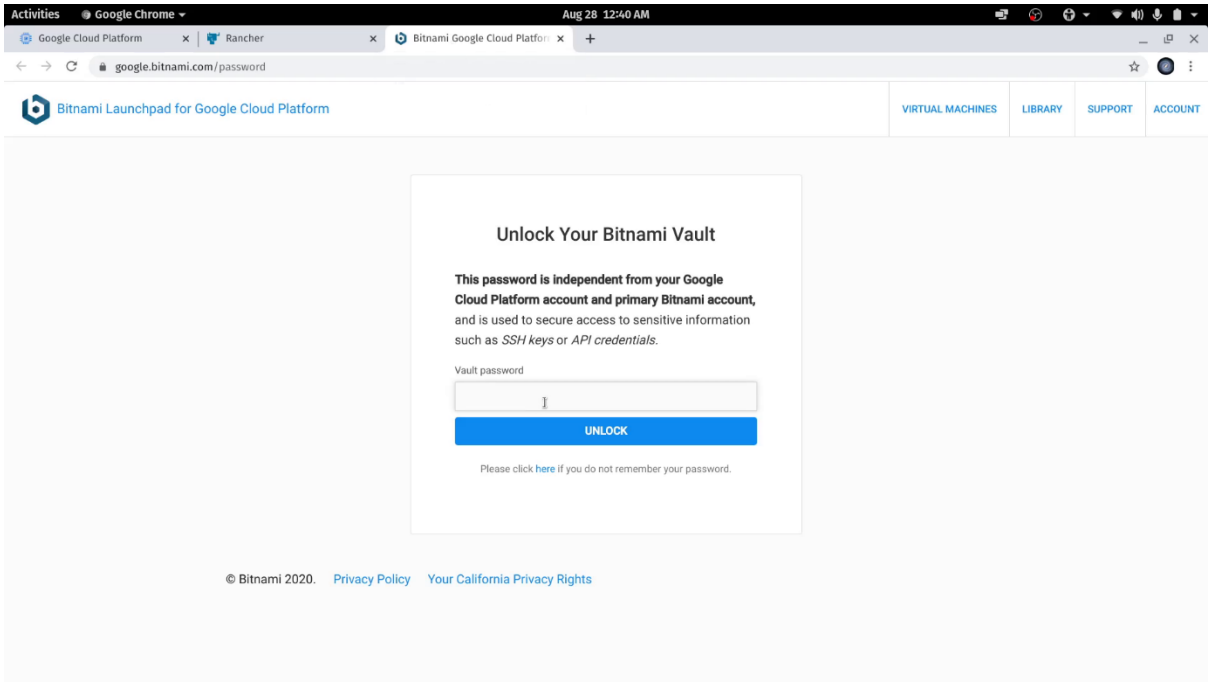
- Then select **tcp**, add port number **9200** and click **Create**.



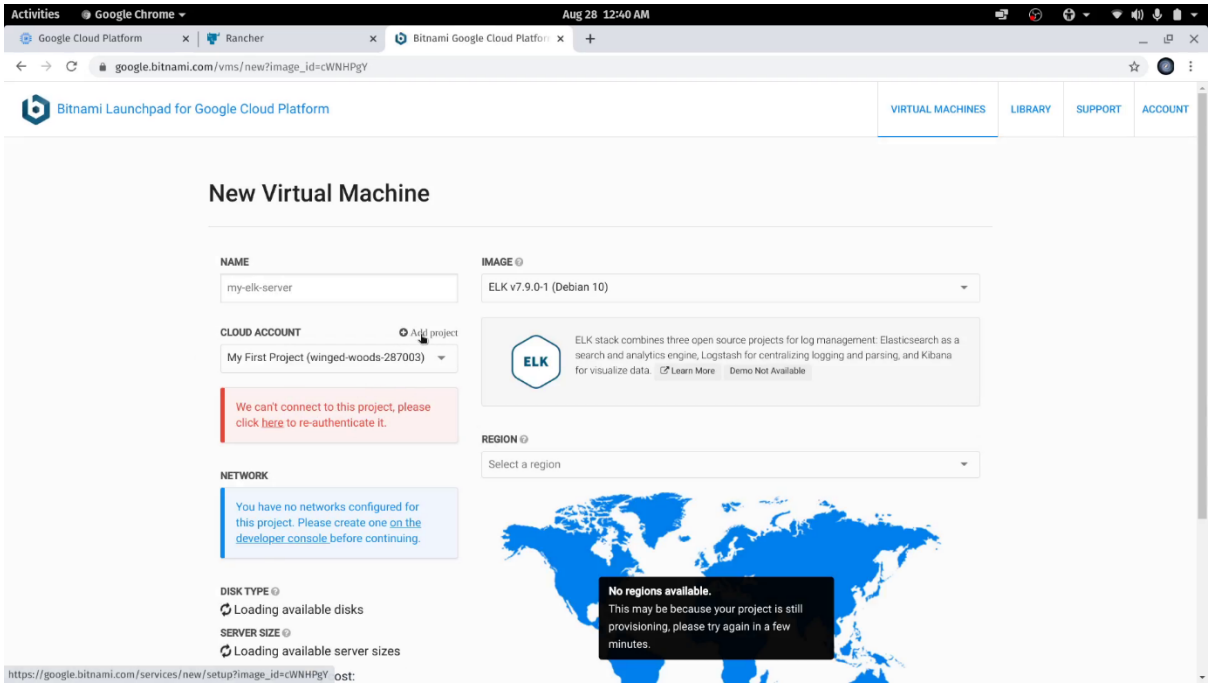
- Now we will head to Bitnami's cloud ELK Stack using the following link: https://bitnami.com/stack/elk/cloud
- Sign in to Bitnami with your google account which is used for accessing GCP account and launch ELK.



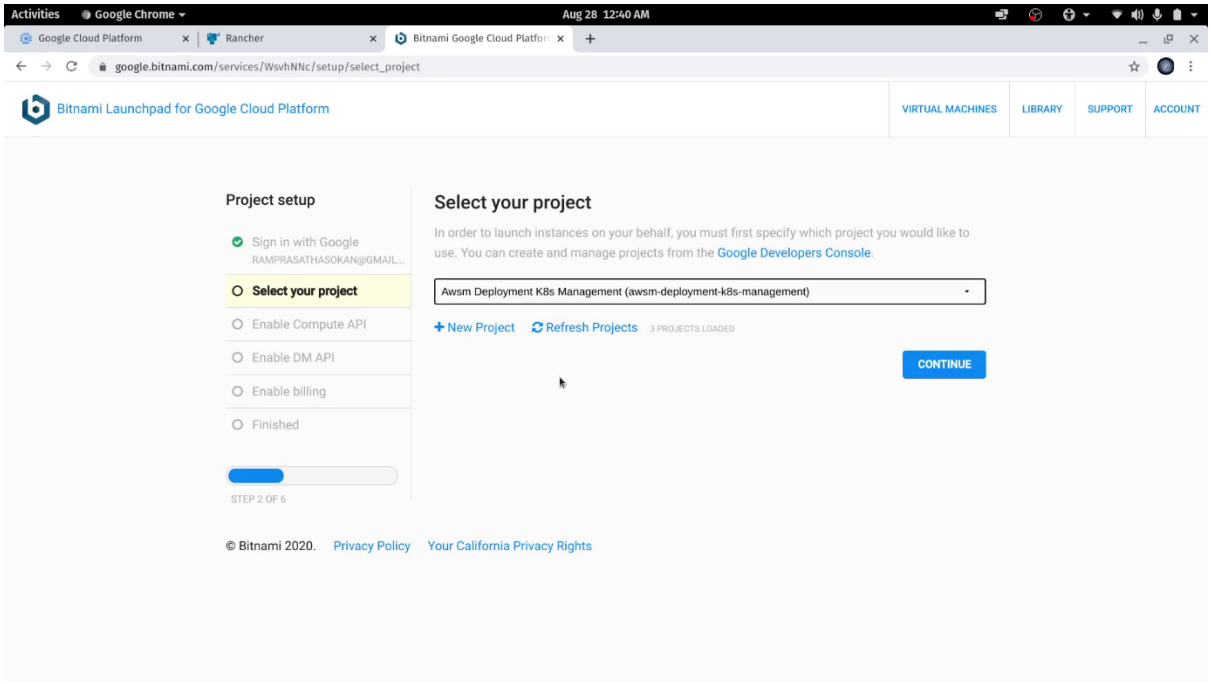
- Unlock Bitnami Vault



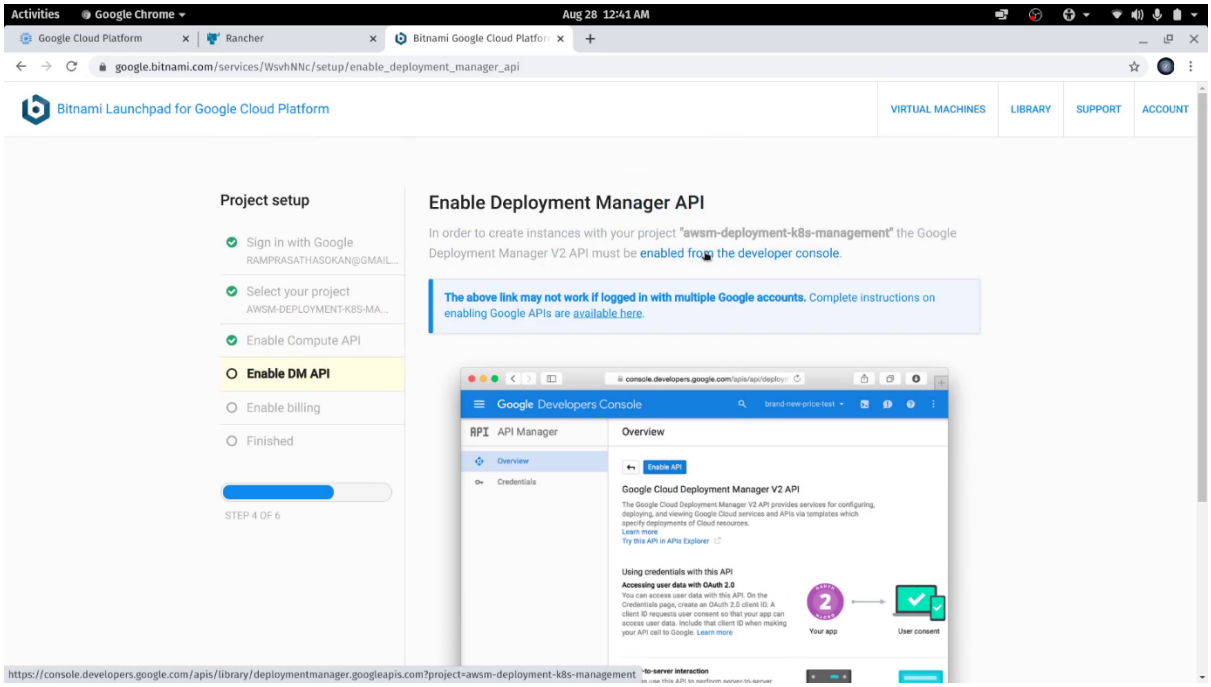
- Then we need to add project which is not used by Kubernetes cluster

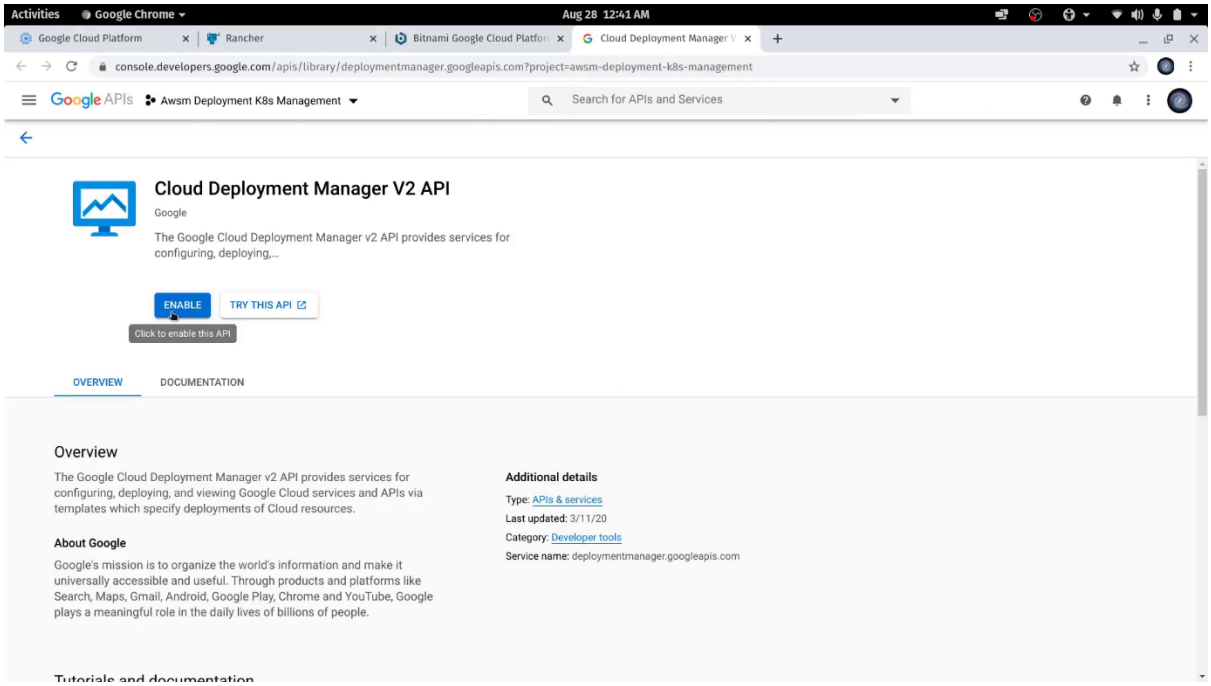


- Choose the appropriate project

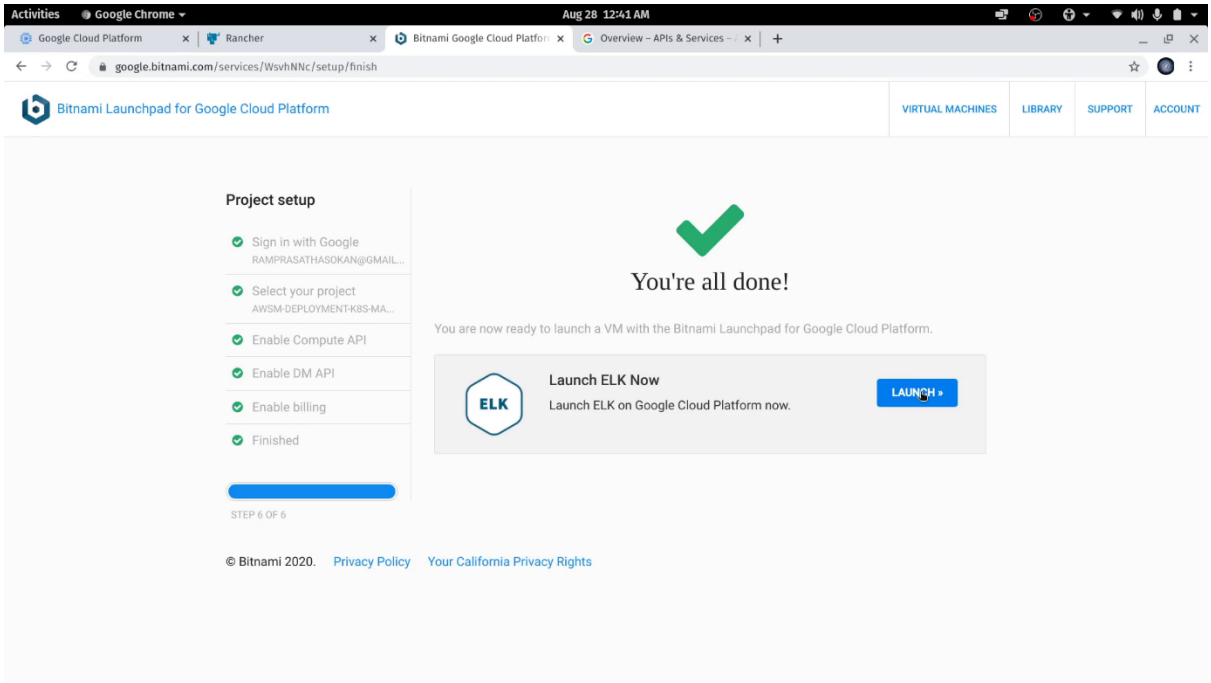


- Enable Deployment Manager API

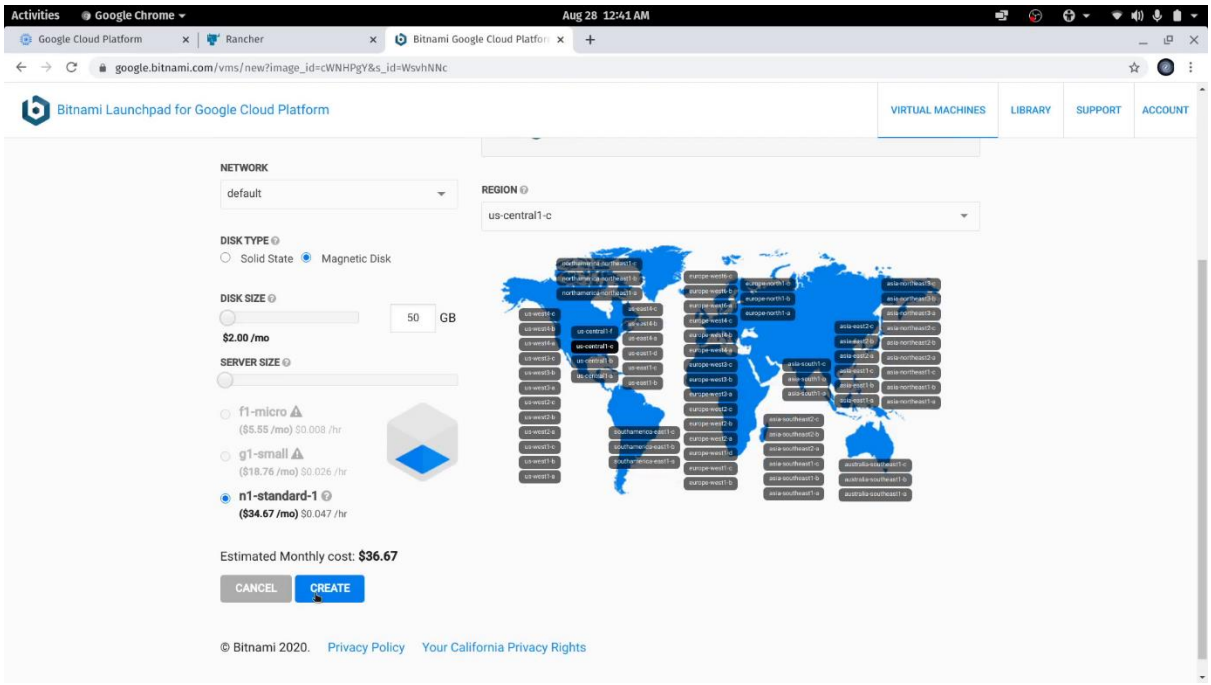




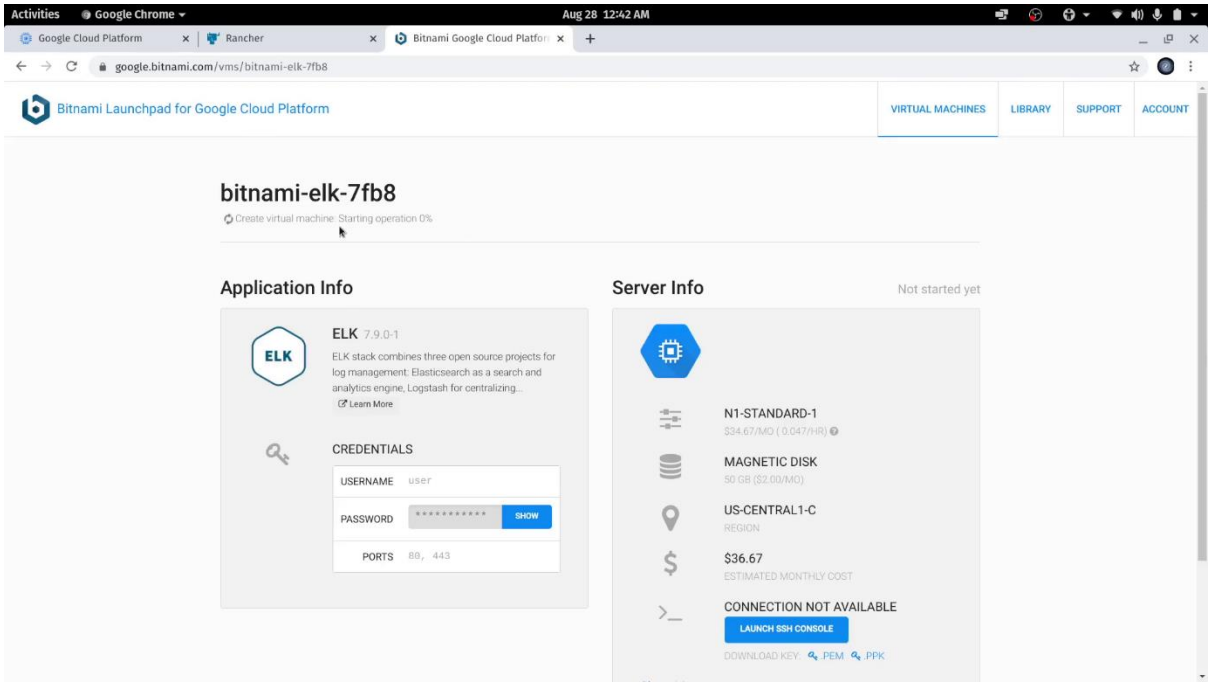
- Then head back to Bitnami and hit **CONTINUE**. Next, click **LAUNCH**.



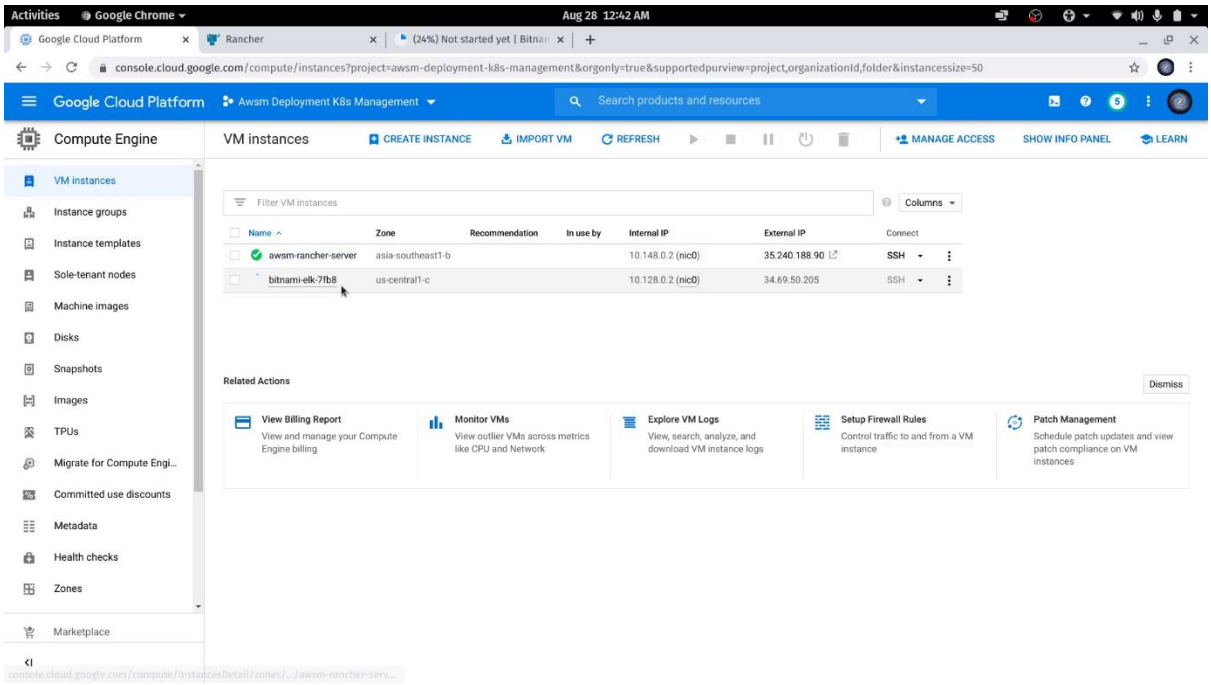
- Now configure instance type, select desirable region as per requirements and click **CREATE**.



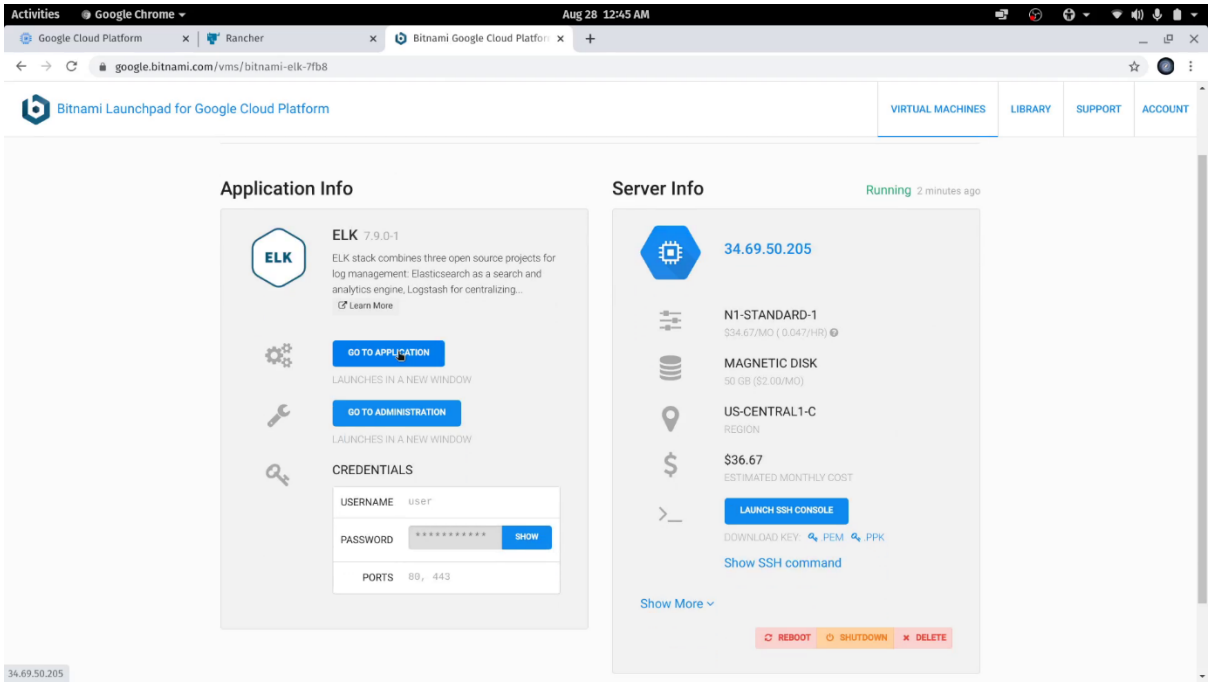
- Then we will be redirected to the console of the ELK instance in Bitnami Launchpad.



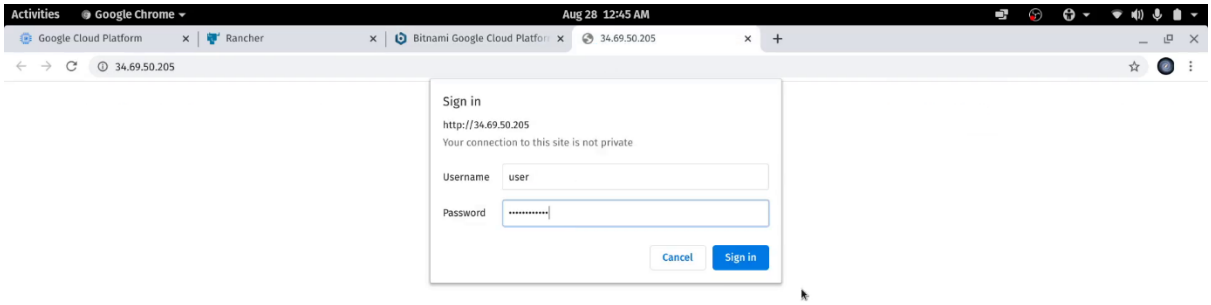
- We can verify the launch of ELK instance from GCP under the respective chosen project.



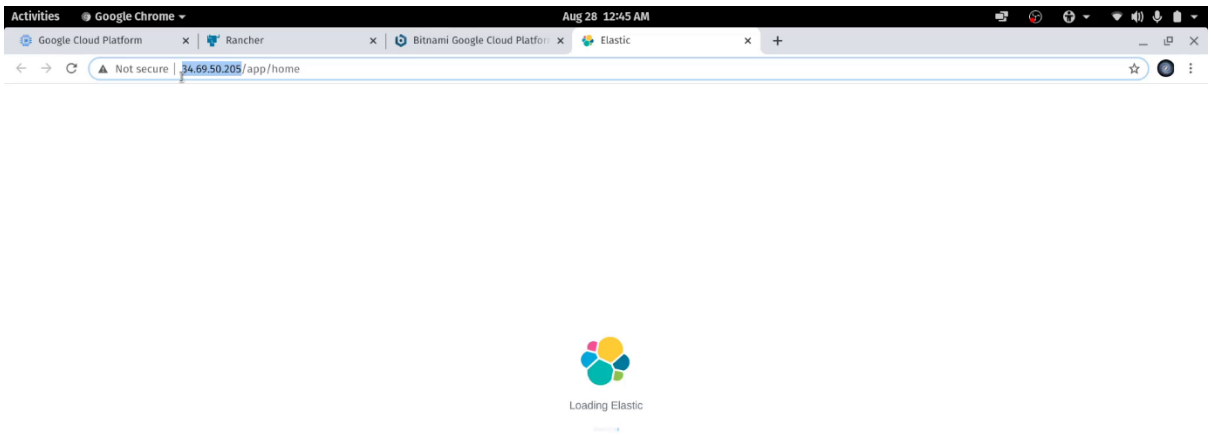
- Once the instance is created, click **GO TO APPLICATION**



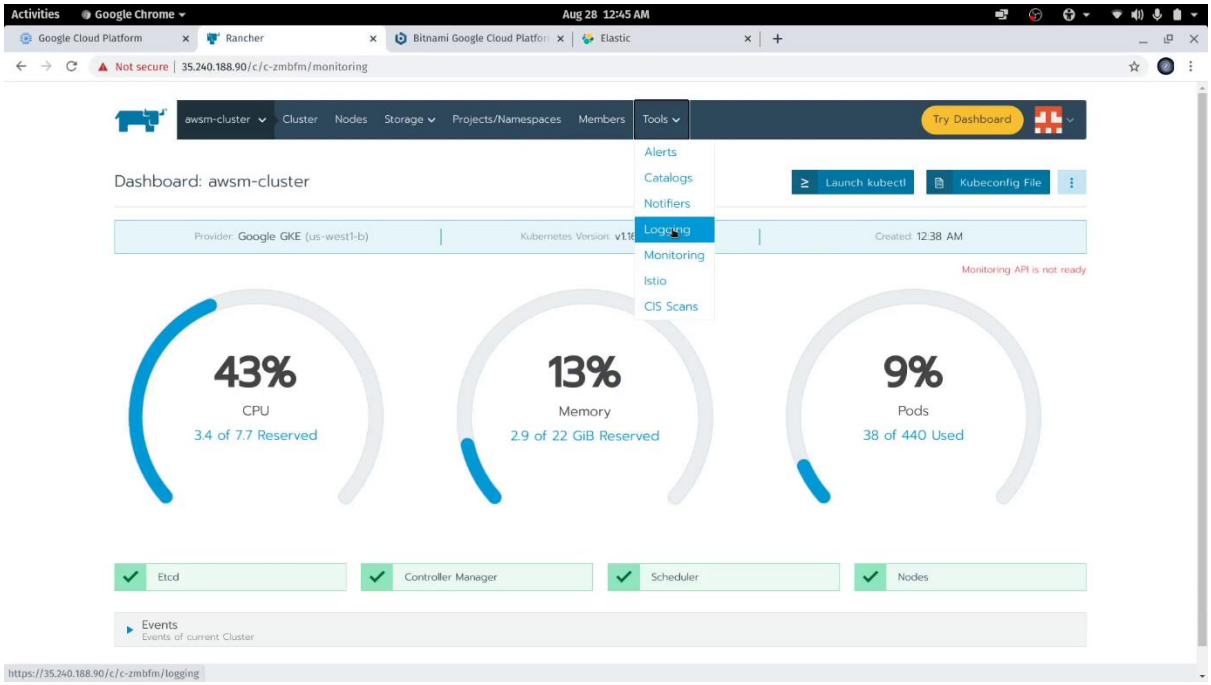
- Then enter **Username** and **Password** to **Sign in**.



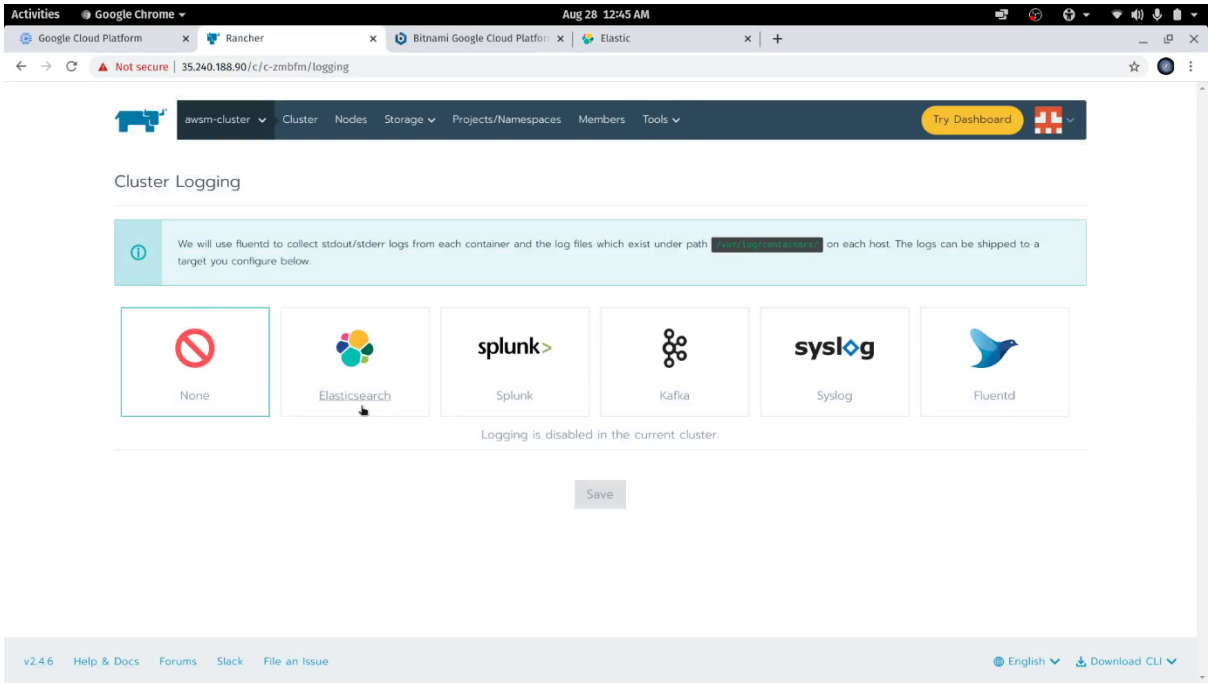
- Now, copy the external ip address of the ELK instance



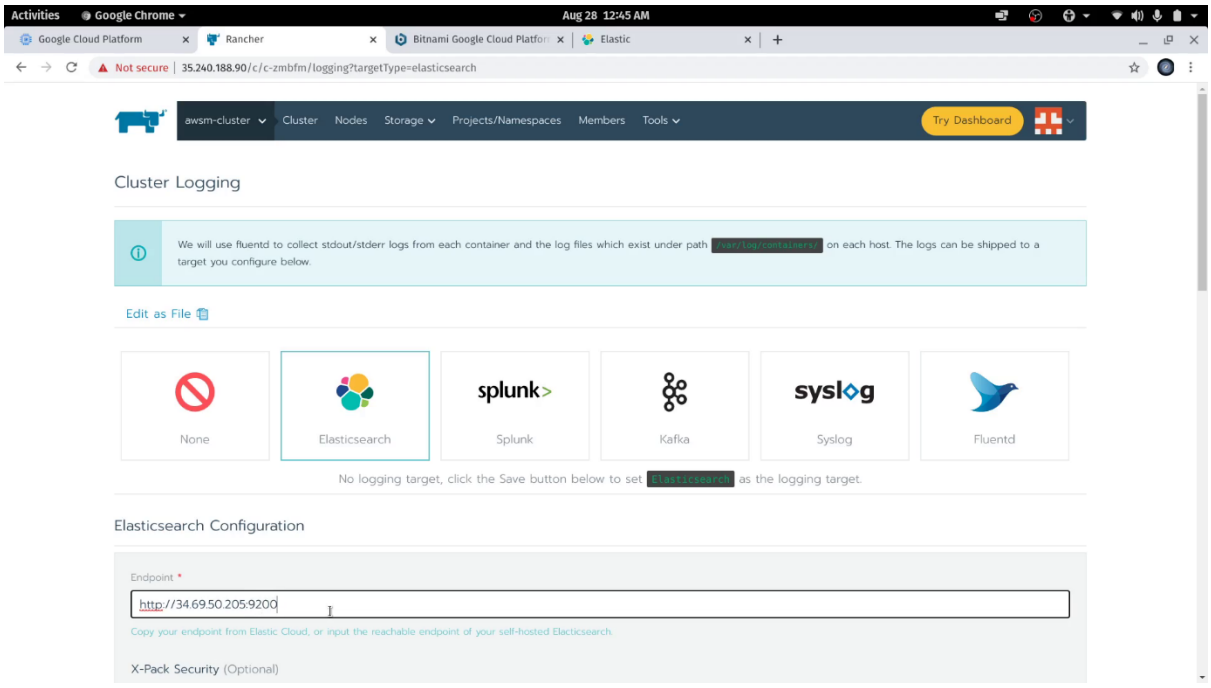
- Go to **Rancher > Cluster > awsm-cluster > Tools > Logging**
- Note: Make sure that monitoring API is up and Grafana is running before setting up logging.



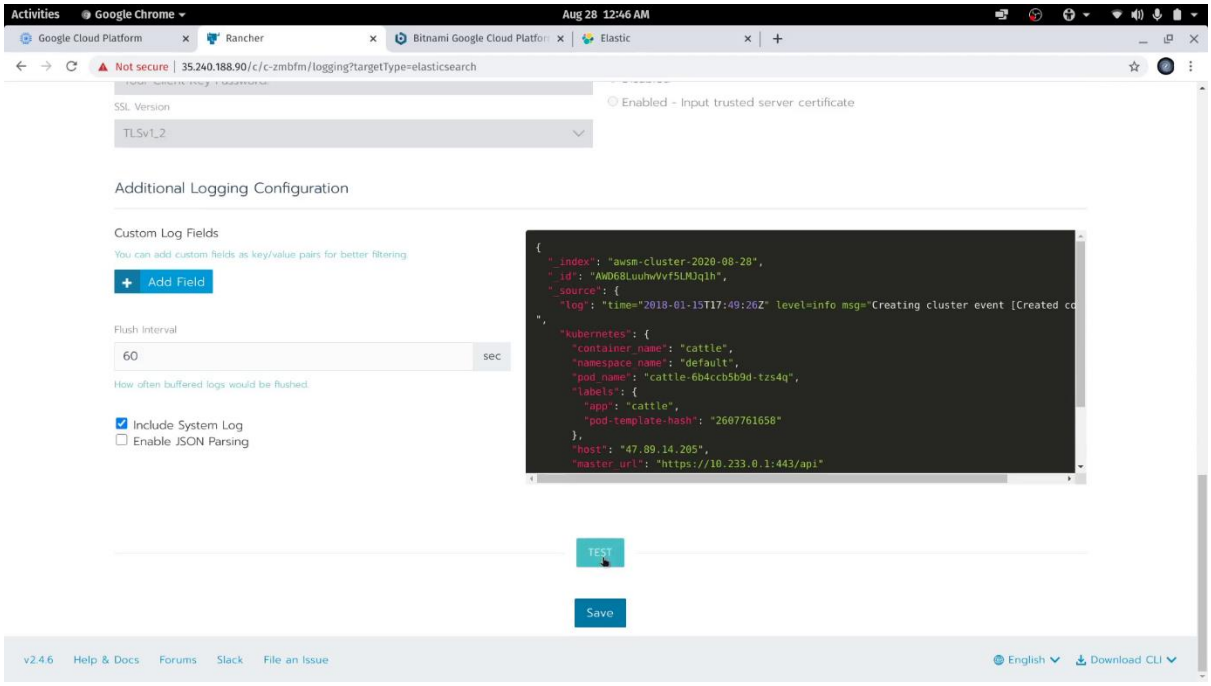
- Now select Elasticsearch from the following



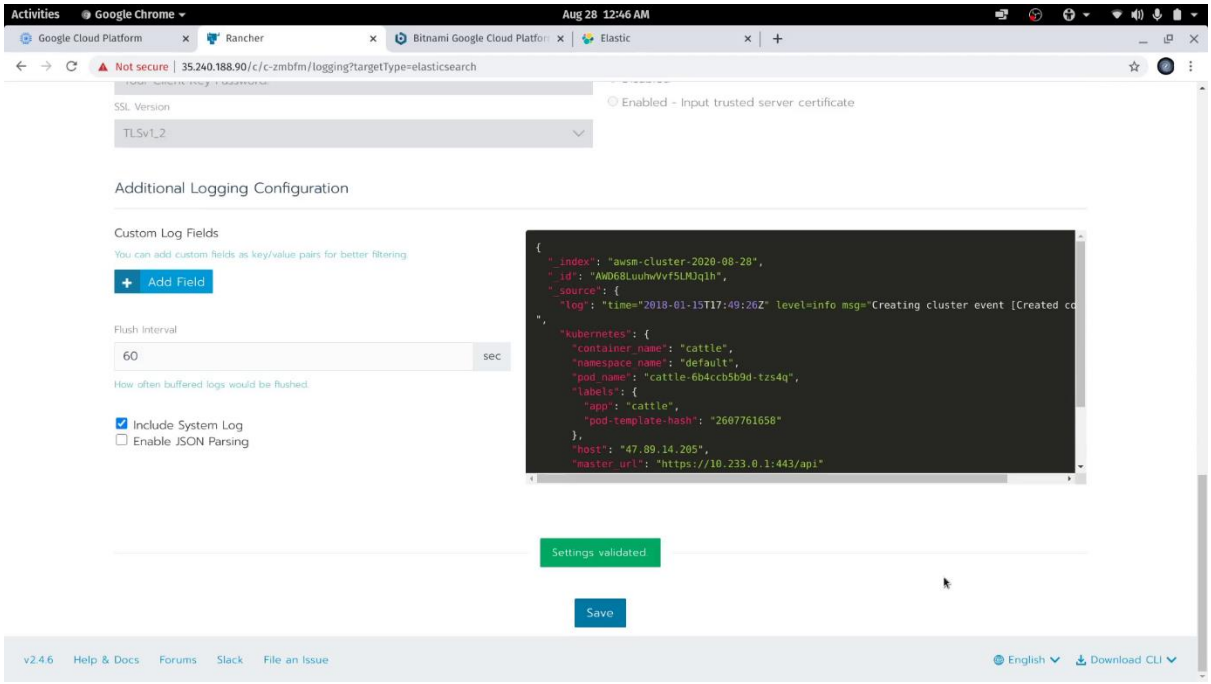
- Enter the external ip address of the ELK instance which we copied earlier and include the 9200 port along with it in the Elasticsearch endpoint field.



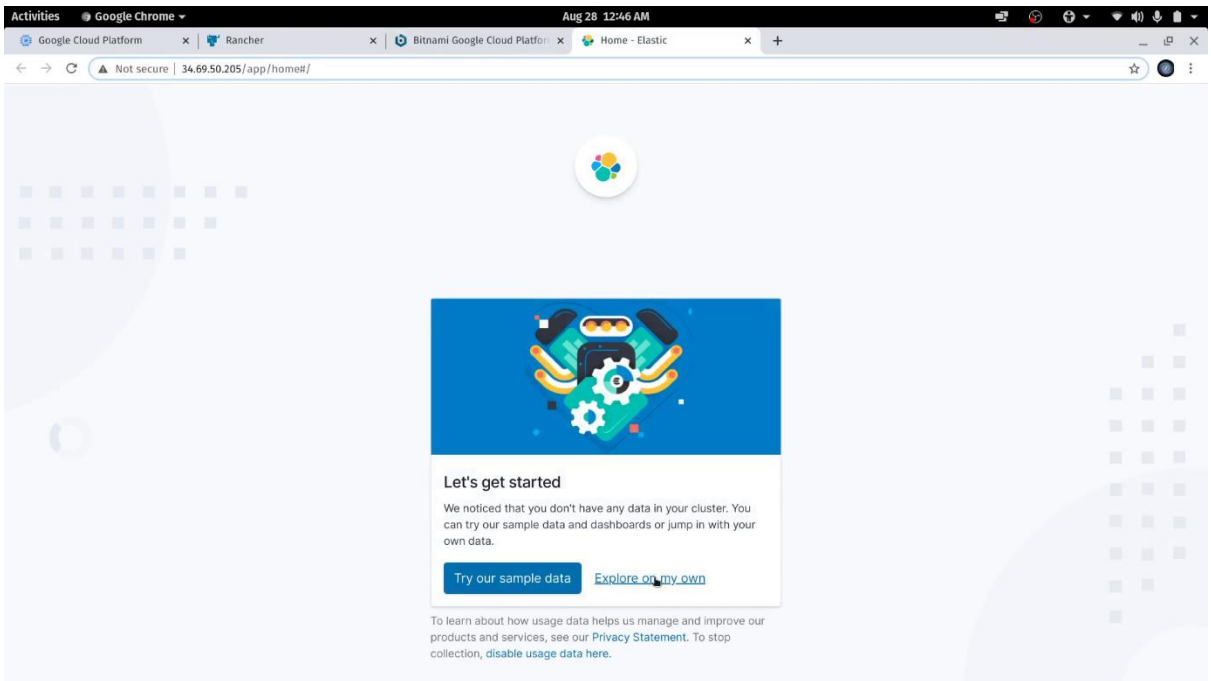
- Let's **TEST** whether the given endpoint is accessible, if not, check whether the tcp port 9200 was added to the firewall else create a new firewall rule and add the port.



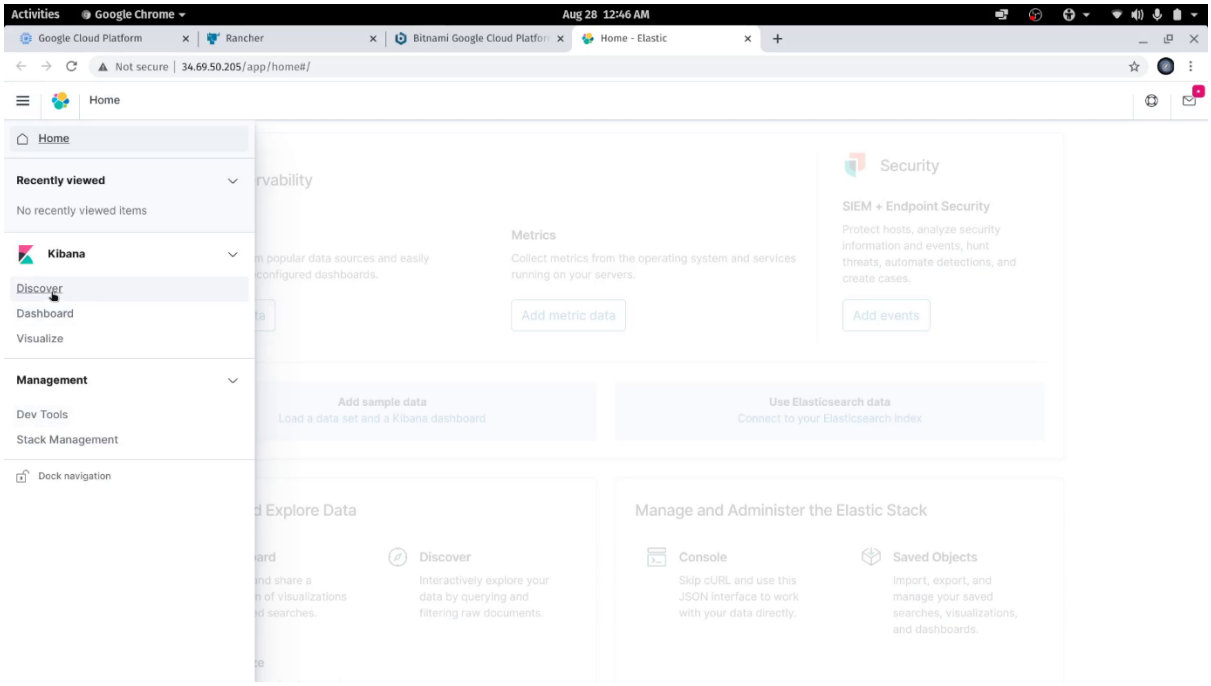
- If the **Settings validated** is displayed, click **Save**.



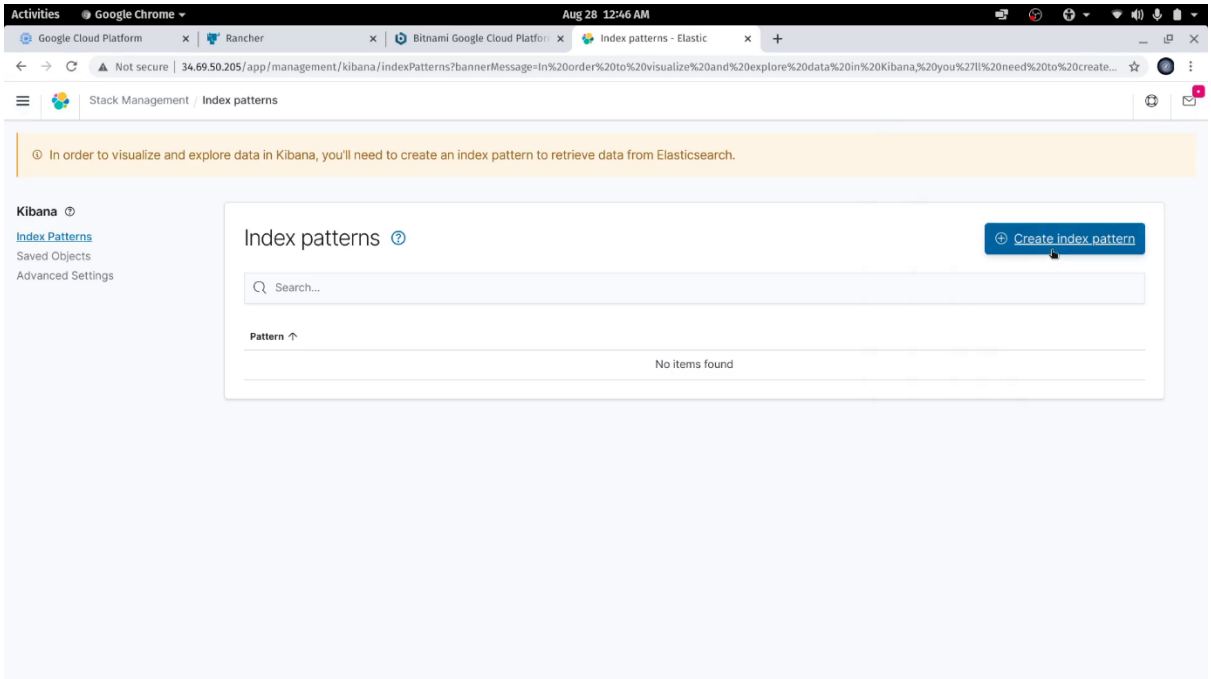
- Now head to the Kibana dashboard, select **Explore on my own**.



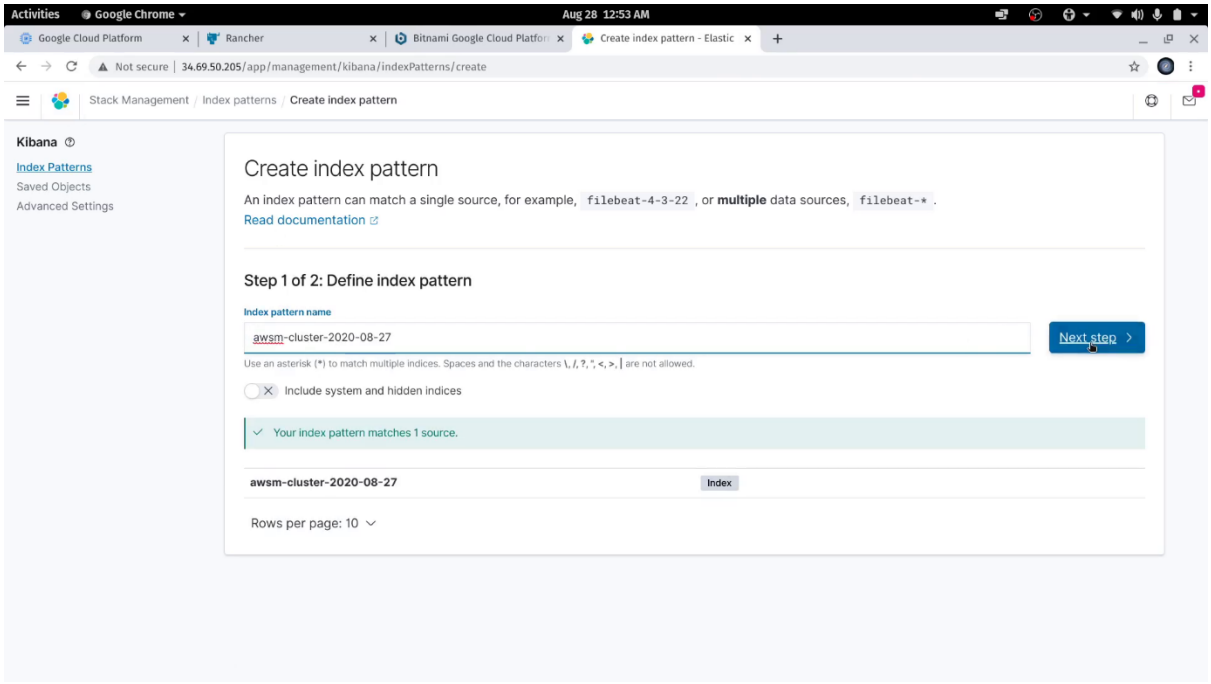
- Navigate to **Discover**.



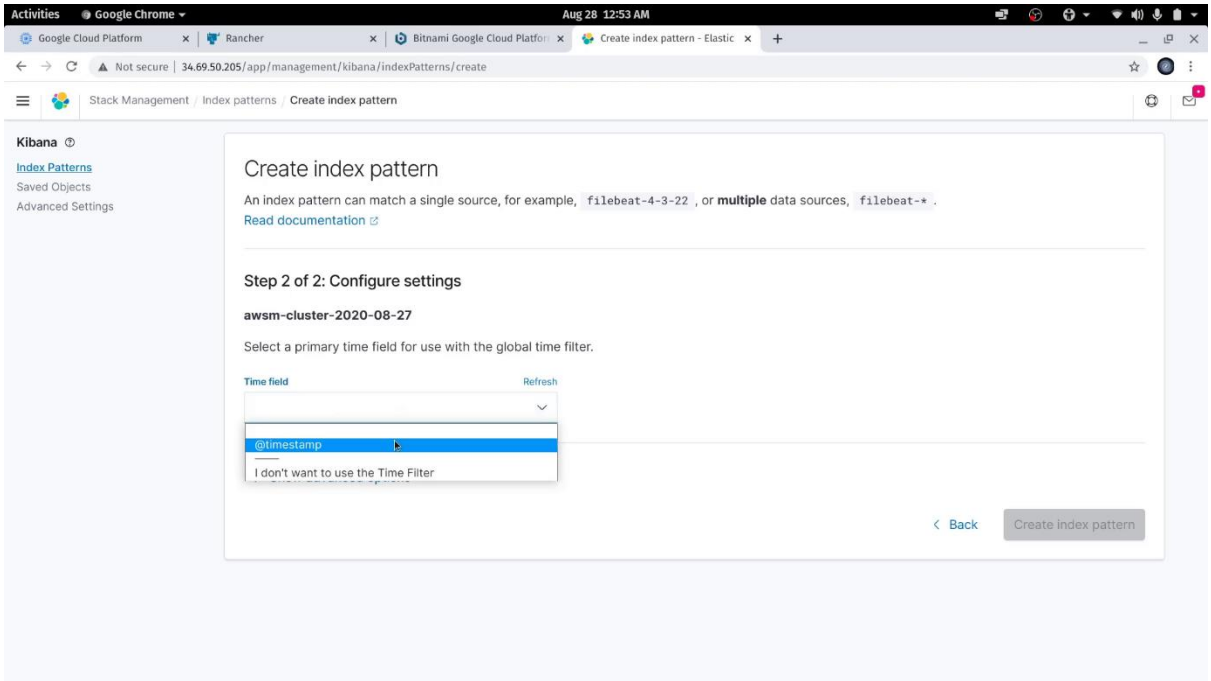
- Then, **Create index pattern**



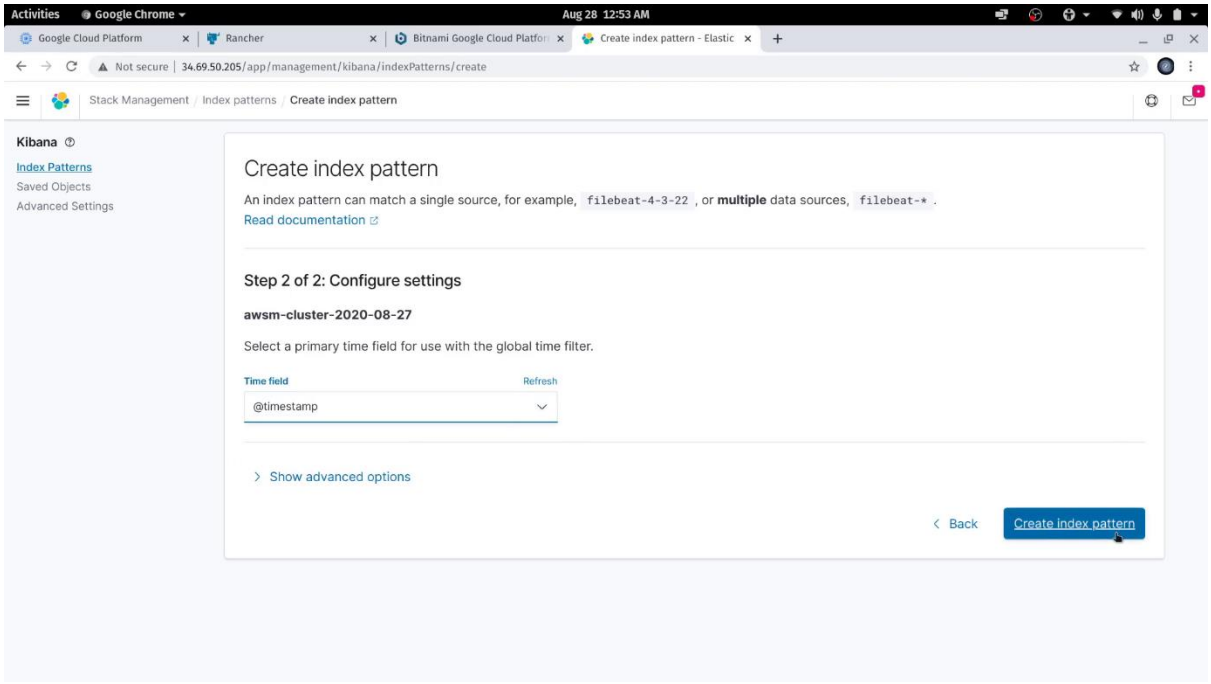
- Next, Define index pattern



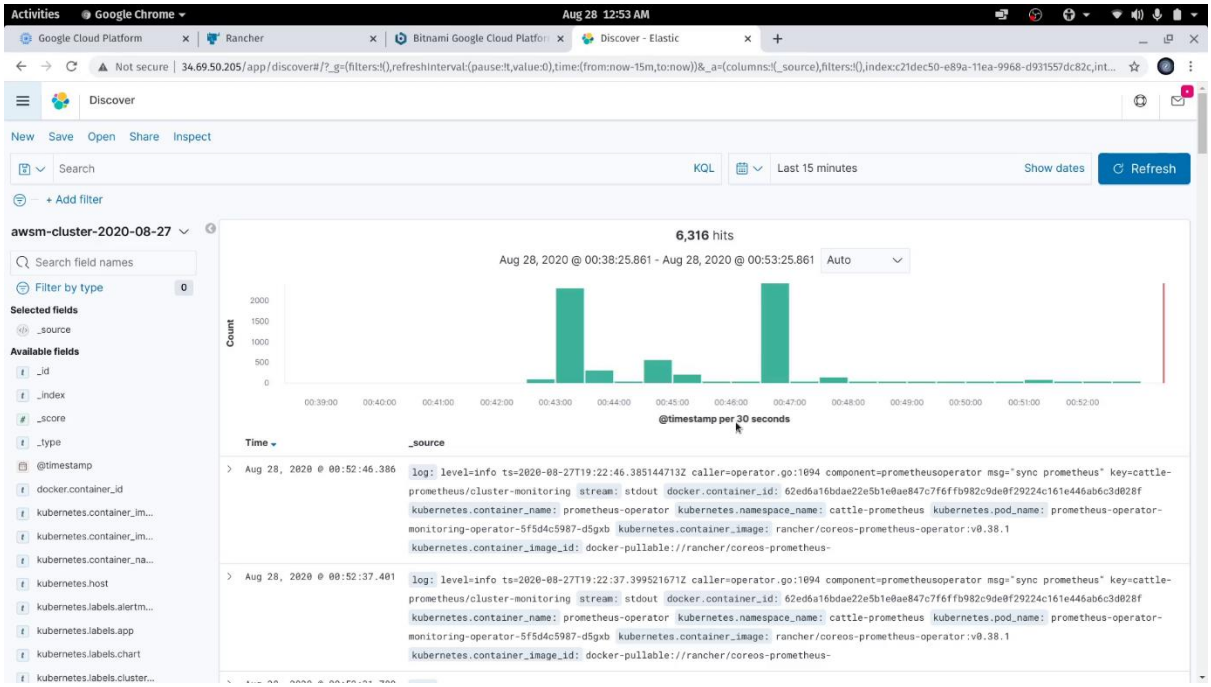
- Configure settings: **Time field > @timestamp**



- Then hit **Create index pattern**.

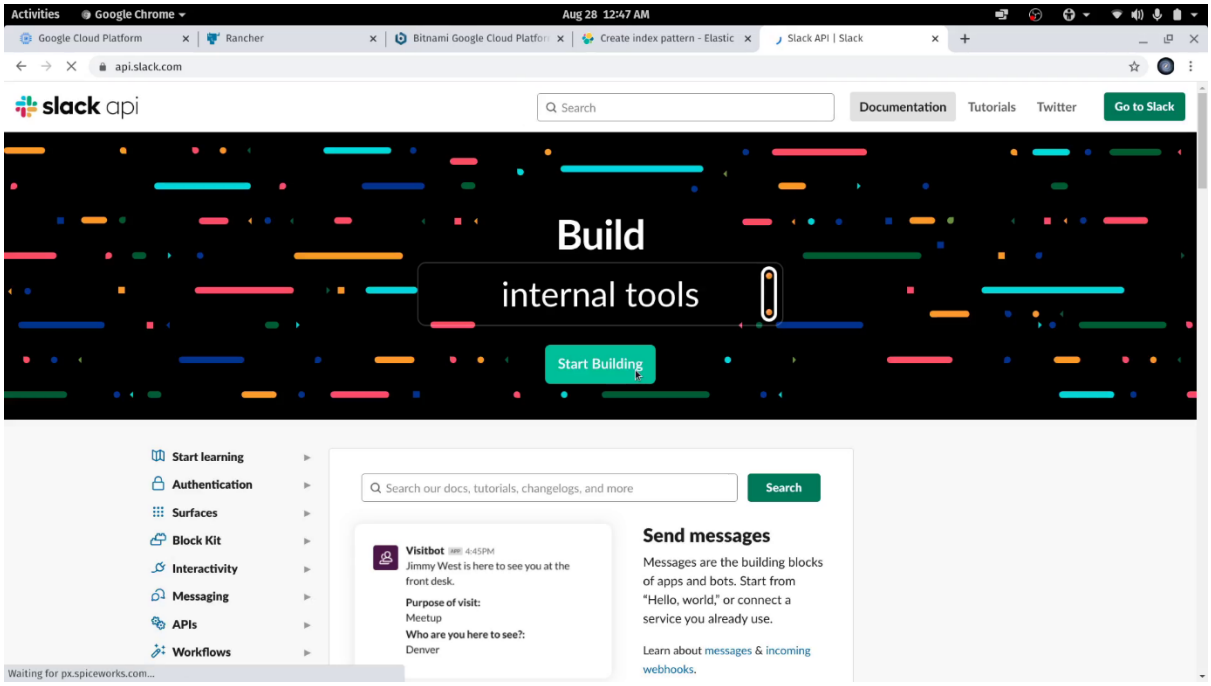


- Now, head back to **Discover**, we can see various logs with their respective timestamp.

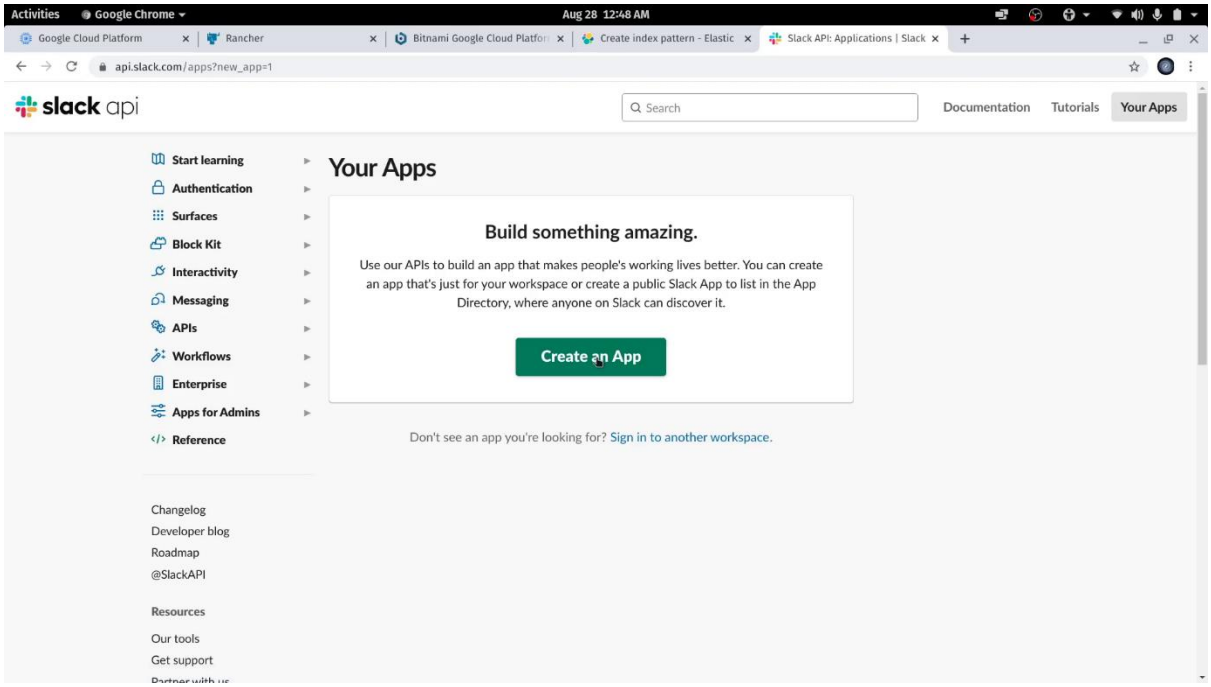


- We have now successfully configured ELK stack with our Kubernetes cluster.

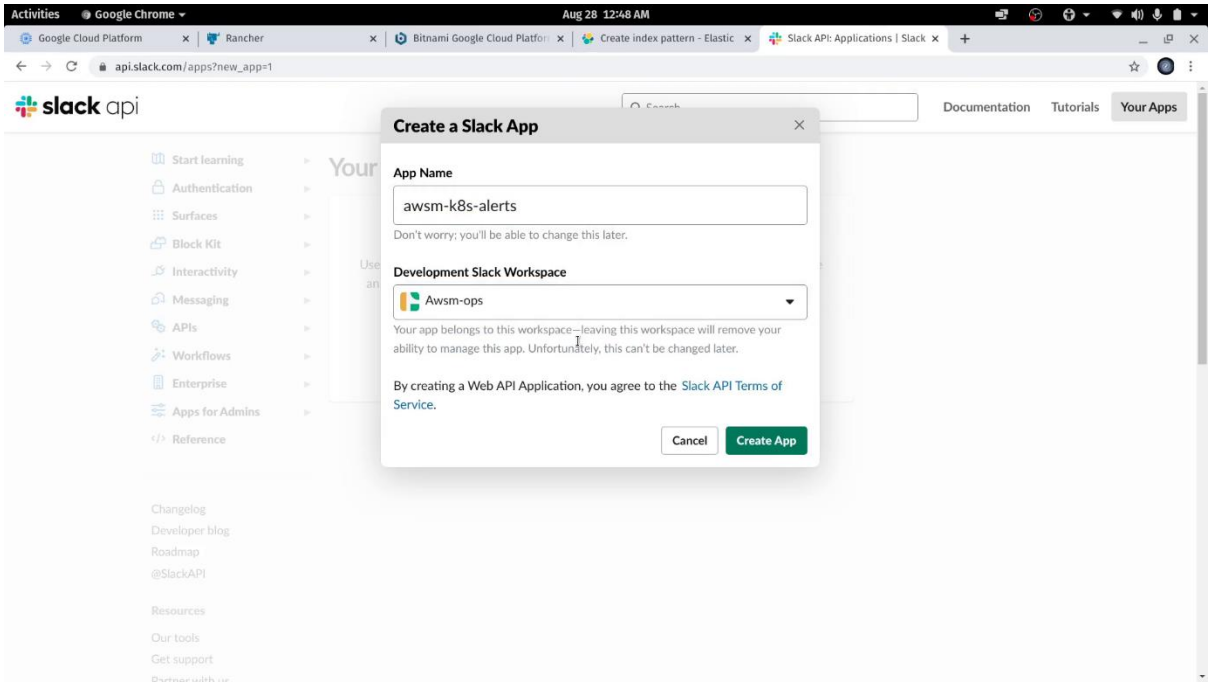
14. Now we will setup Slack for notifications regarding our cluster alerts.
    - Create a Slack workspace.
    - Create a channel.
    - Then go to https://api.slack.com and click **Start Building**.



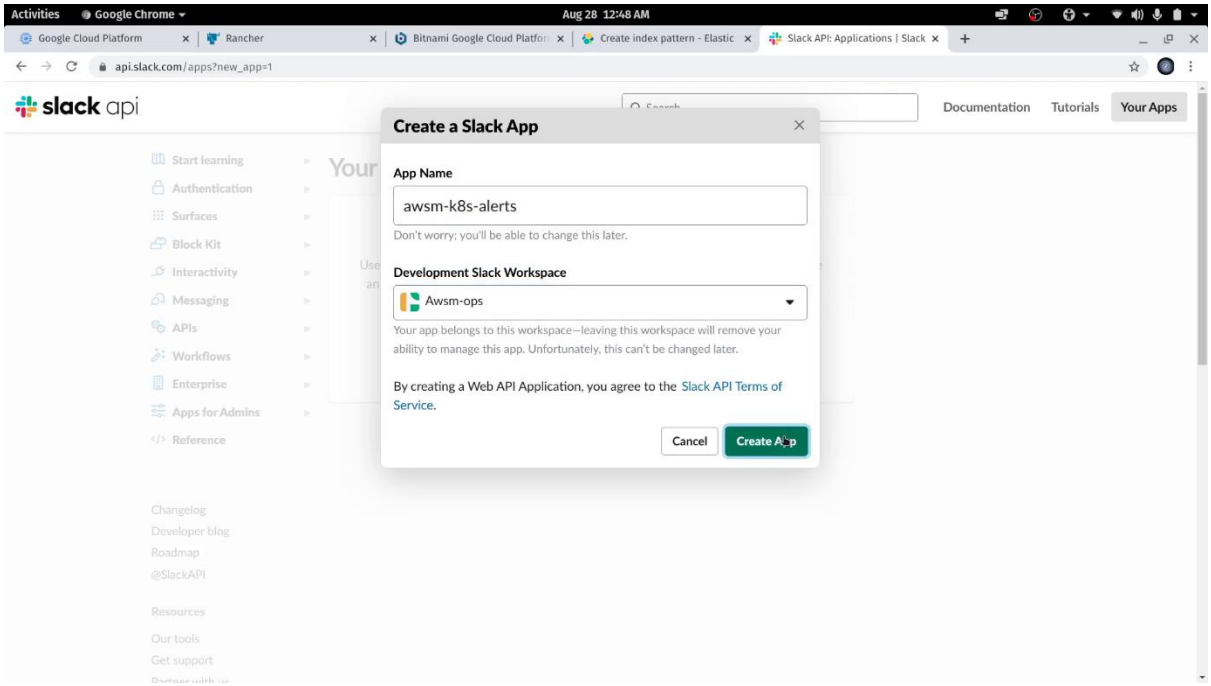
    - Click on **Create an App**.



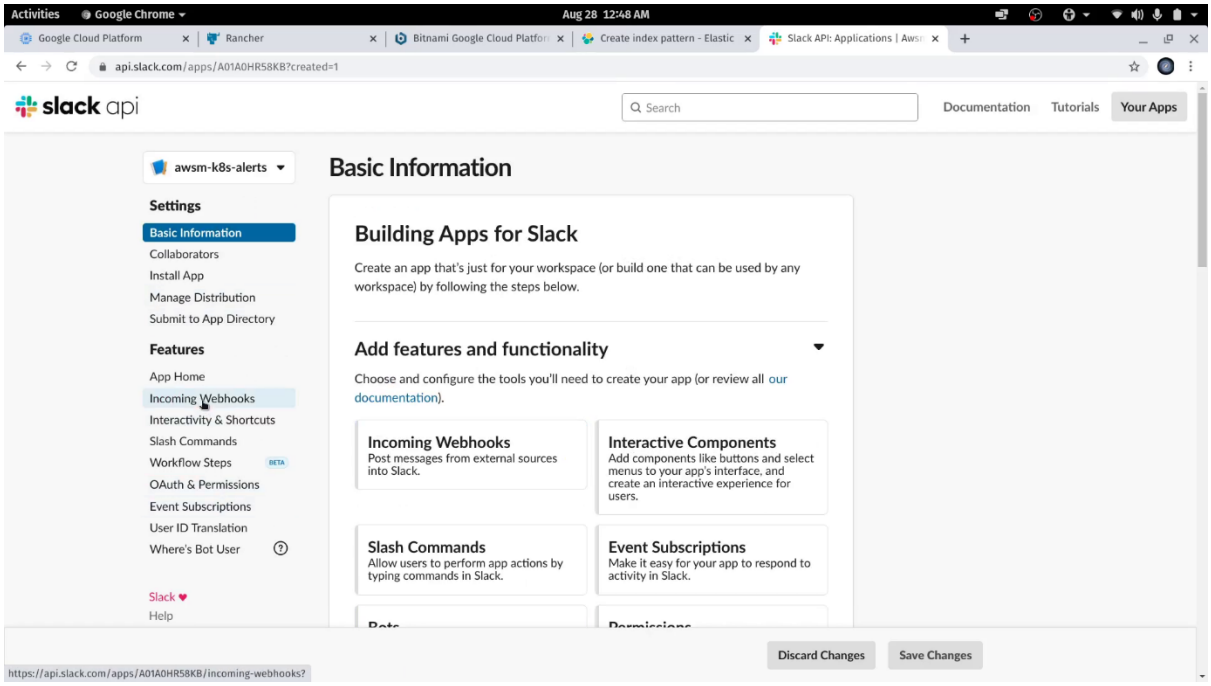
- Enter an **App Name** and select a **Development Slack Workspace**.



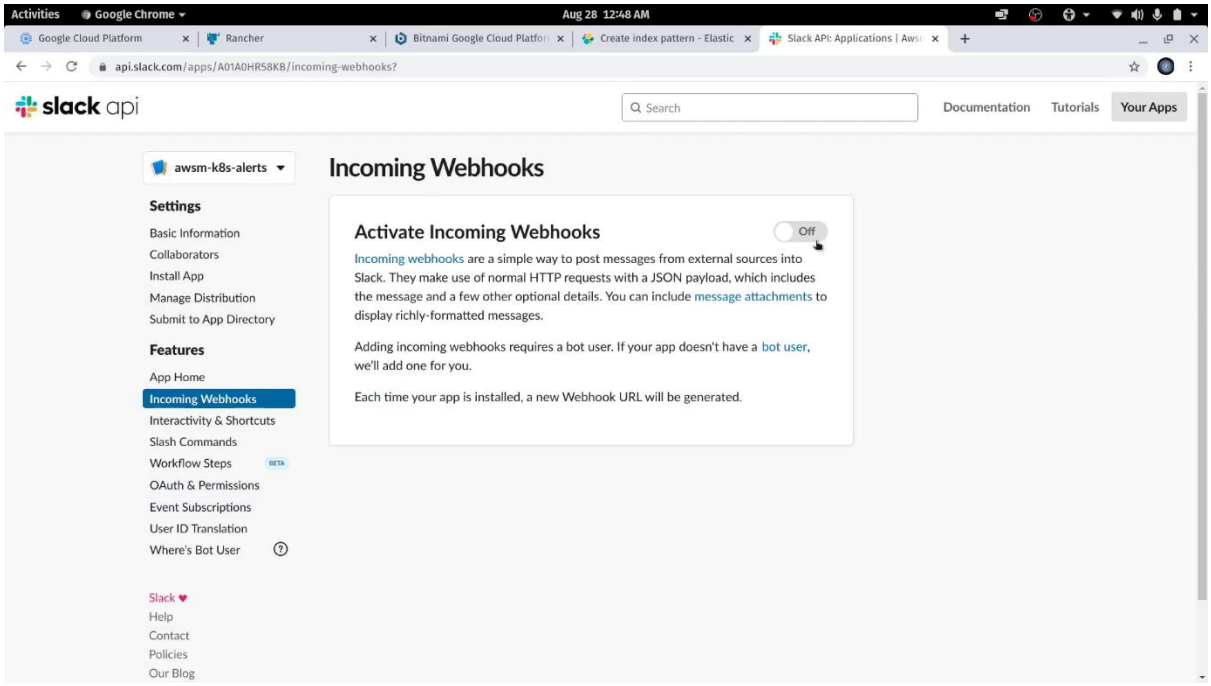
- Then click **Create App**.



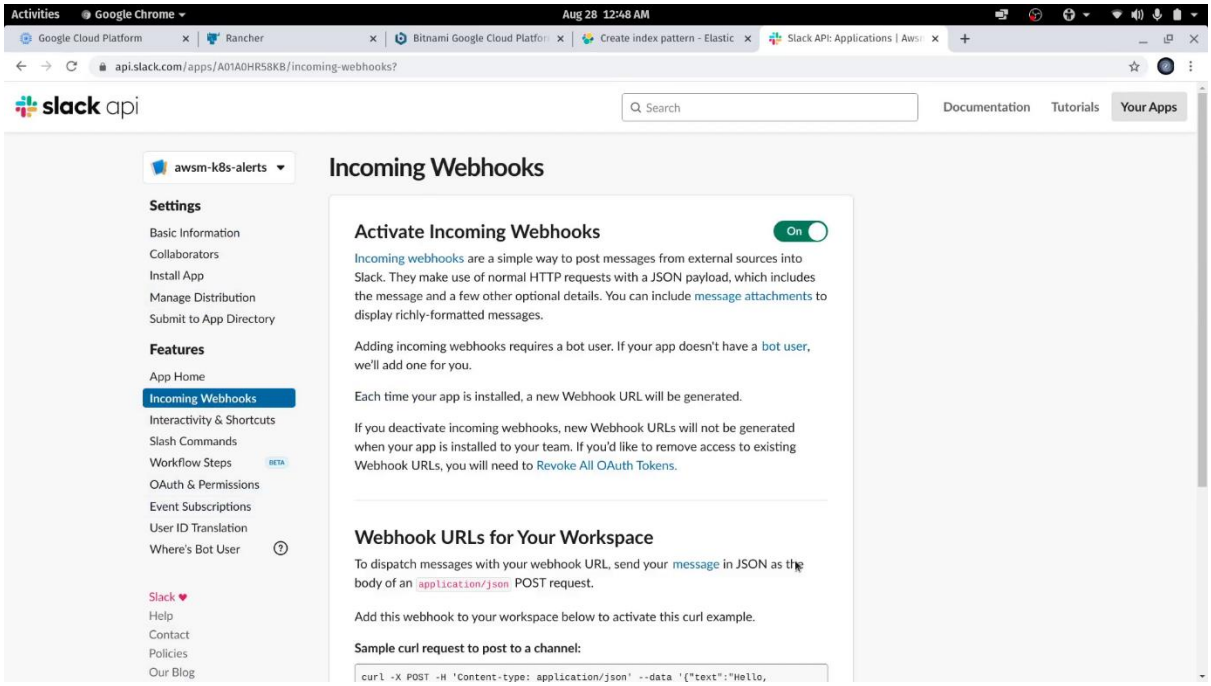
- Click **Incoming Webhooks** under Features.



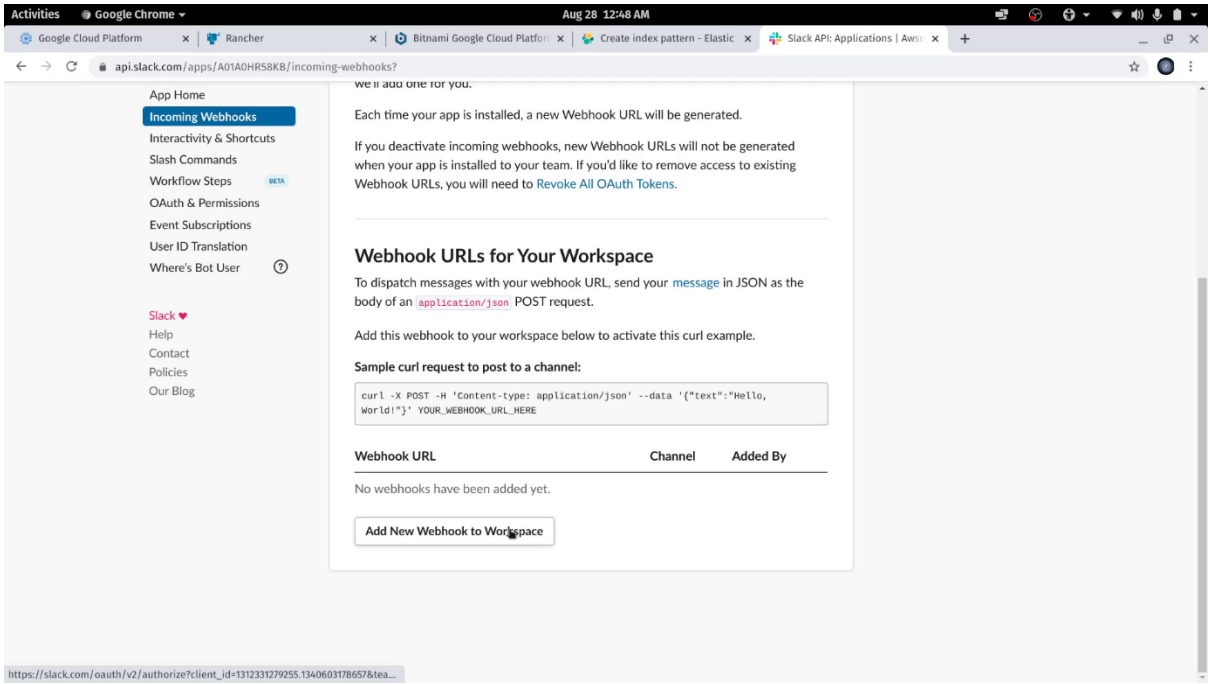
- **Activate Incoming Webhooks**



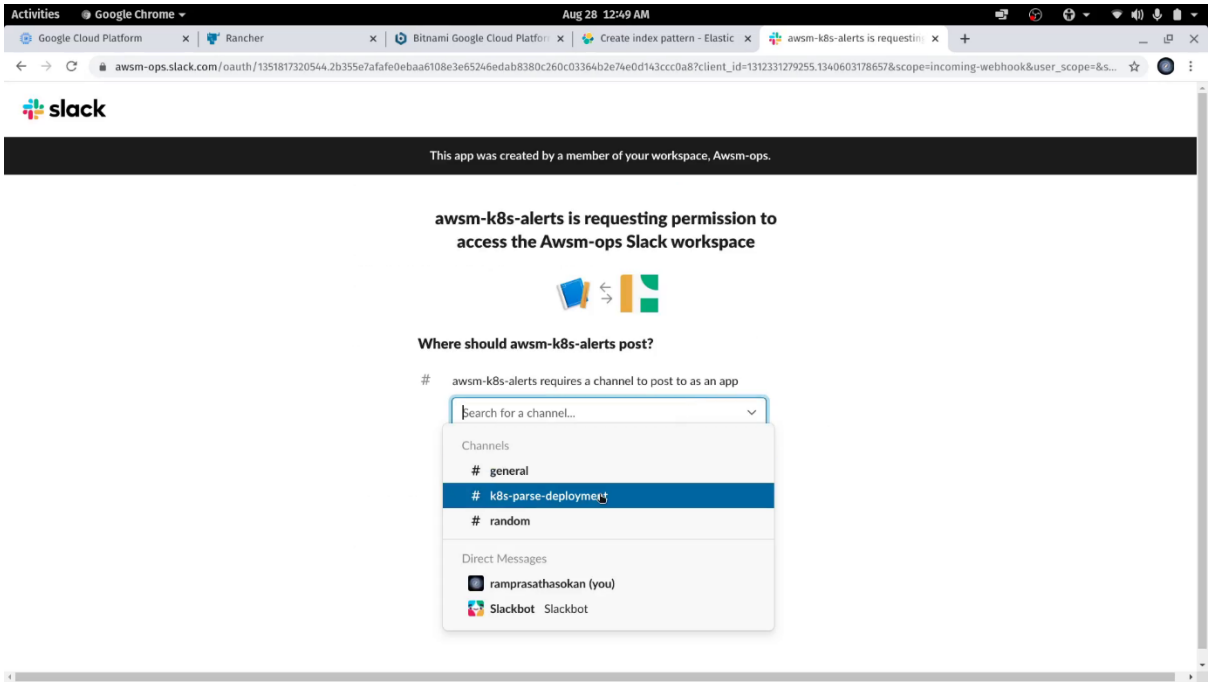
- Once Incoming Webhooks are activated, proceed to the next step.



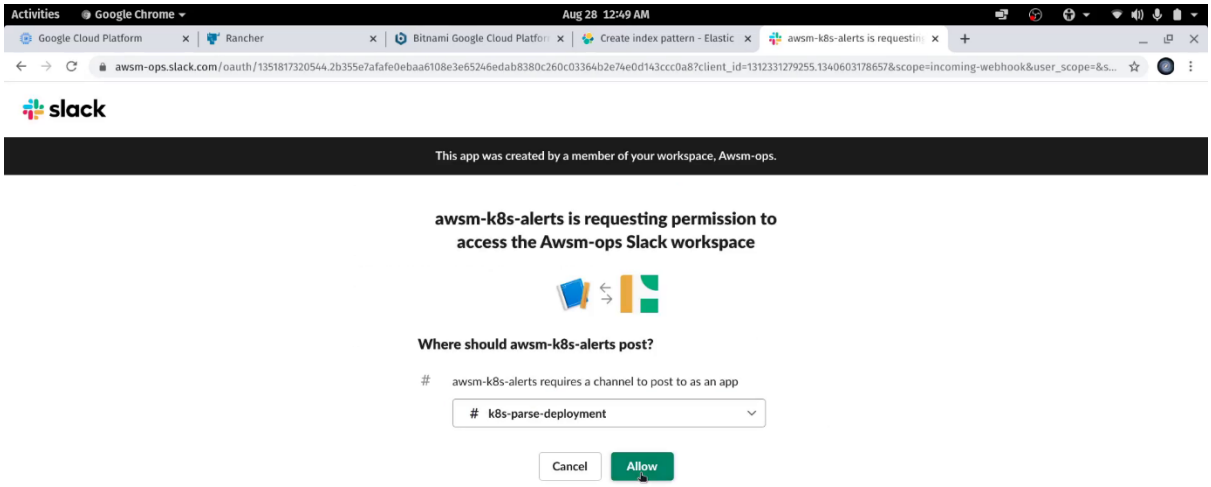
- **Add New Webhook to Workspace**



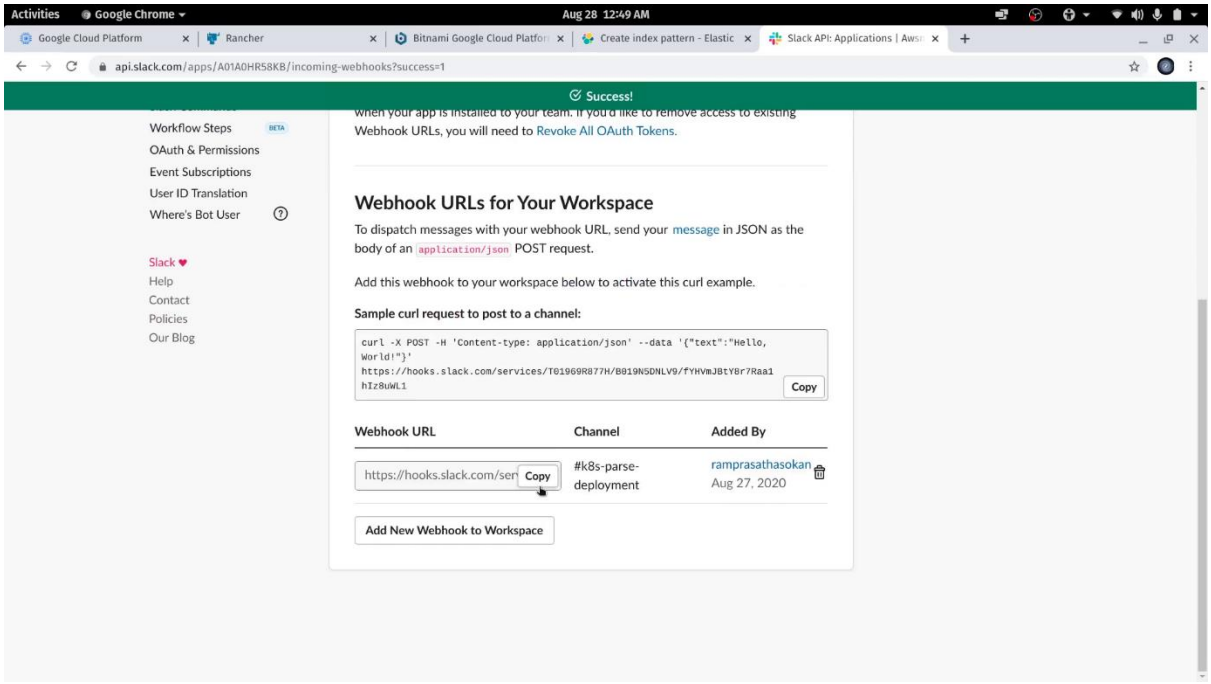
- **Select for a channel** to post alerts.



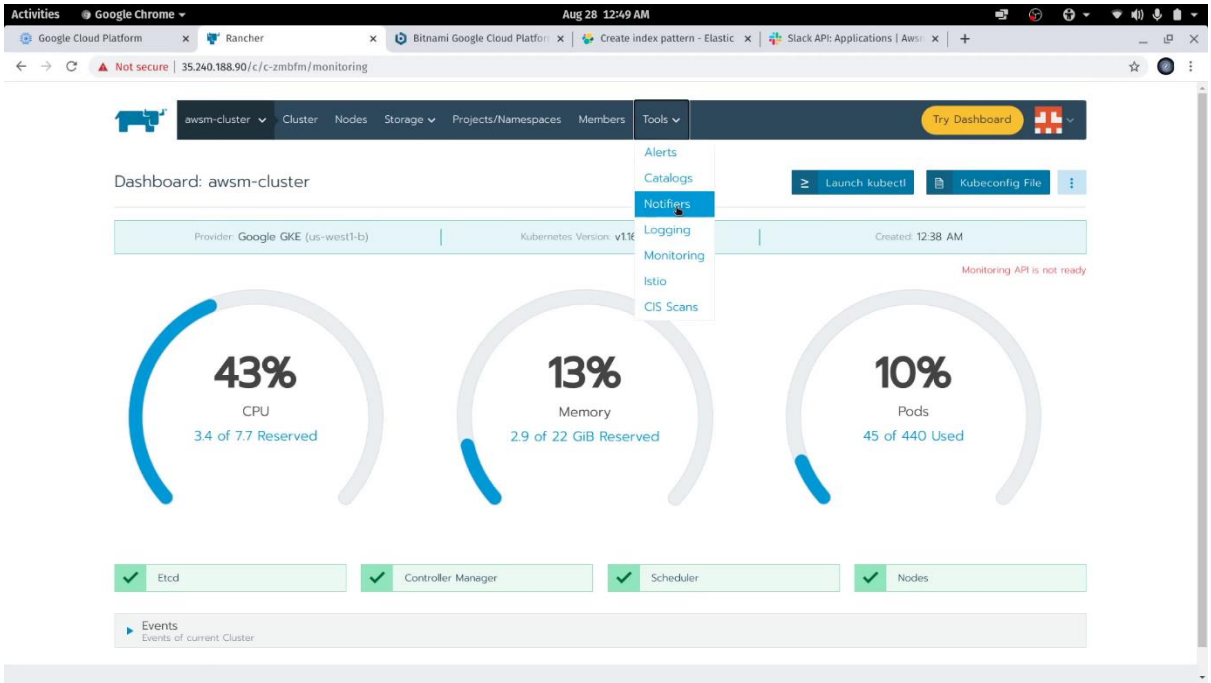
- Now click **Allow**.



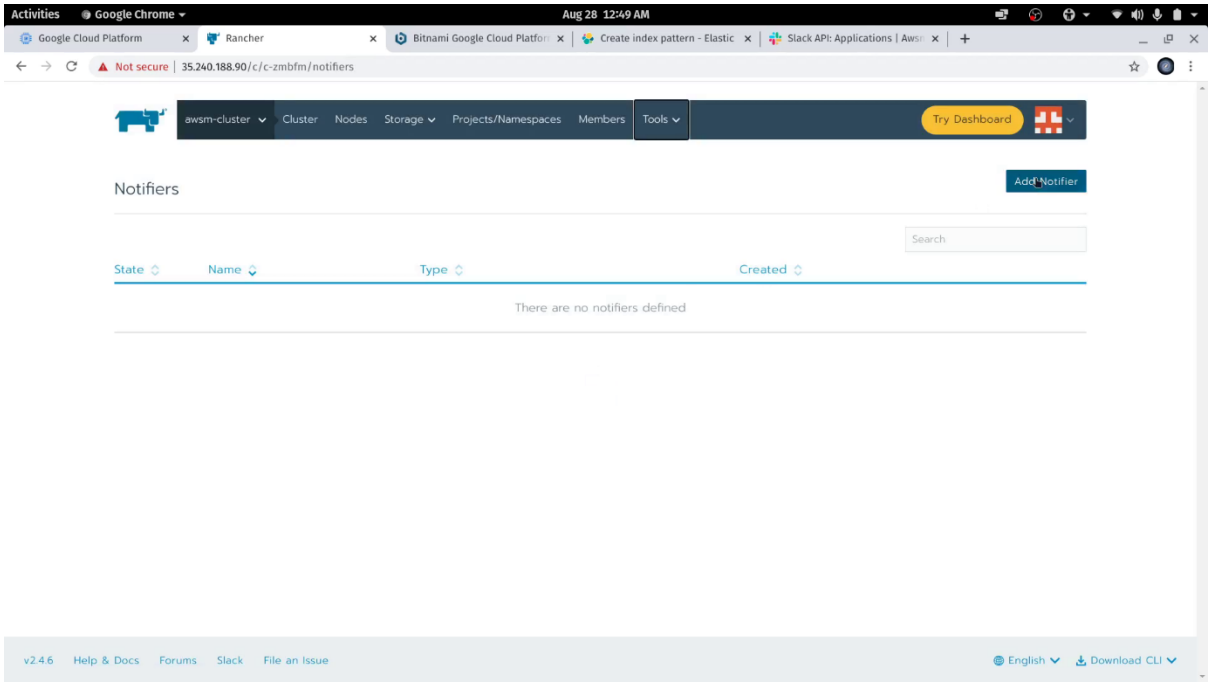
- Once the Webhook is created, copy the Webhook URL.



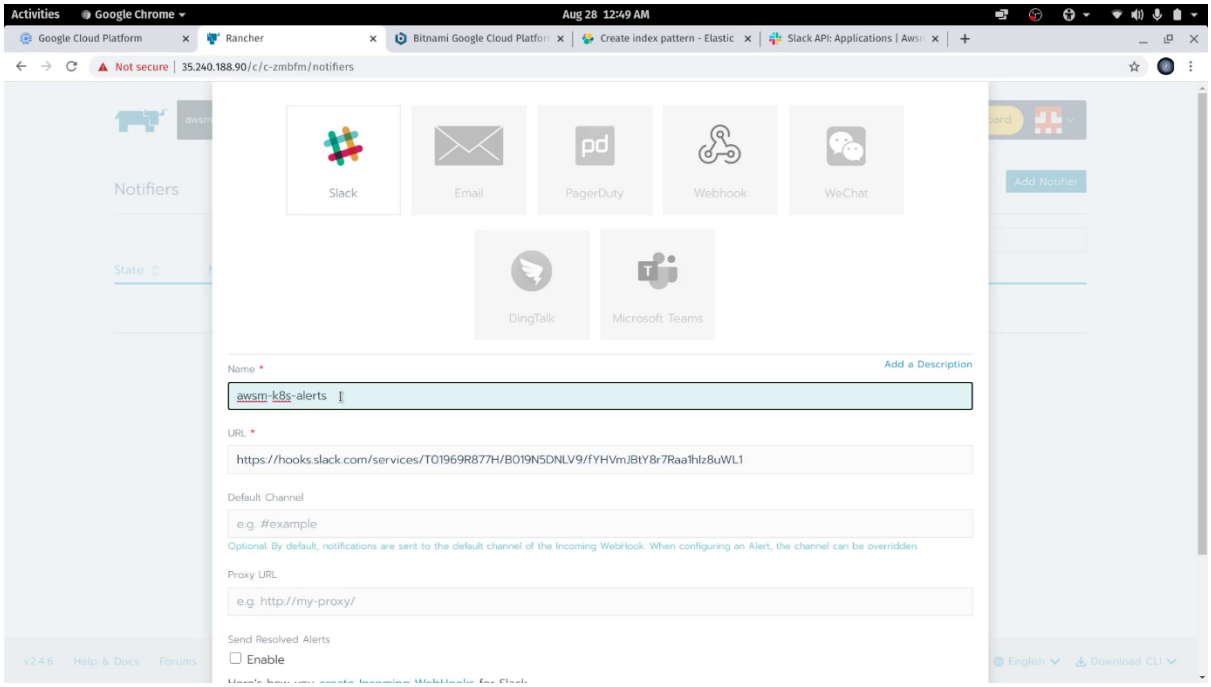
- Now let's head to **Rancher > Cluster > awsm-cluster > Tools > Notifiers**



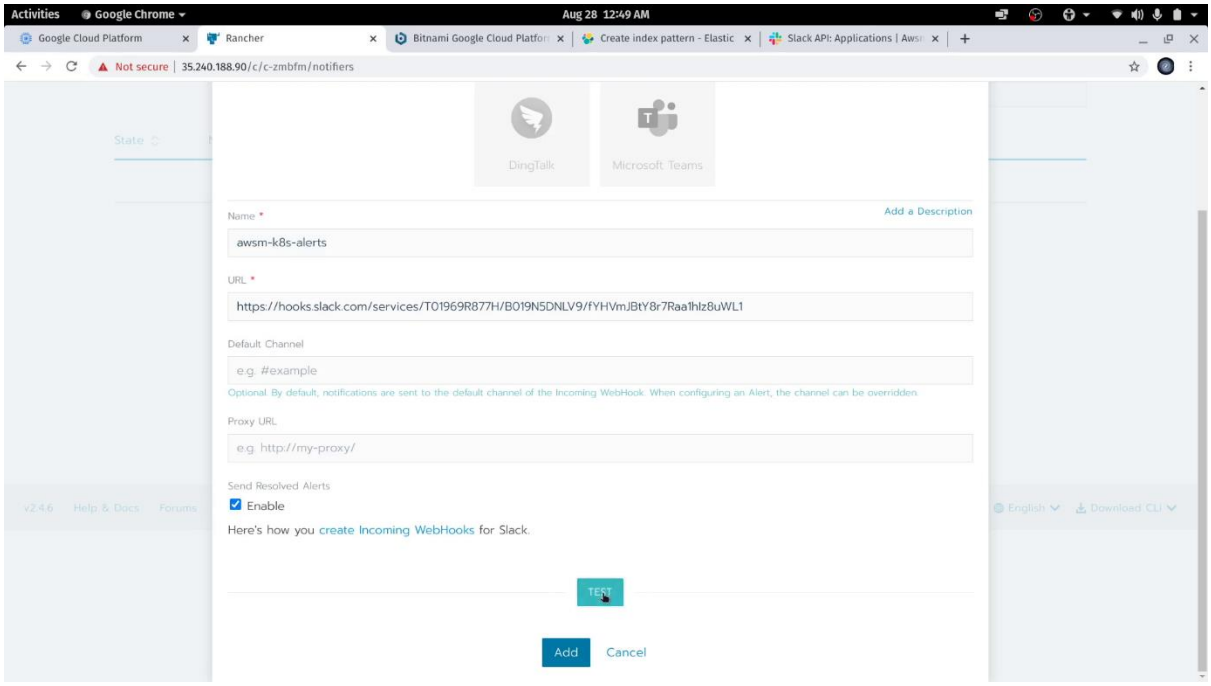
- Click **Add Notifier**



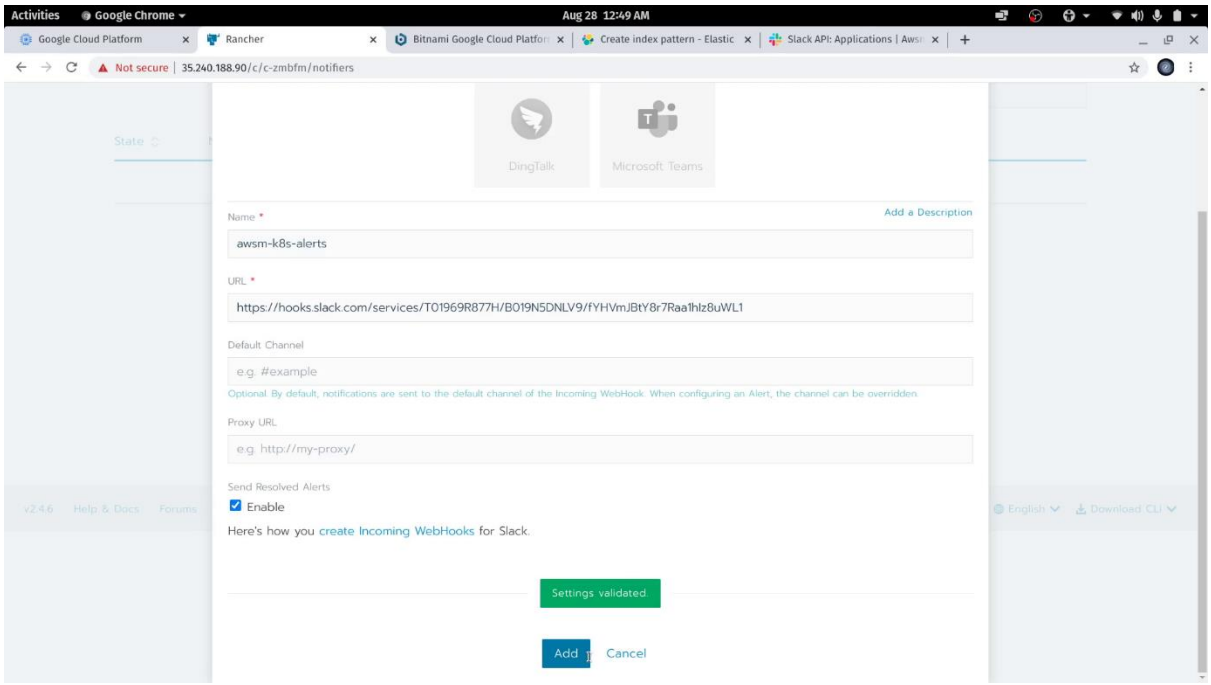
- Enter Webhook URL under **URL** and a **Name**.



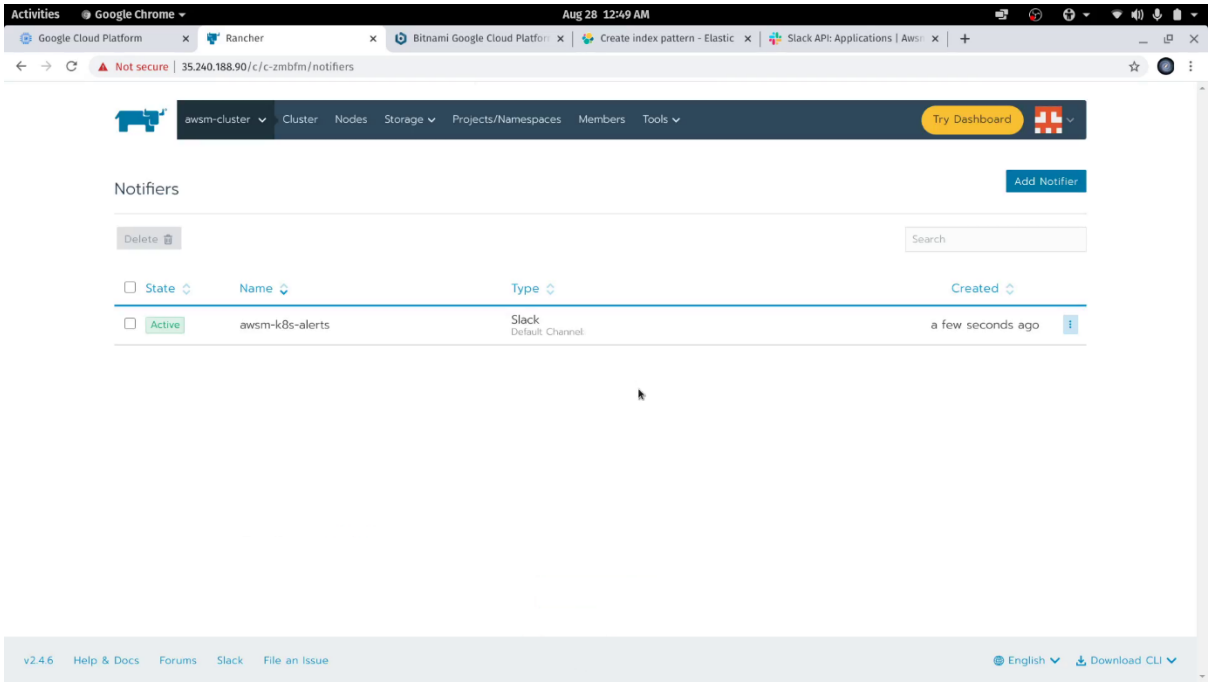
- Let's **TEST** whether it's working.



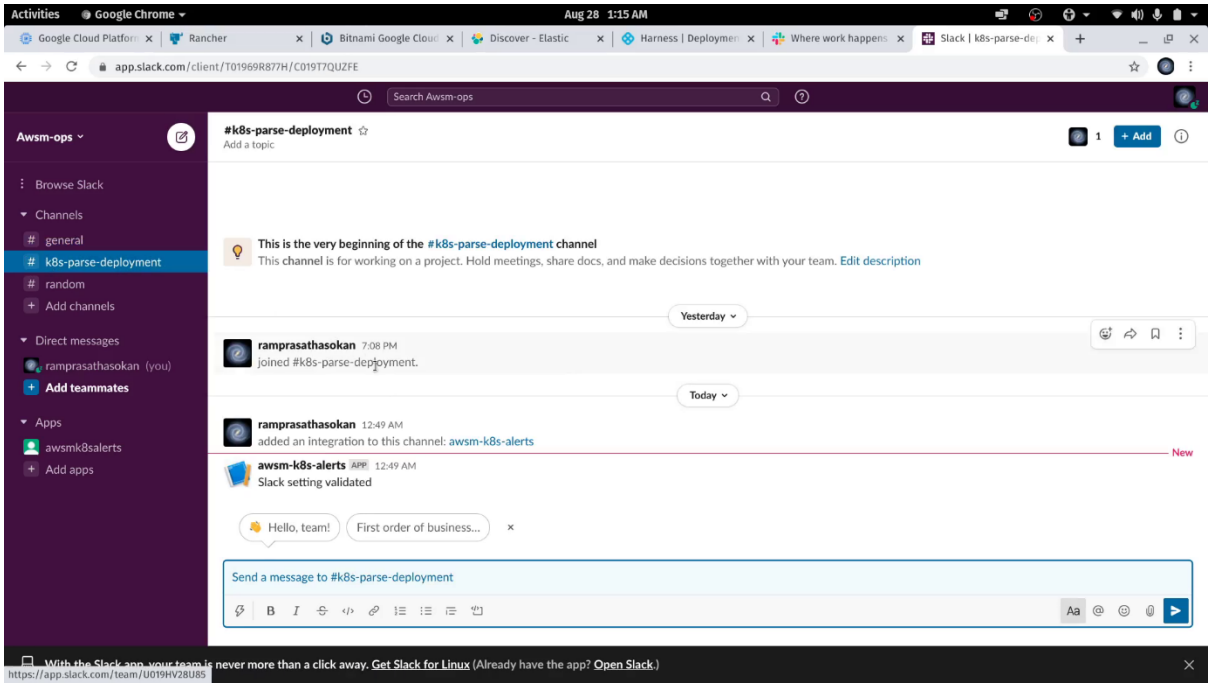
- If **Settings validated** is displayed, click **Add**.
- Note: The default channel is the channel linked to the incoming webhook.



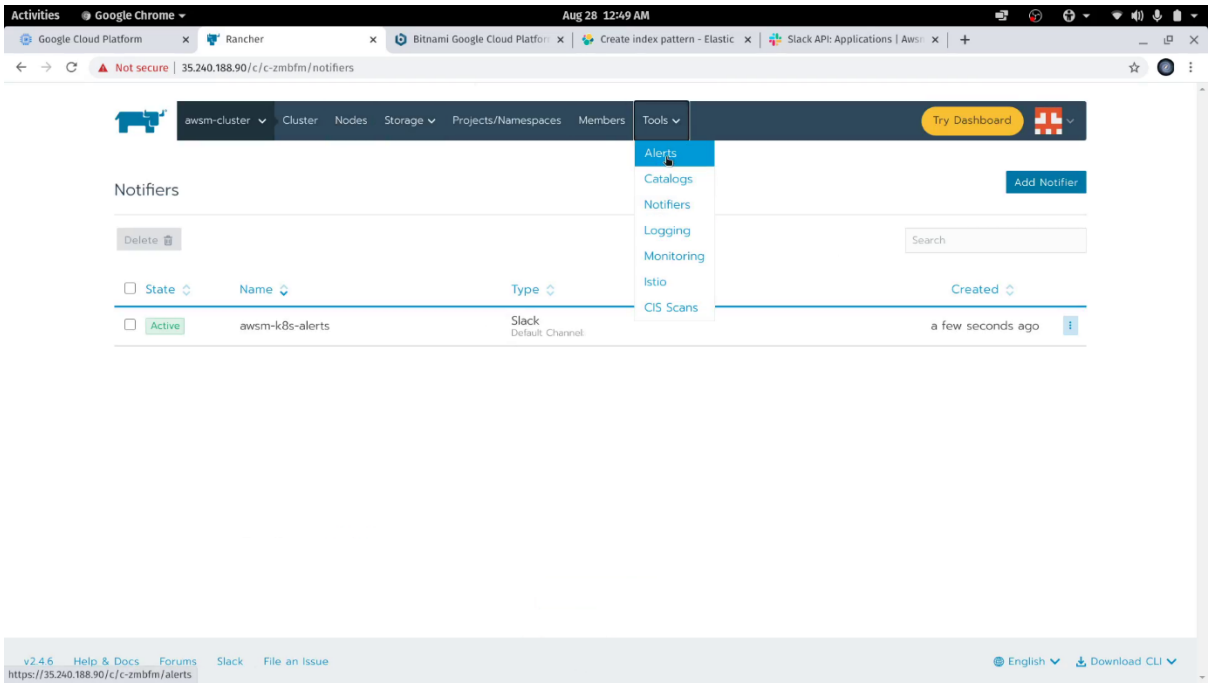
- We can now see that our Slack notifier has been successfully added.



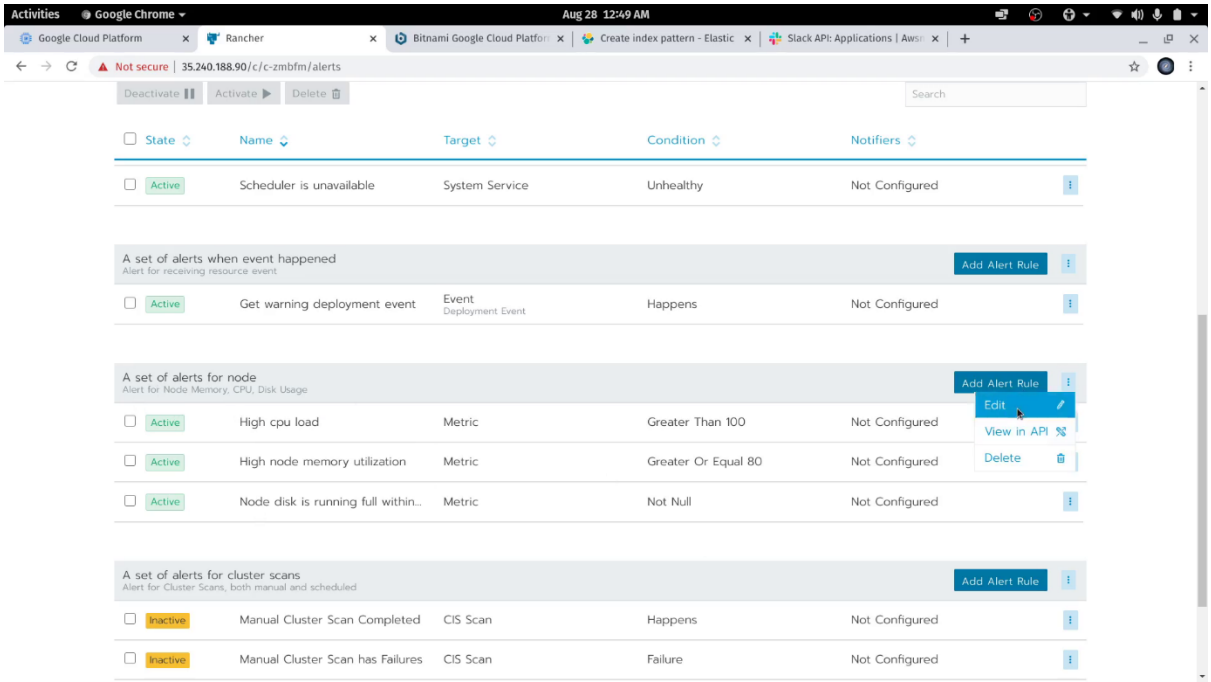
- Verification from Slack workspace



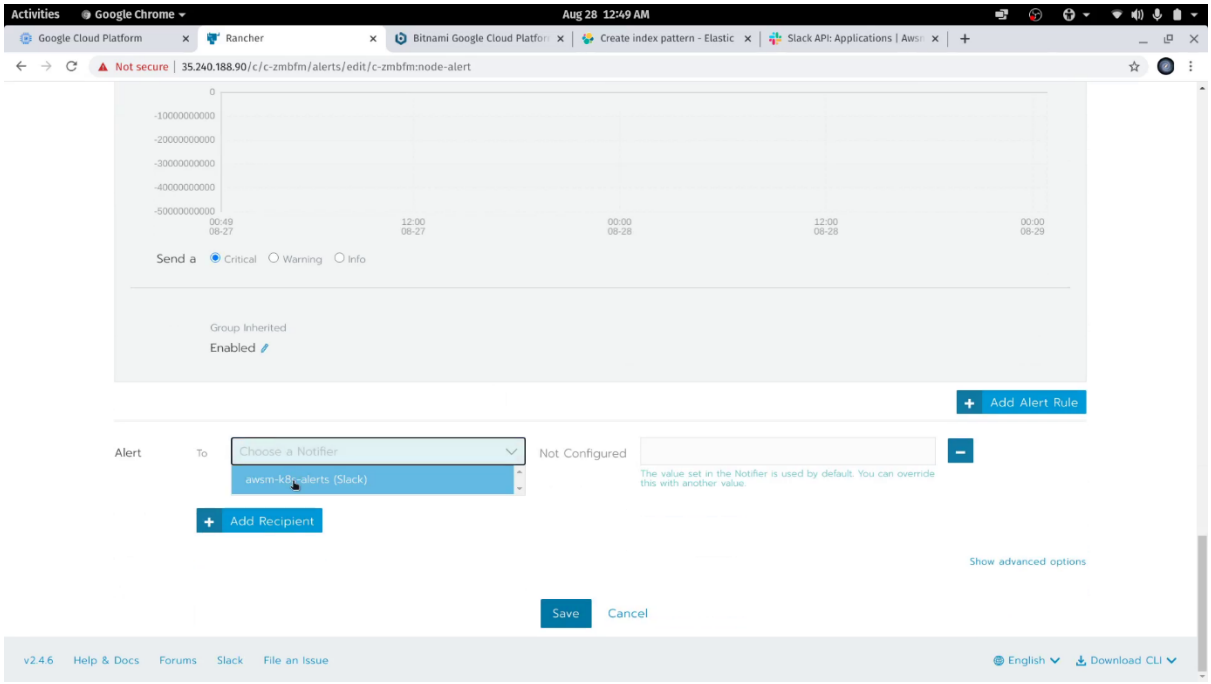
- Next, we need to choose the alerts to be notified of. To do so, go to **Rancher > Cluster > awsm-cluster > Tools > Alerts**



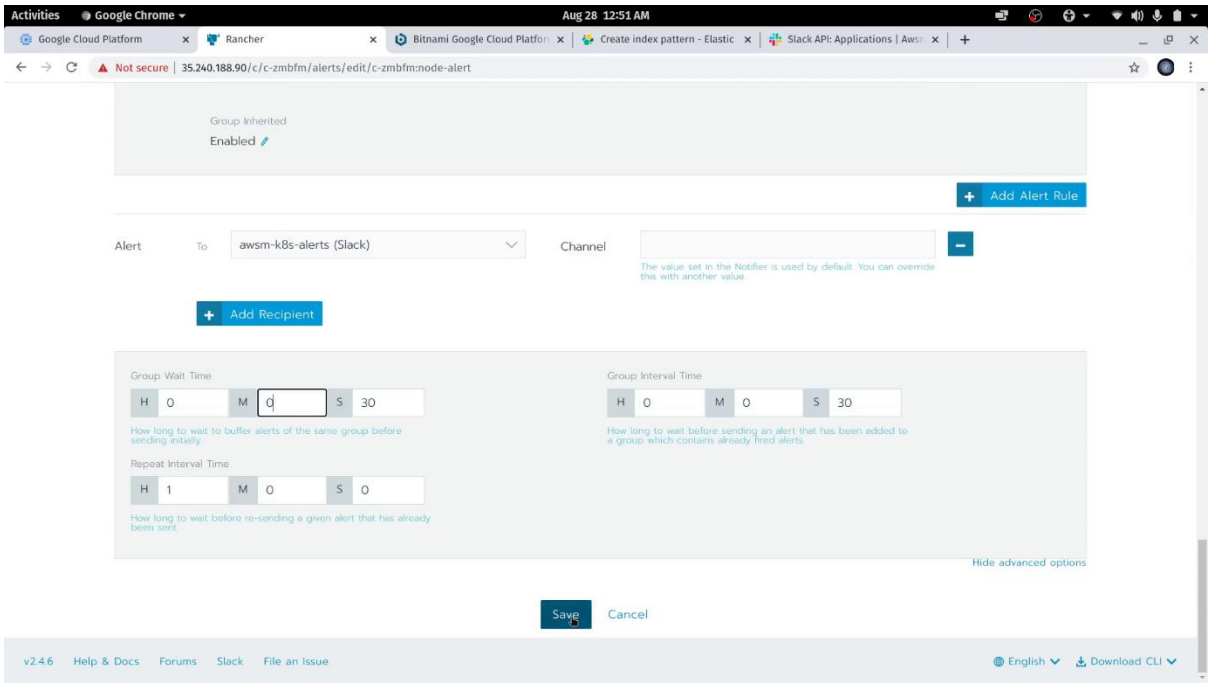
- Then select an alert group to be notified of and click **Edit**.



- Now choose our newly added Slack notifier.



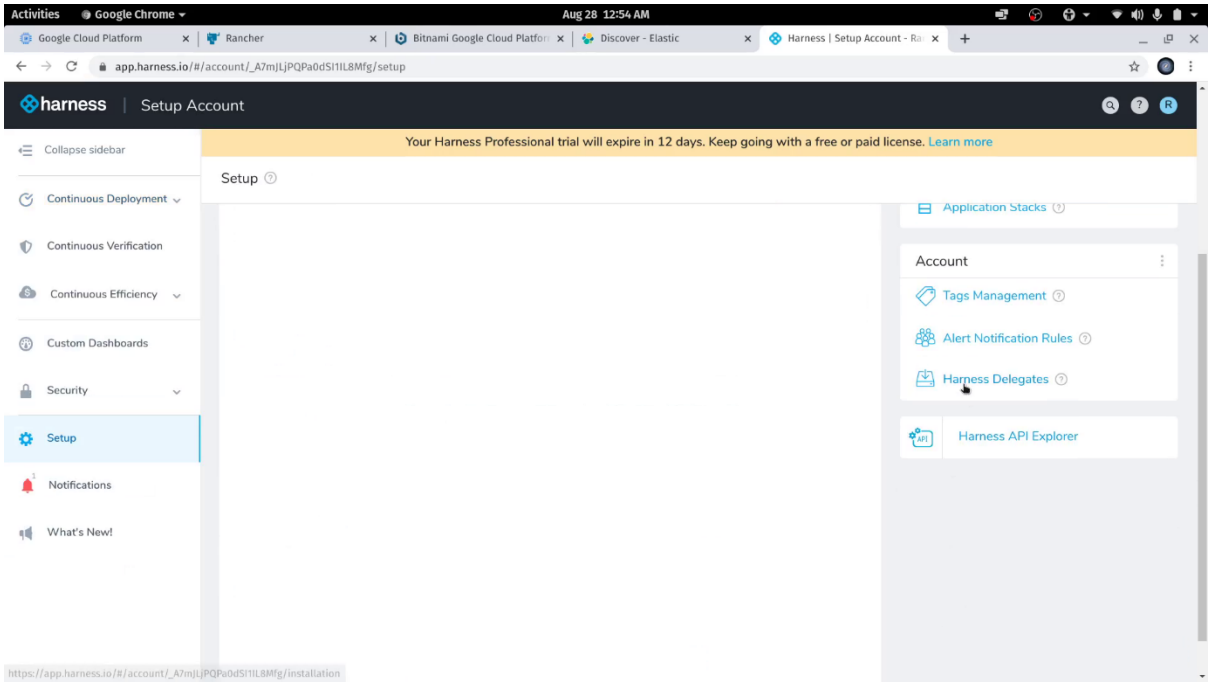
- Then click **Show advanced options** to configure **Group Wait Time** and **Group Interval Time**.
- Note: Group Wait Time is the duration of the interval to buffer alerts from the same group.
- Note: Group Interval Time is the duration taken to send the first alert to the notifier since the time of addition of the group.
- After configuring, click **Save**



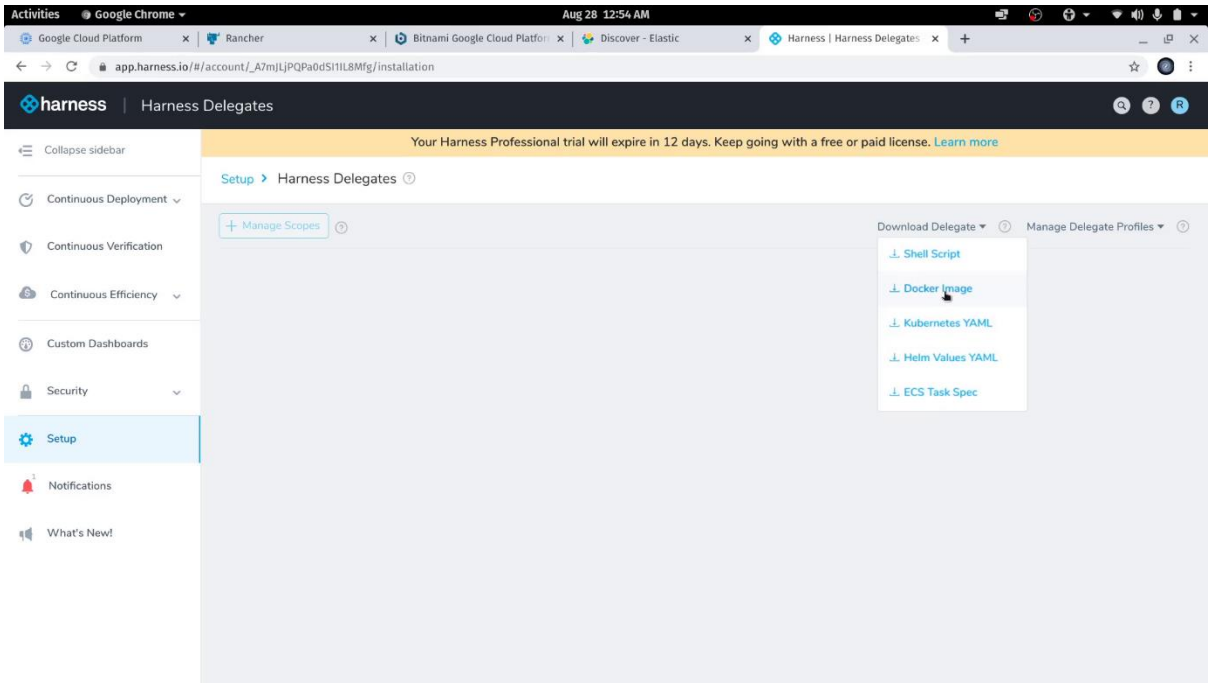
- We have now successfully setup the Slack notifier.

15. Now let's setup CD pipeline with Harness using https://harness.io

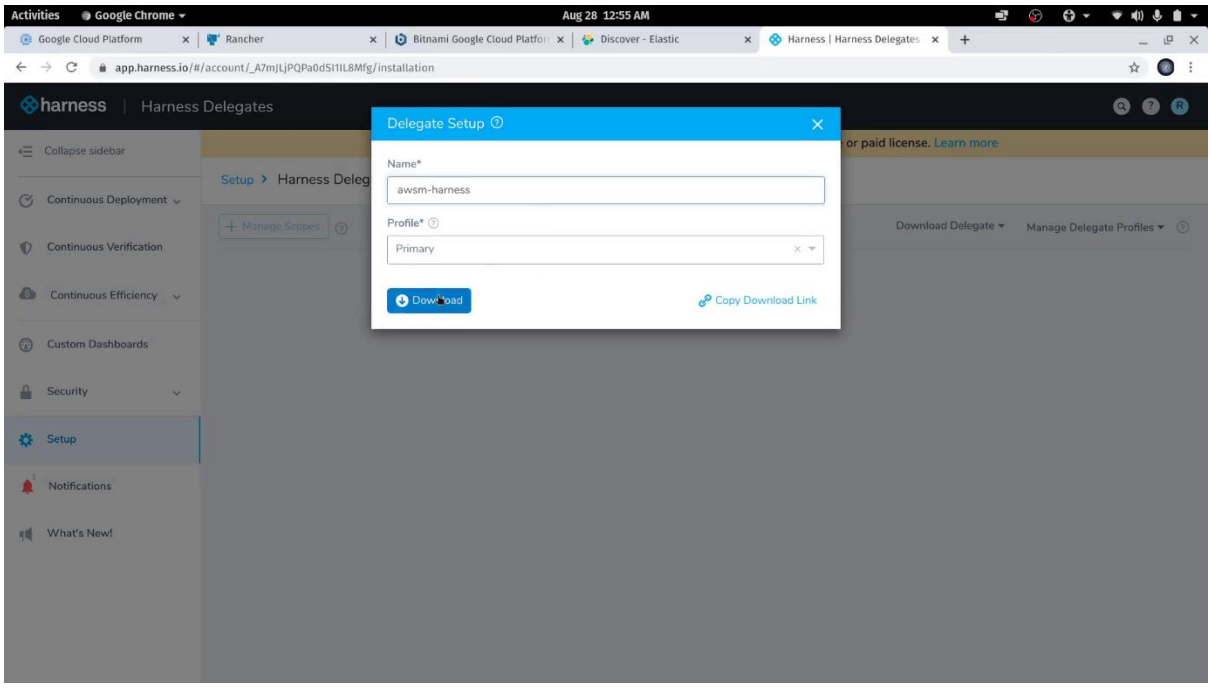
- First, we need to setup Harness delegate. Now let's see why we need Harness delegate: The Harness Delegate is a service you run in your deployment target environment, such as your local network, VPC, or cluster. The Delegate connects all of your artifact, infrastructure, collaboration, verification and other providers with the Harness Manager. Most importantly, the Delegate performs all deployment operations.
- After signing in, go to **Setup > Harness Delegates**



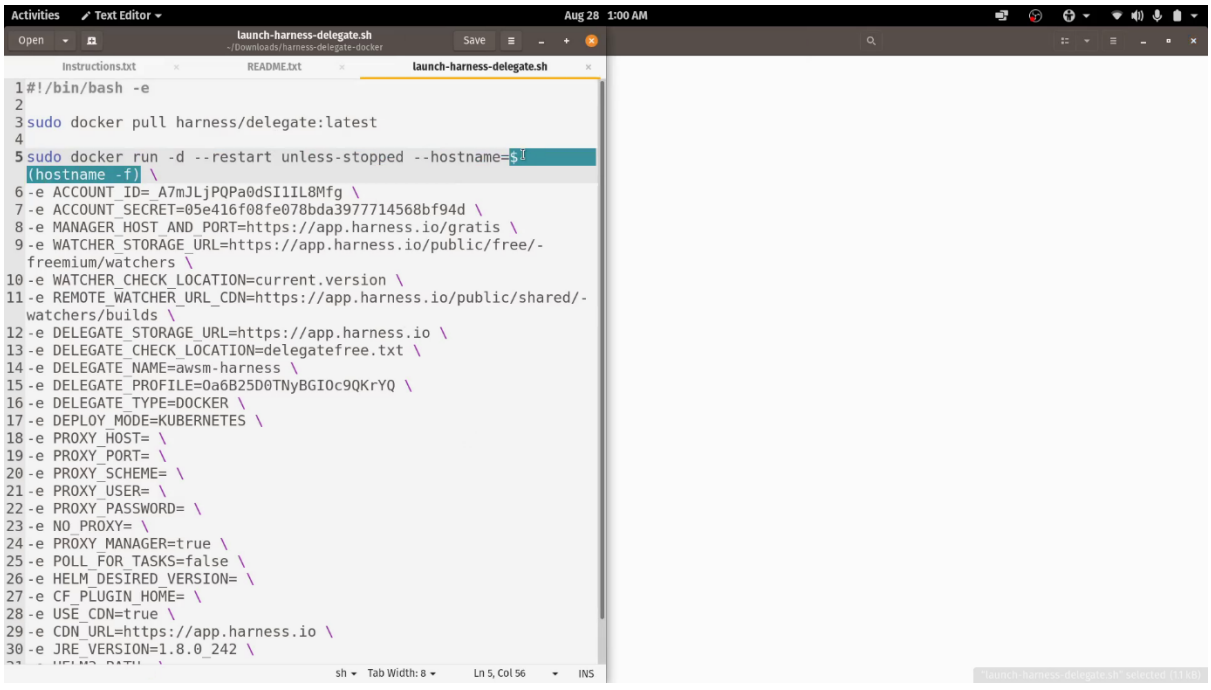
- Download delegate as Docker Image but for actual use case download as Kubernetes YAML or Helm Values YAML and run the delegate in a Kubernetes environment.



- Set profile as Primary as of now and download, we can change it later once the delegate is setup.

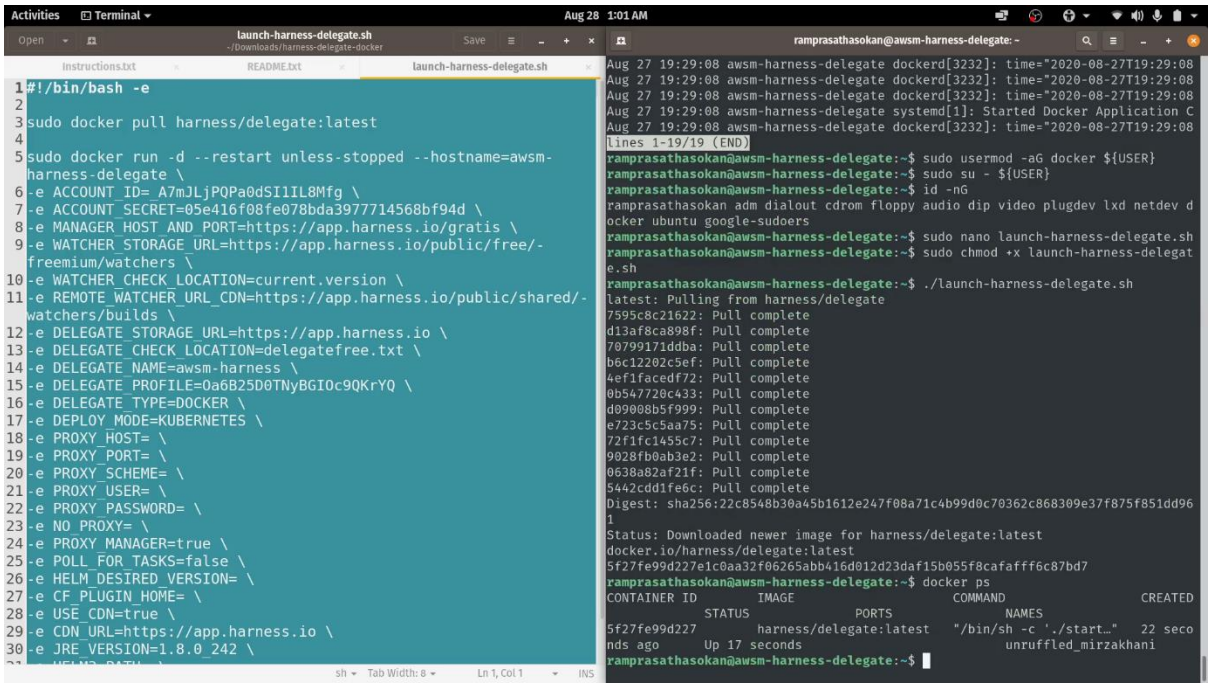


- After downloading the delegate setup, we need to extract it and open the shell script named launch-harness-delegate.sh in a text editor and change the *--hostname* value to the name of the VM instance.

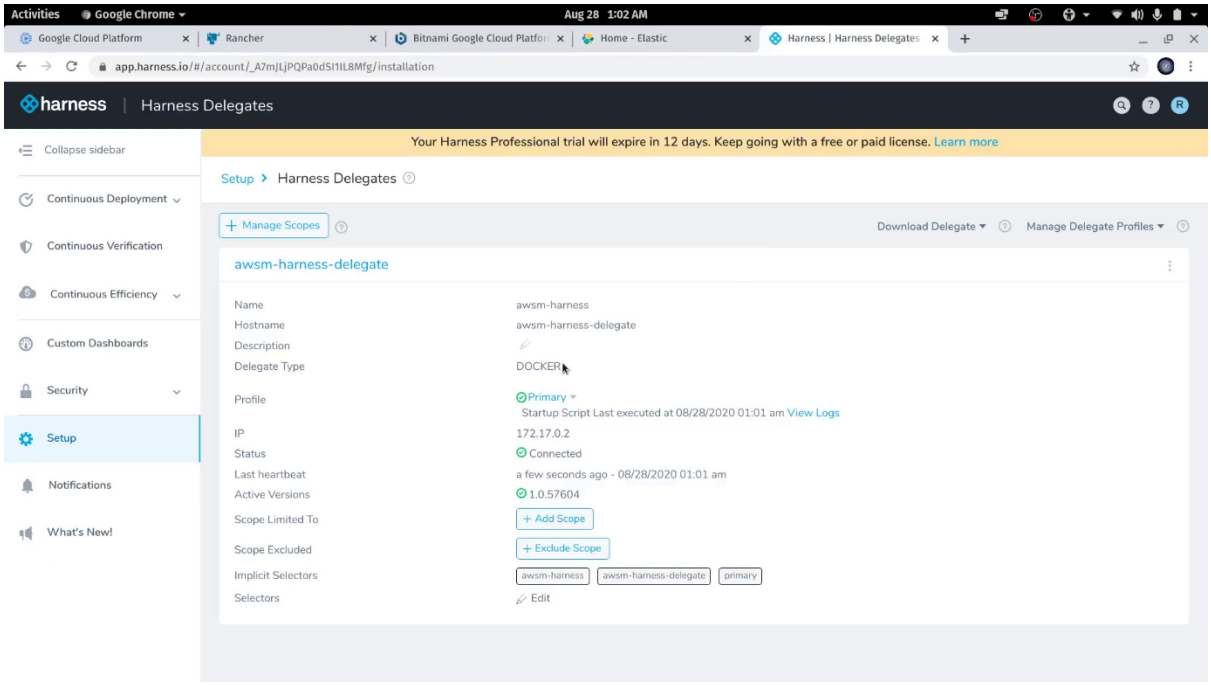


- Then, SSH into the VM instance created for Harness delegate. We already finished installing Docker container engine in this instance.
- Now, copy the contents of the shell script launch-harness-delegate.
- In the VM, run *sudo nano launch-harness-delegate.sh*

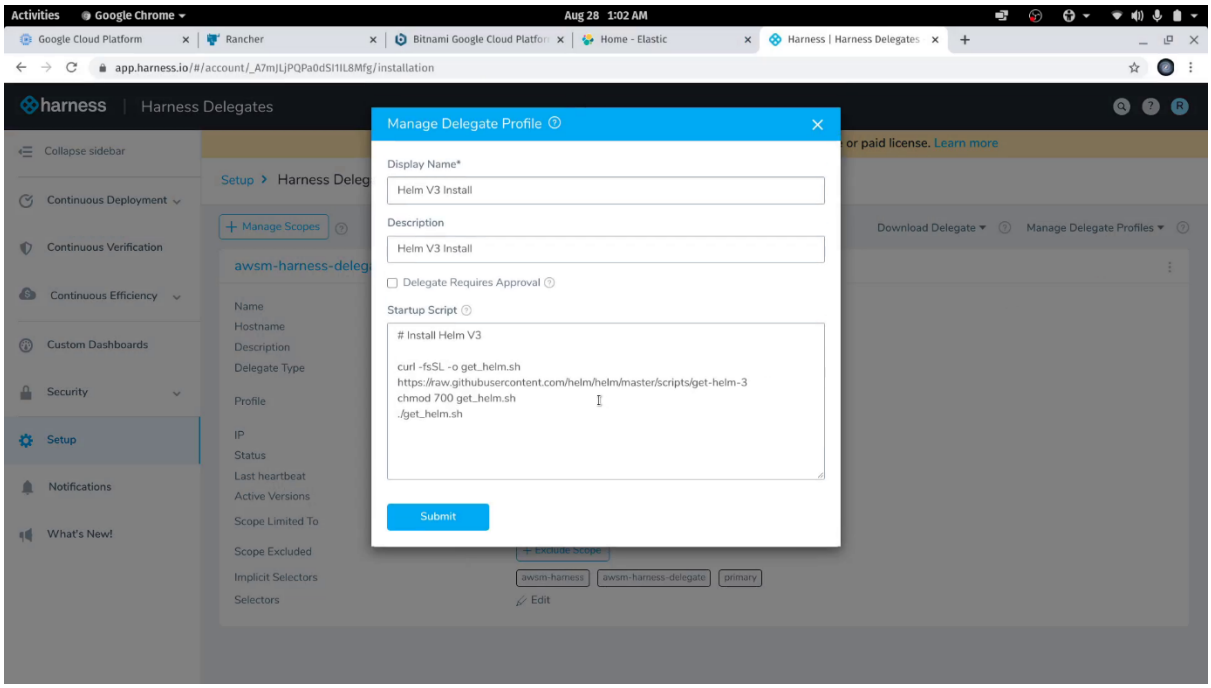
- Paste the copied contents
- Ctrl+O
- Ctrl+X
- Now, run *sudo chmod +x launch-harness-delegate.sh*
- Then execute the script by running *./launch-harness-delegate.sh*
- Now, let's verify whether our delegate is running by using *docker ps* command.



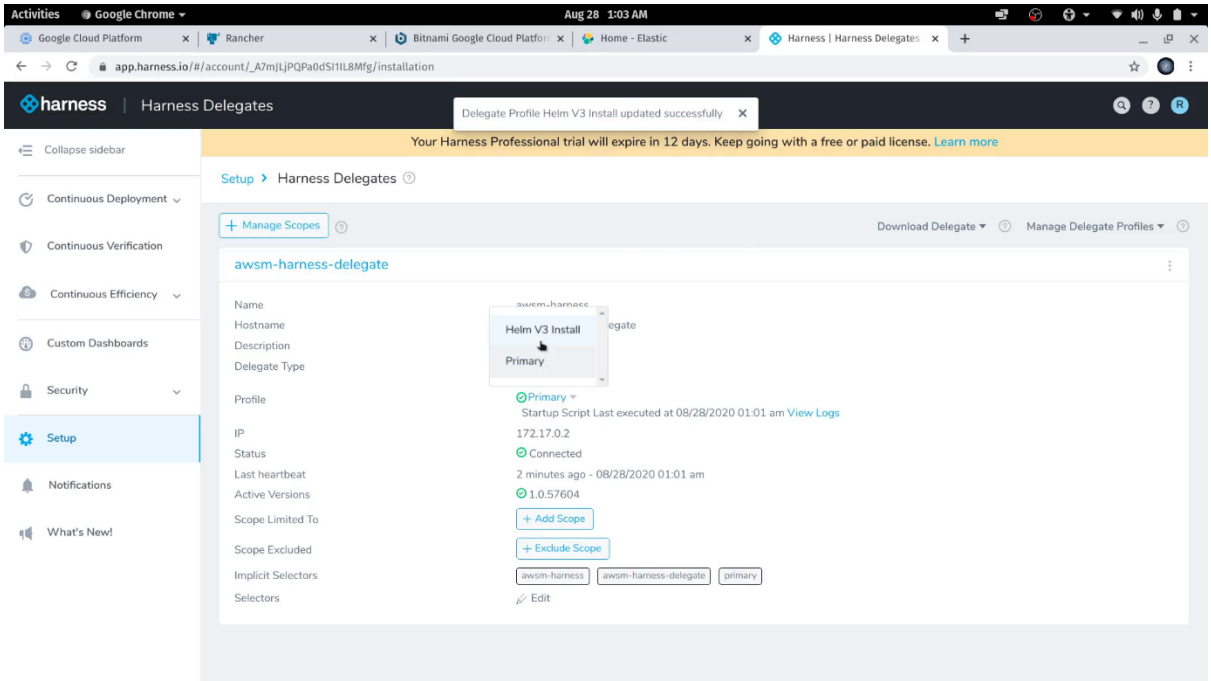
- A container with image name harness/delegate:latest is currently running, so delegate setup is successful.
- Next, let's verify the same from Harness



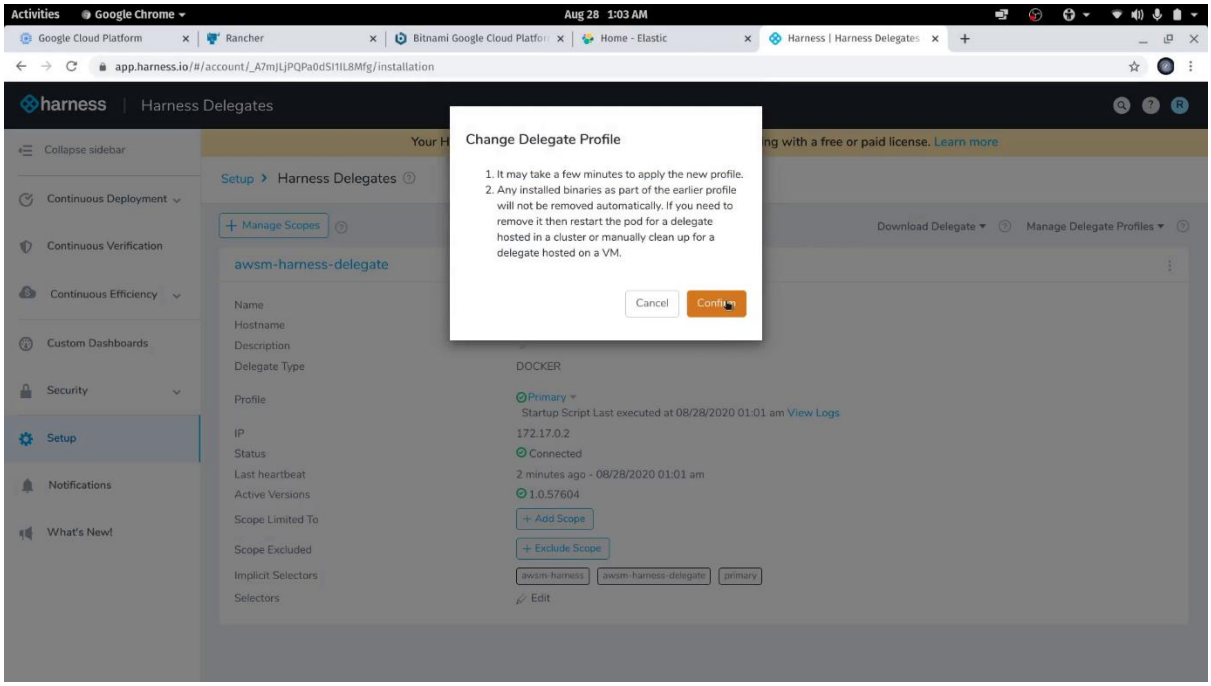
- Now, we are going to change the delegate profile.
- **Manage Delegate Profile > Add New Profile**
- Enter the following shell commands in Startup Script in order install Helm version 3.
- *curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3*
- *chmod 700 get_helm.sh*
- *./get_helm.sh*
- Then, **Submit**.



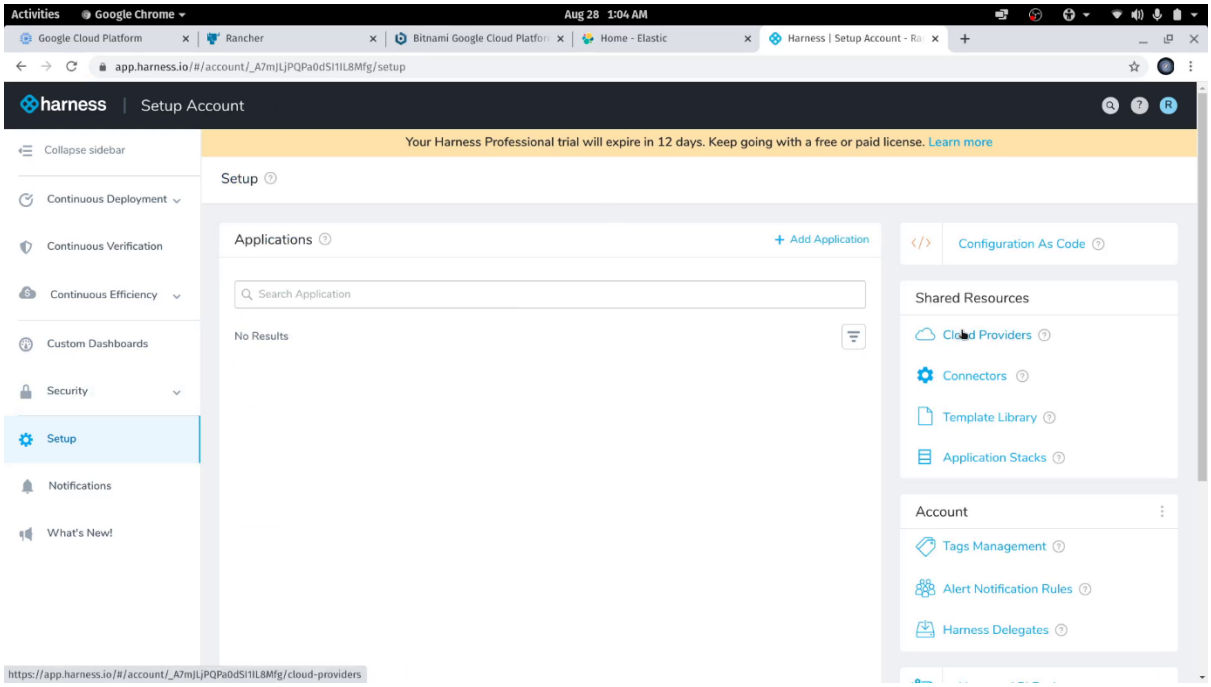
- Now, change delegate profile from Primary to Helm V3 Install



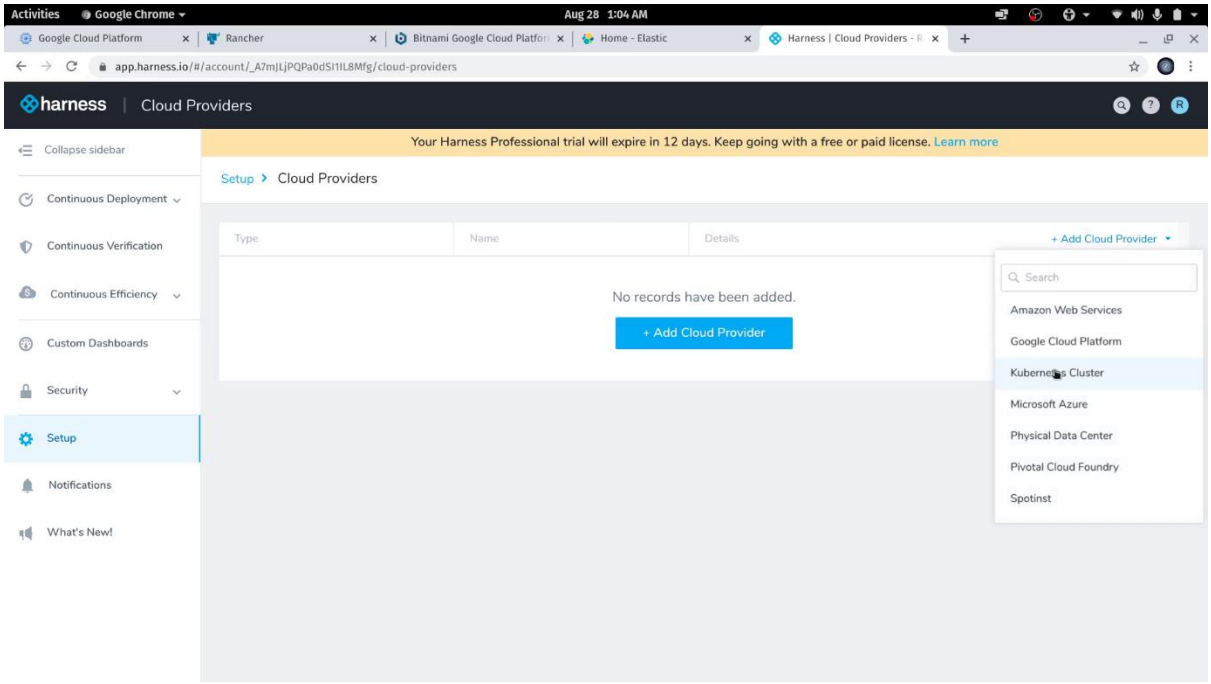
- **Confirm** profile change.



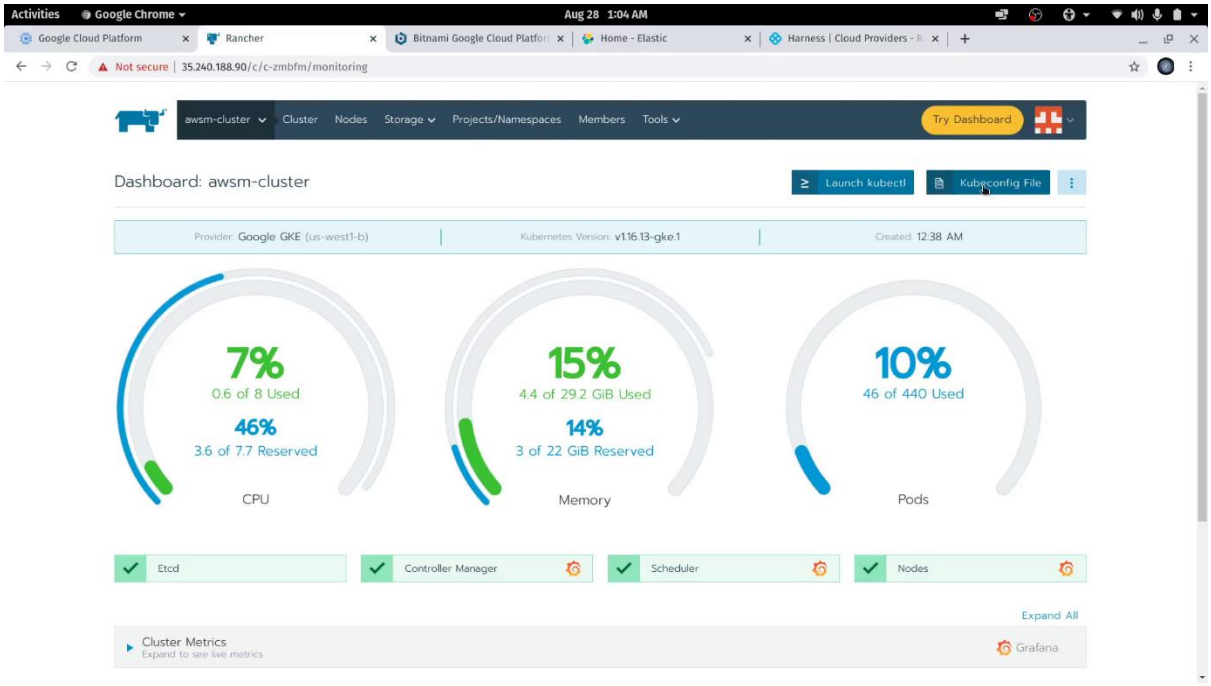
- Now, go to **Setup > Cloud Providers**



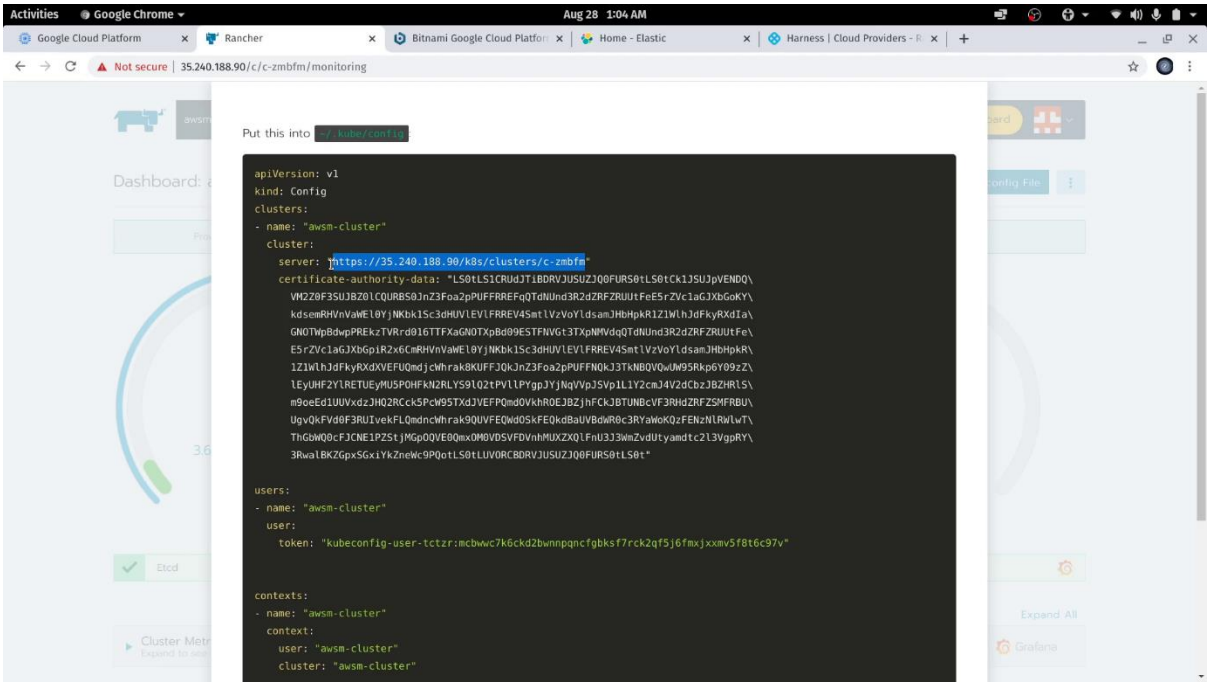
- **Add Cloud Provider > Kubernetes Cluster**



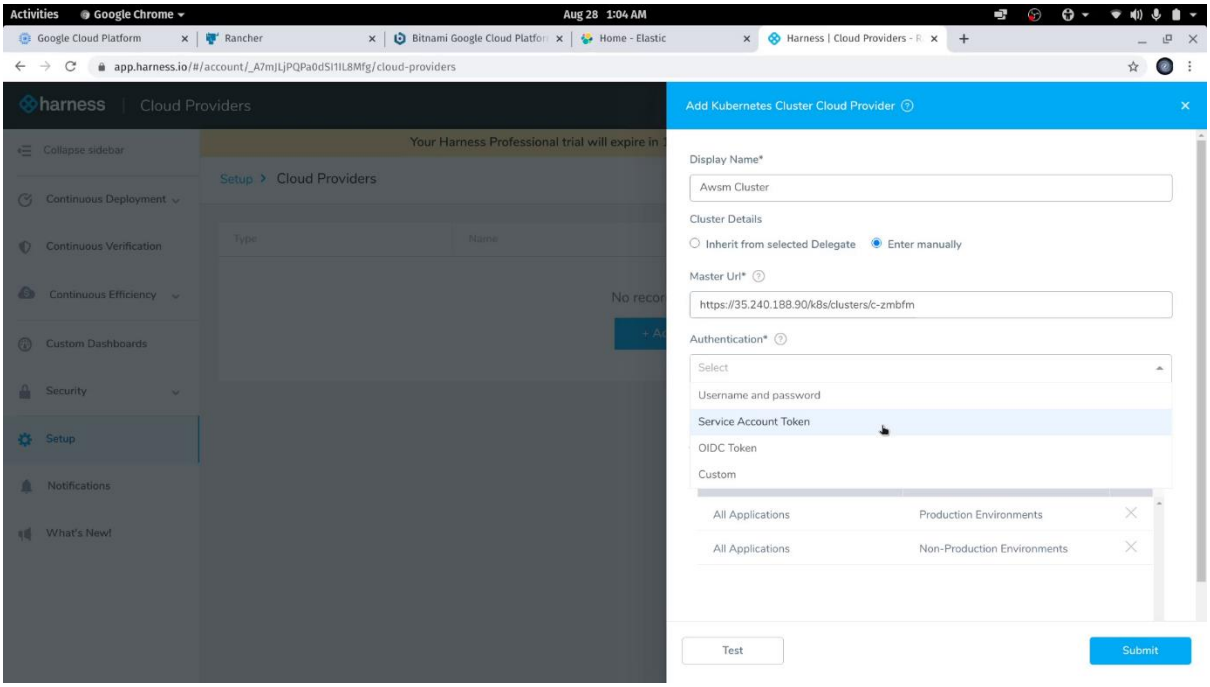
- Go to **Rancher > Cluster > awsm-cluster > Kubeconfig file**



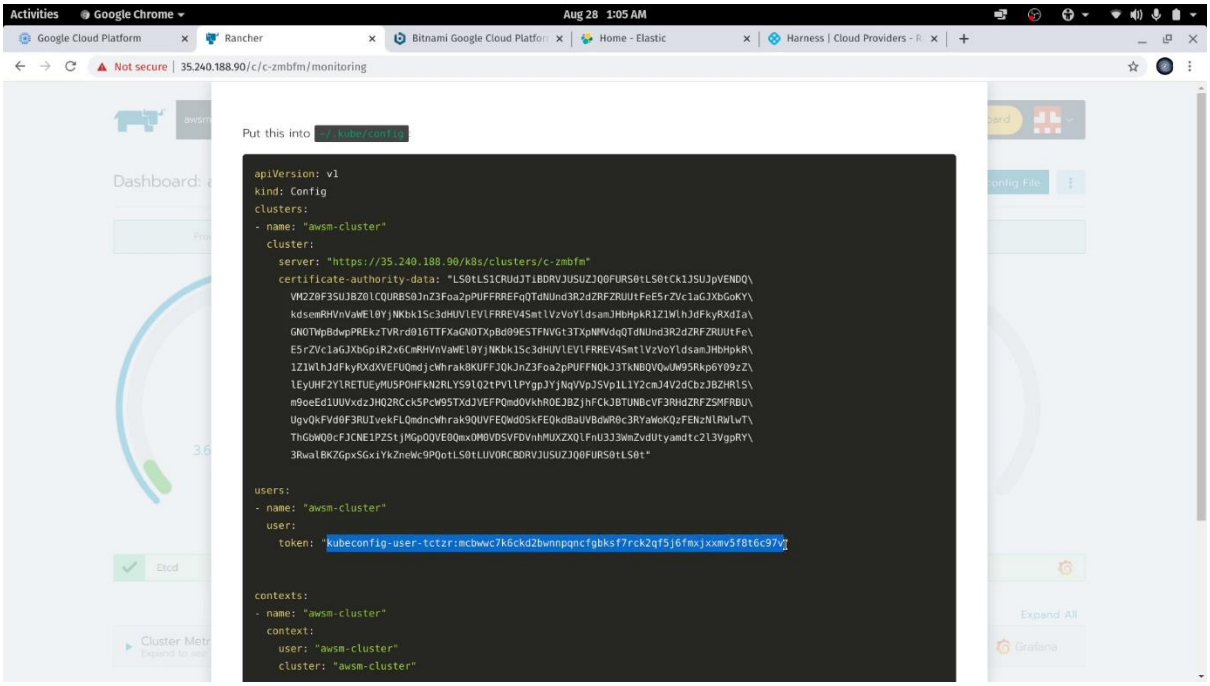
- Copy **Master URL** form **Kubeconfig file**



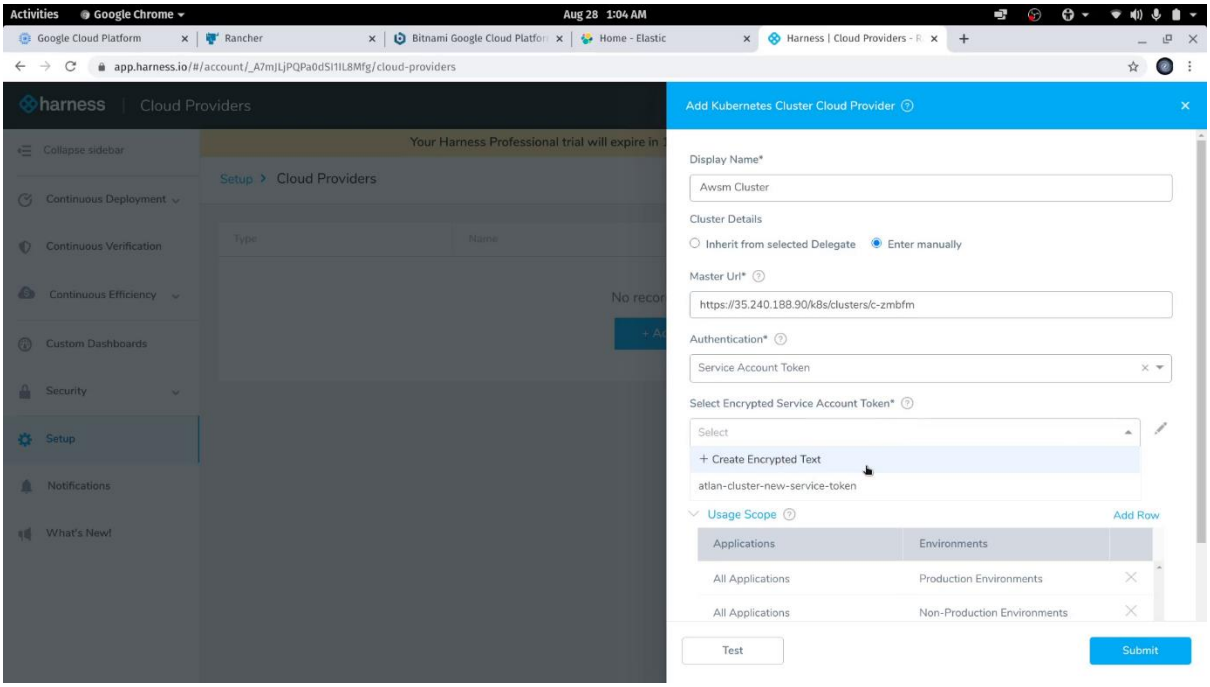
- Paste the **Master URL** and for **Authentication** select **Service Account Token**.



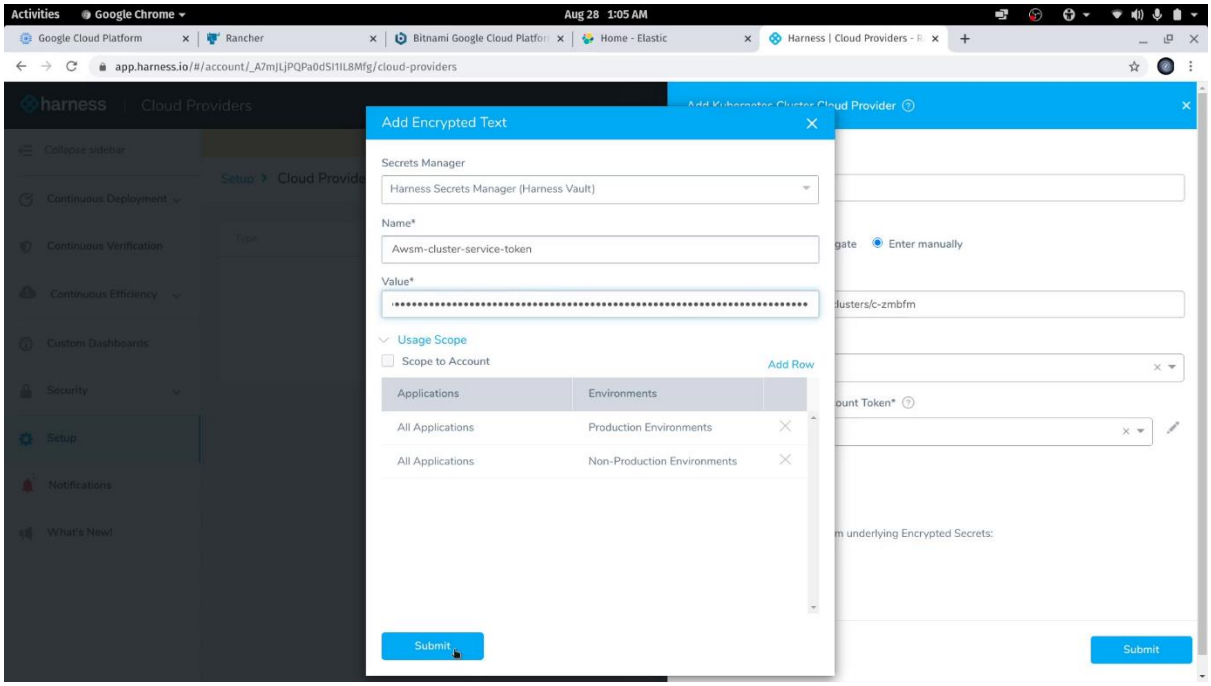
- Copy the **Service Account Token** from **Kubeconfig file**



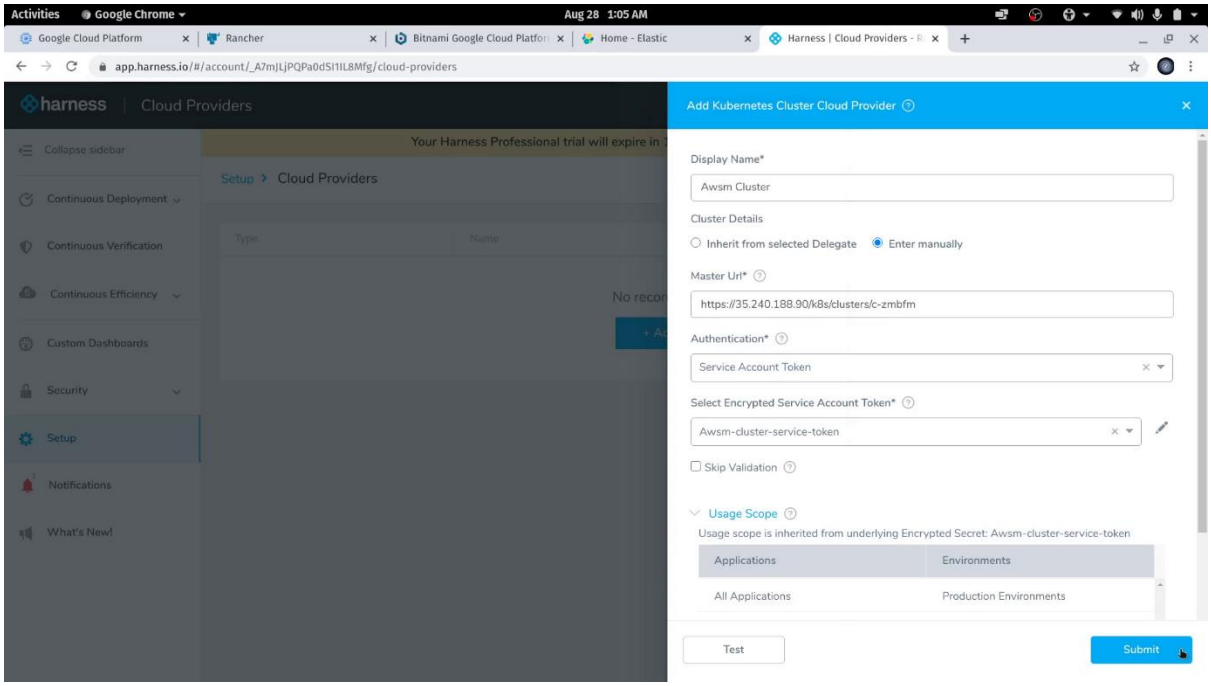
- Now, let's see how to **Create Encrypted Text** in Harness and use it



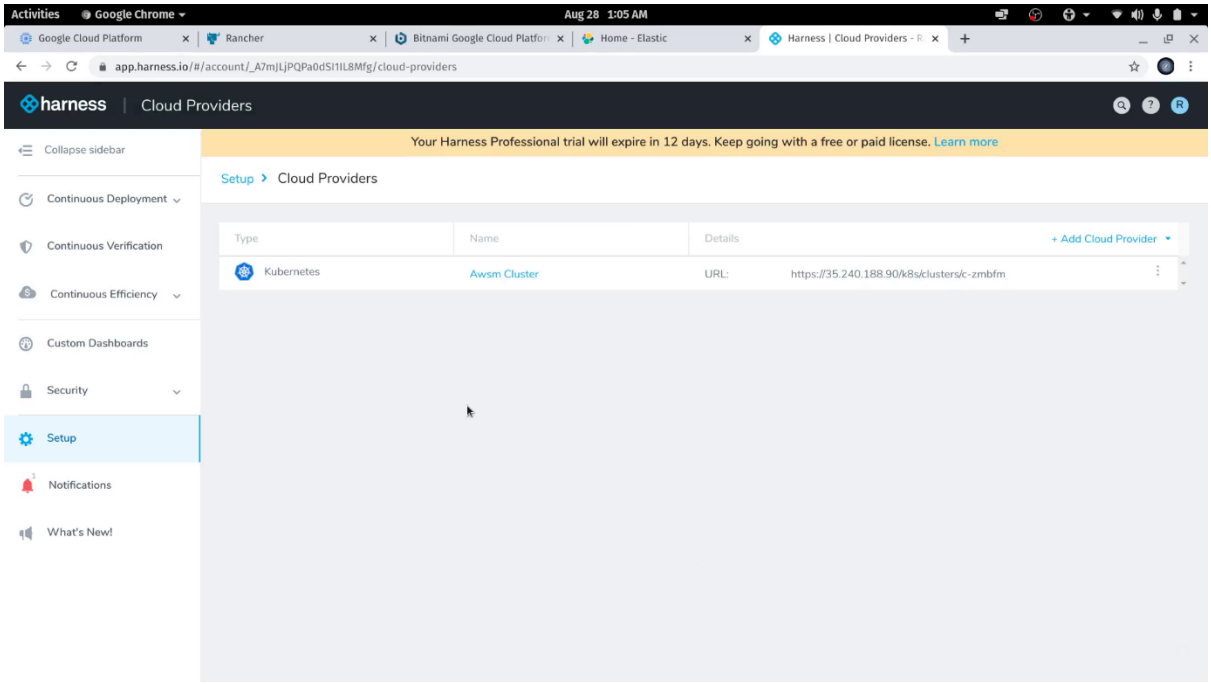
- Paste the copied **Service Account Token** value in the **Value** field and **Submit**.



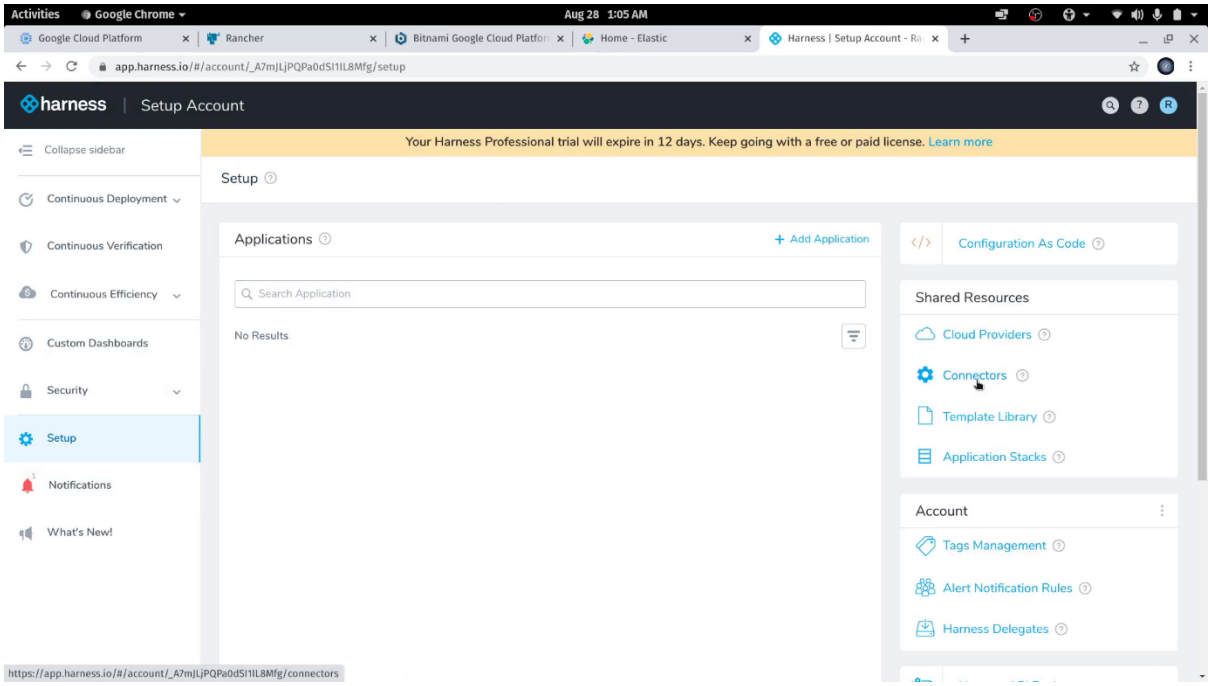
- After creation of the encrypted text, select it in the **Encrypted Service Account Token** field and **Submit**.



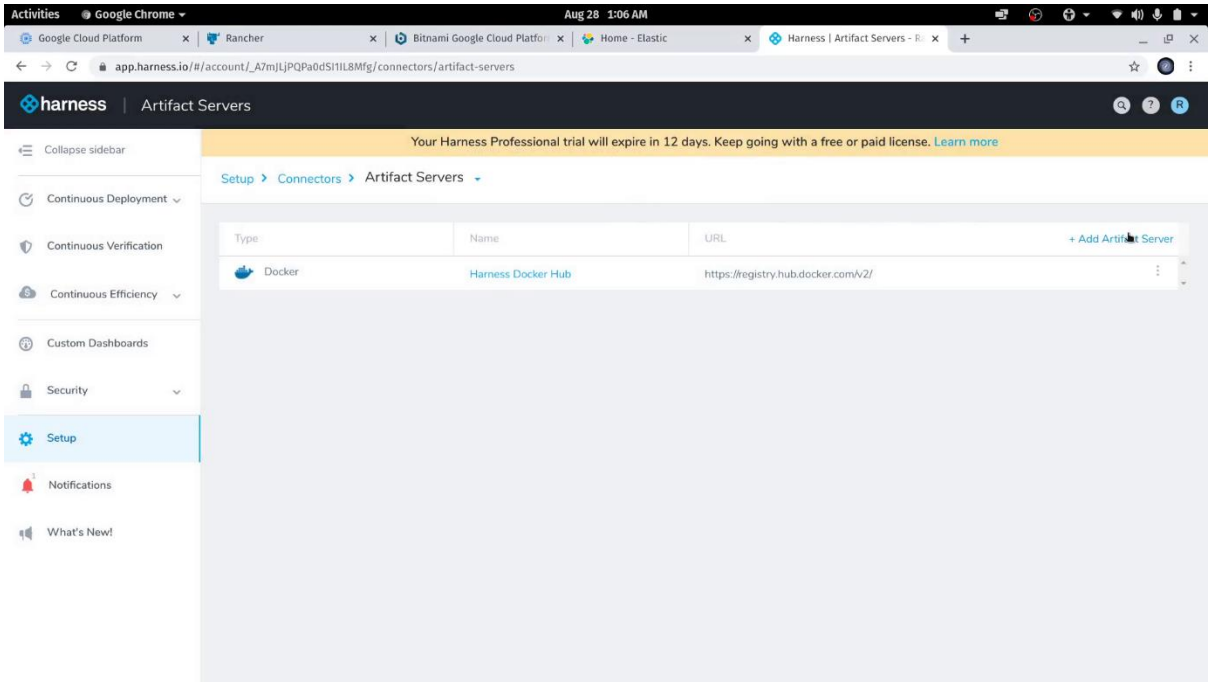
- Now, our Kubernetes cluster is successfully added.



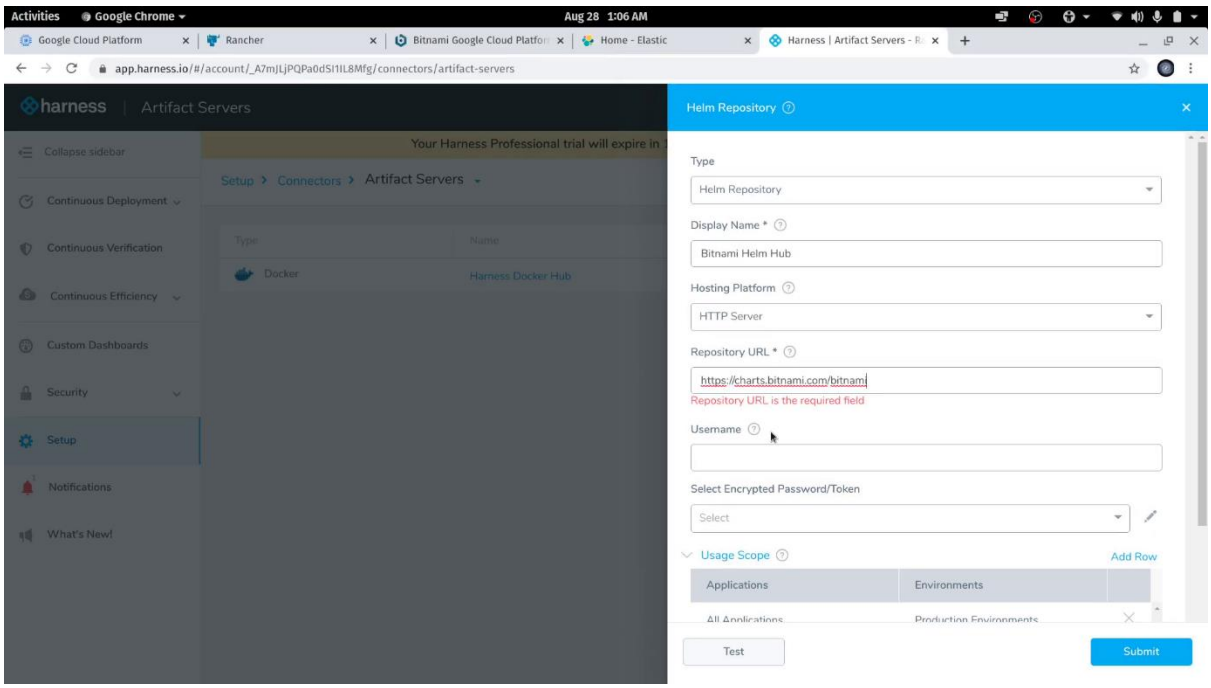
- Next, let's go to **Settings > Connectors > Artifact Servers** add our artifact server from where we are going to retrieve the Helm Charts for deploying the parse server.



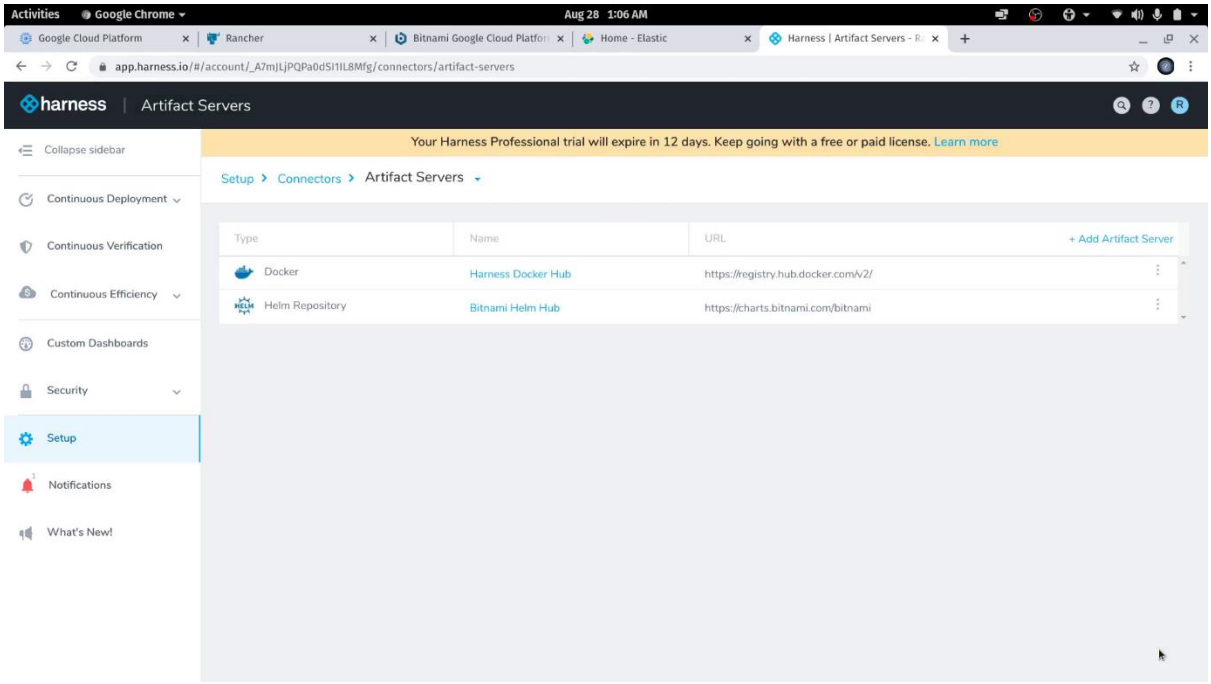
- Now let’s add our artifact server by click **Add Artifact Server**.



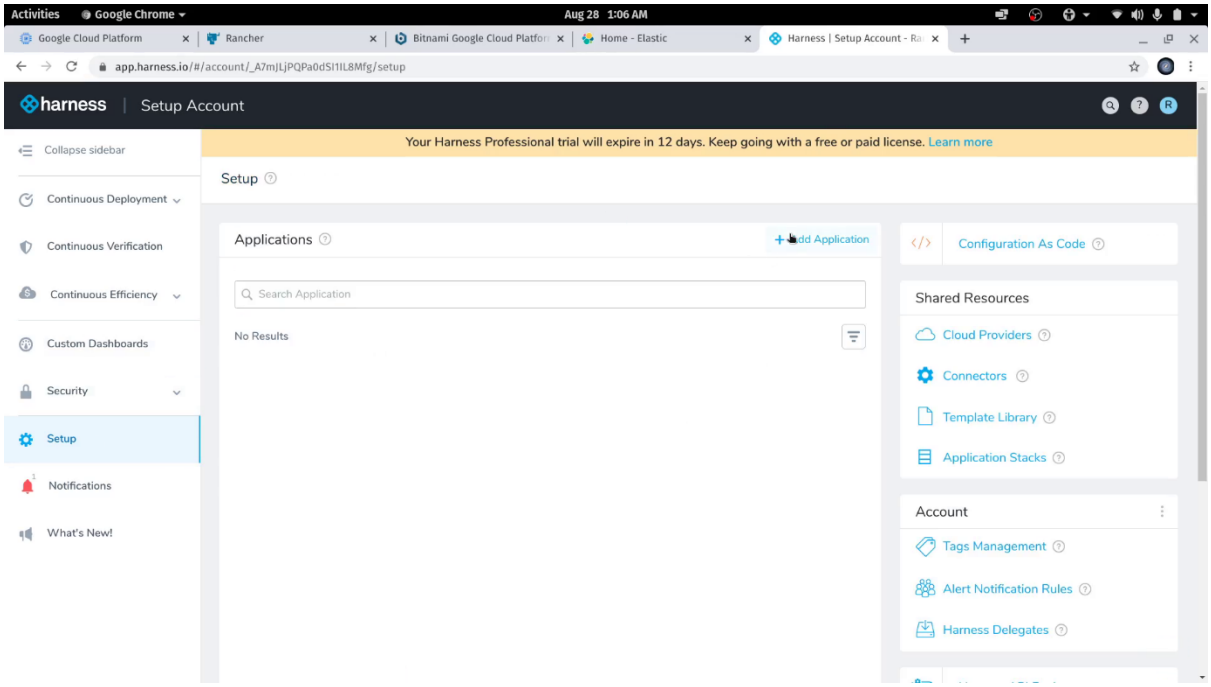
- Let’s add our Helm repository details
- The URL of Bitnami Helm repository is https://charts.bitnami.com/bitnami
- No Username/Password/Token is necessary
- Click **Submit**



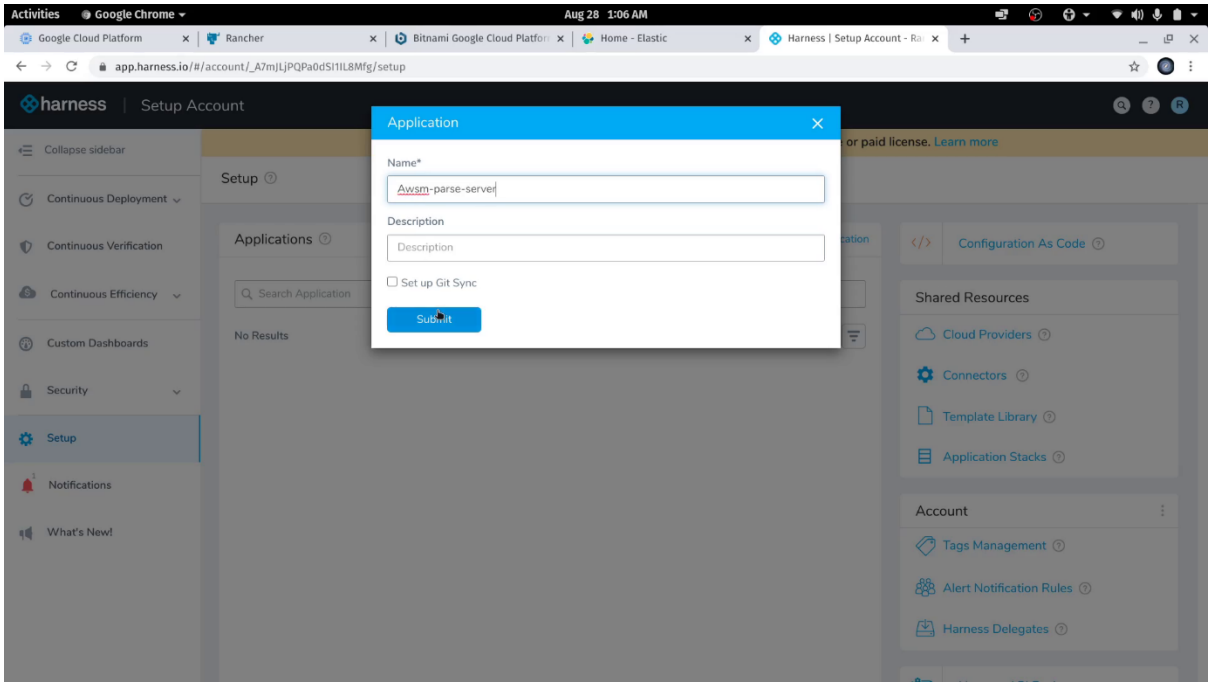
- We have successfully added our Bitnami Helm Repository in our Artifact Server.



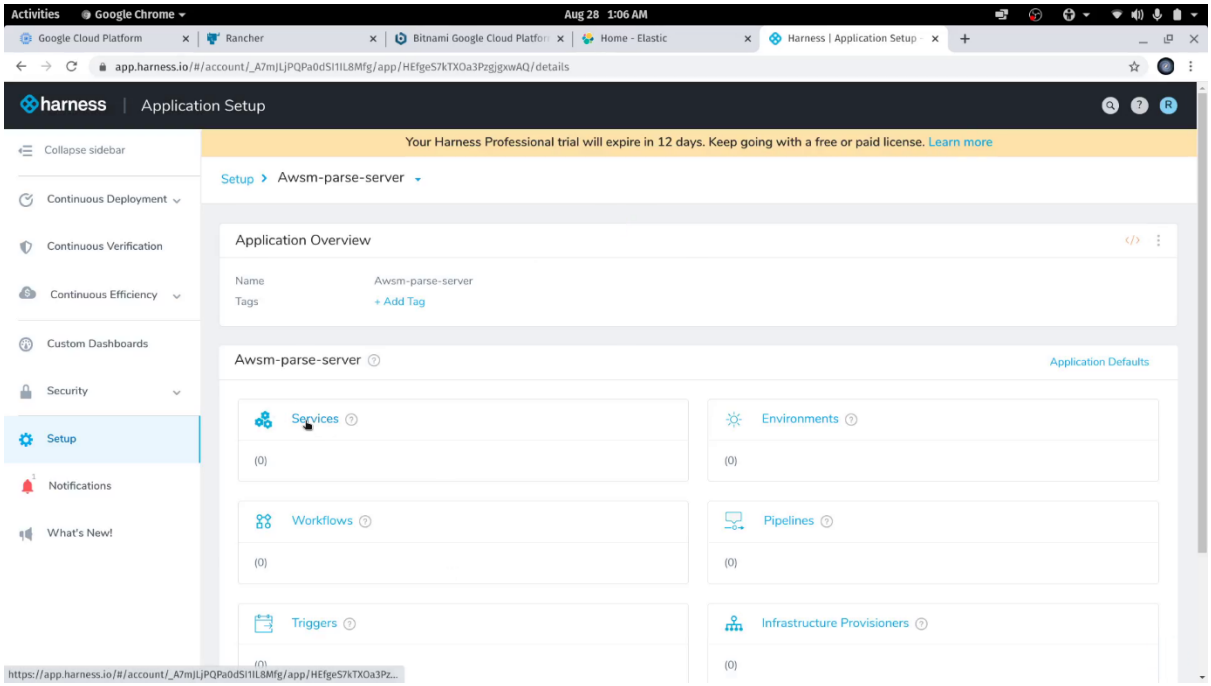
- Now, let's **Add Application** from **Setup**.



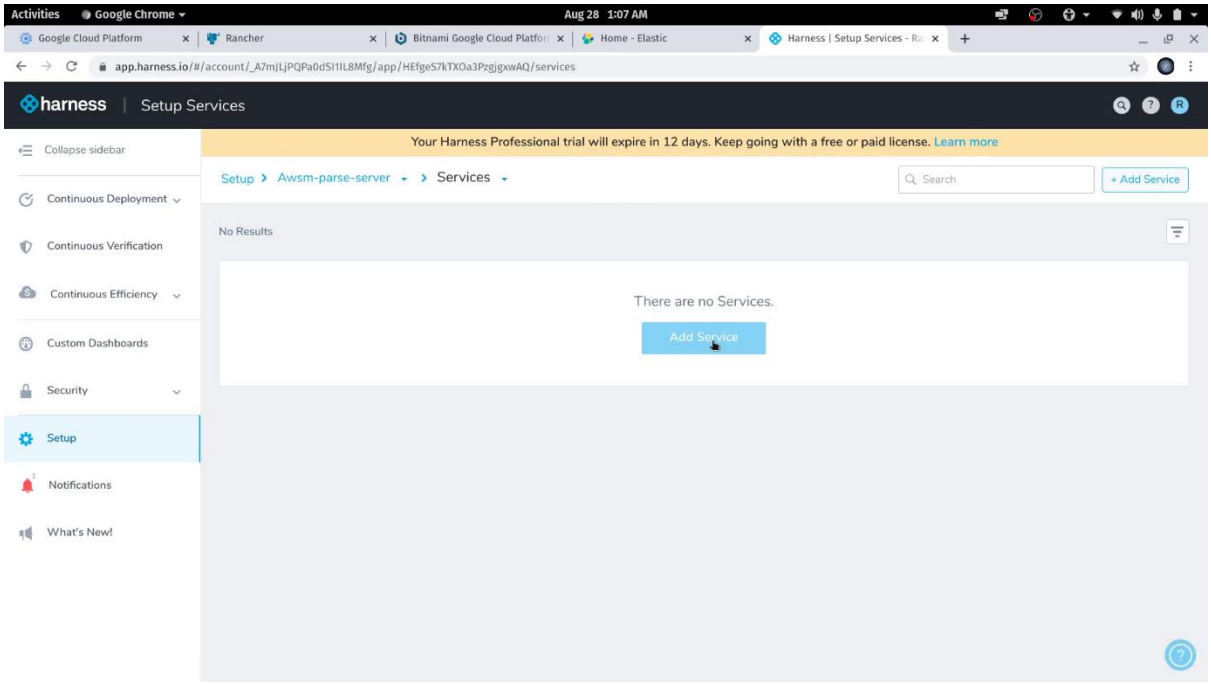
- Enter **Application Name** and **Submit**.



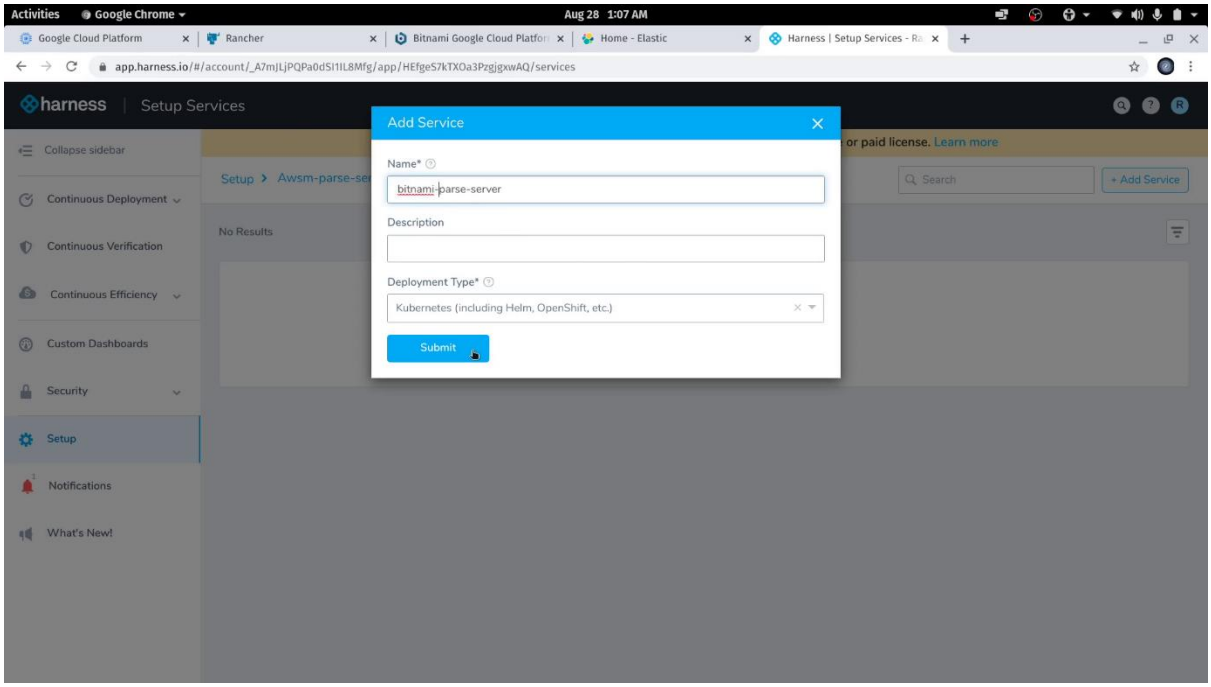
- Next, go to **Awsm-parse-server > Services**



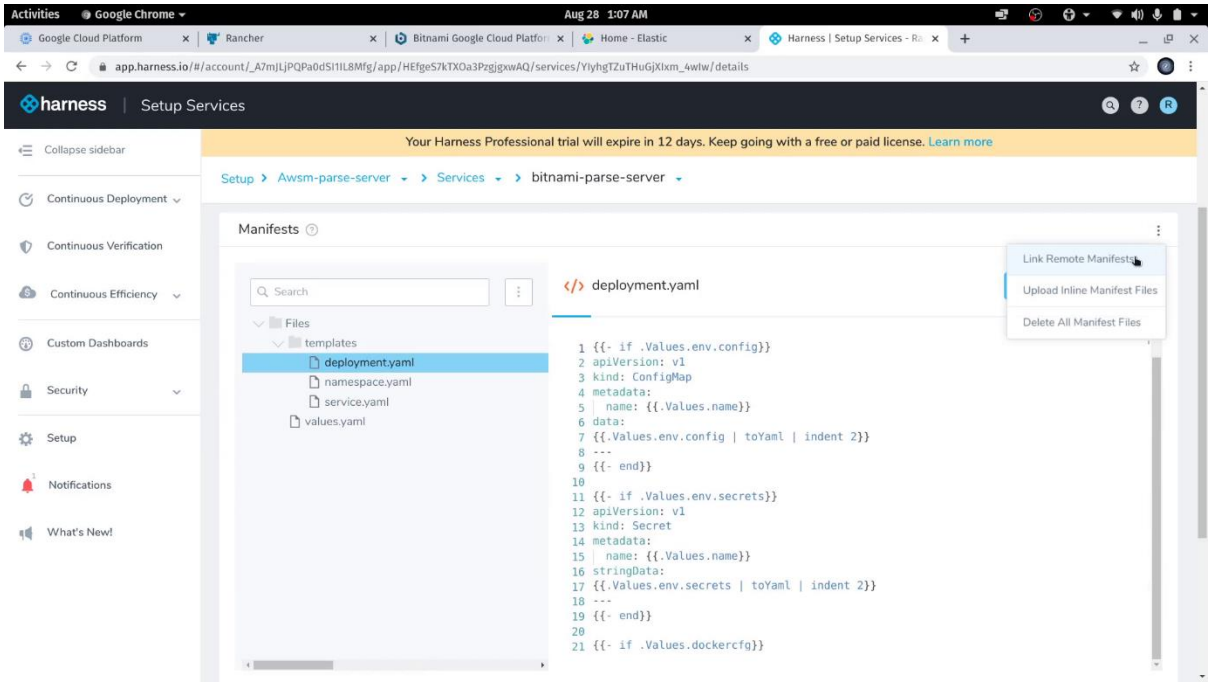
- Now, let's **Add Service**.



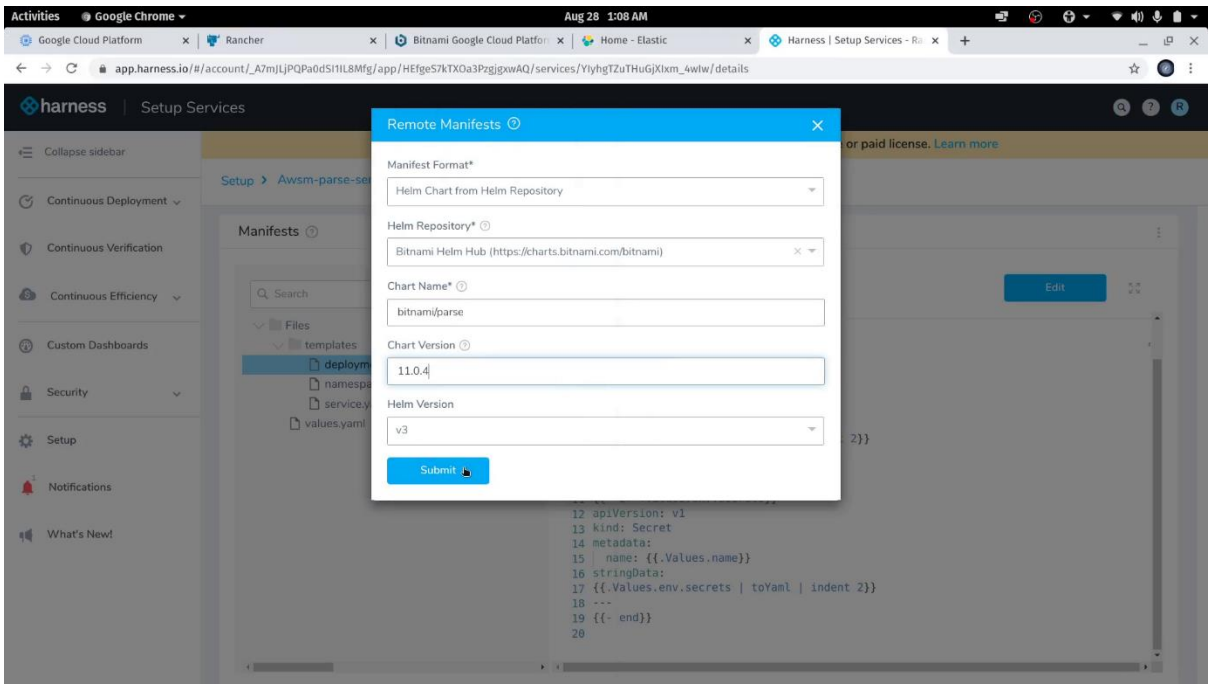
- Add a name for our service.
- Choose **Kubernetes** under **Deployment Type** since we are deploying in a Kubernetes Cluster.
- Then, **Submit**.



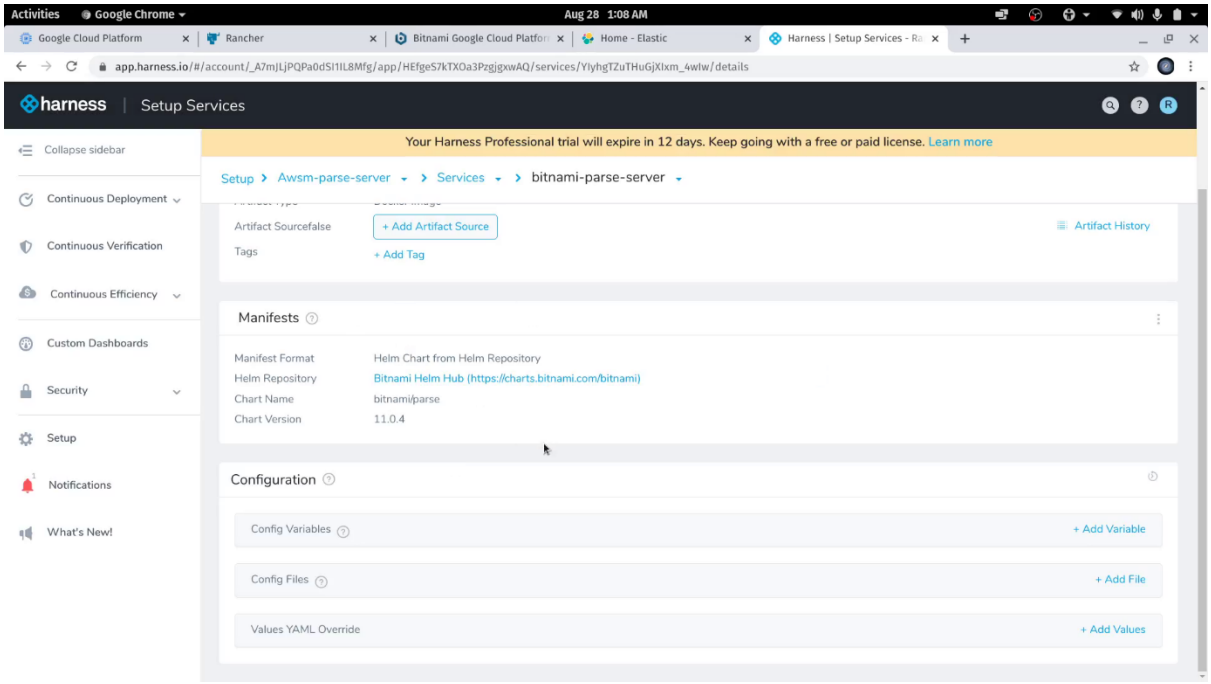
- Let us link our Bitnami Parse Server Manifest



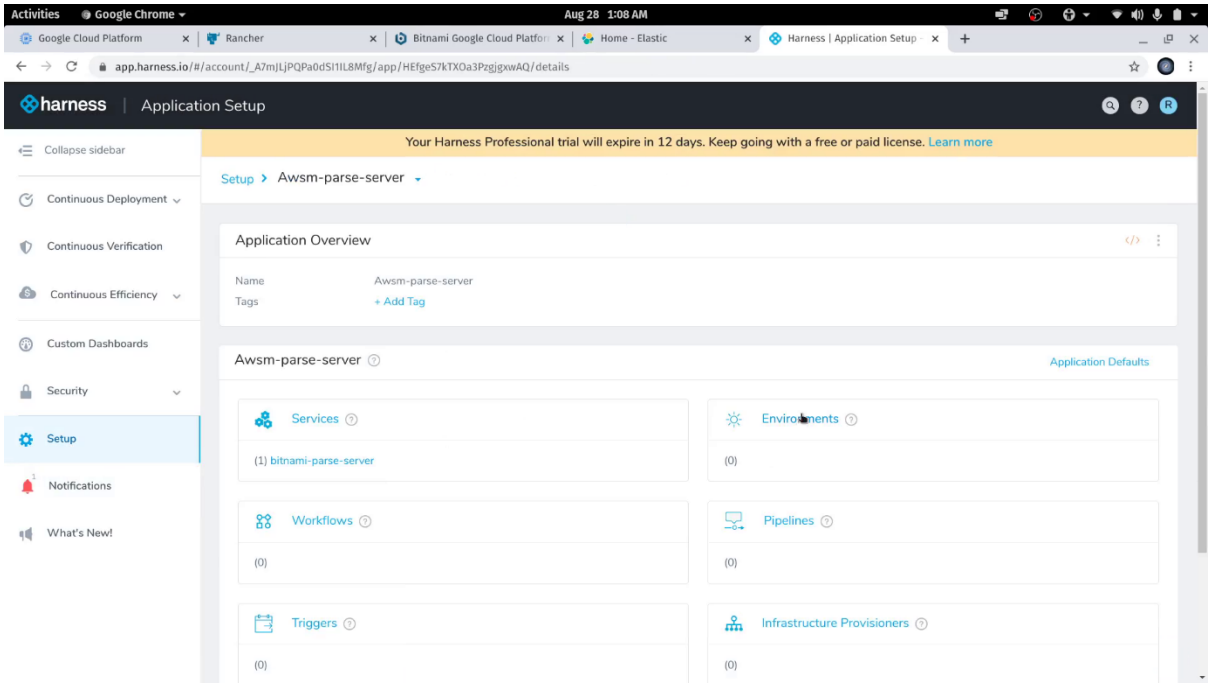
- Add the chart specifications in the Remote Manifests form.
- Chart Name: bitnami/parse
- Chart Version: 11.0.4
- Then, **Submit**.



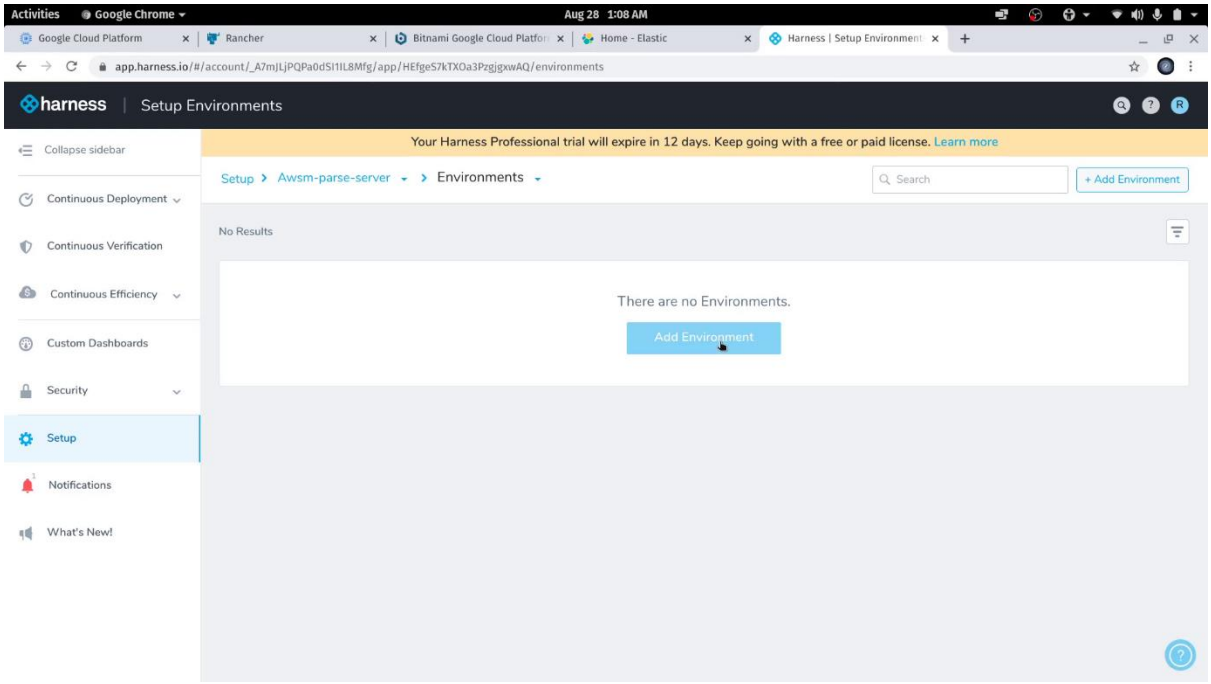
- Our manifest is successfully updated.



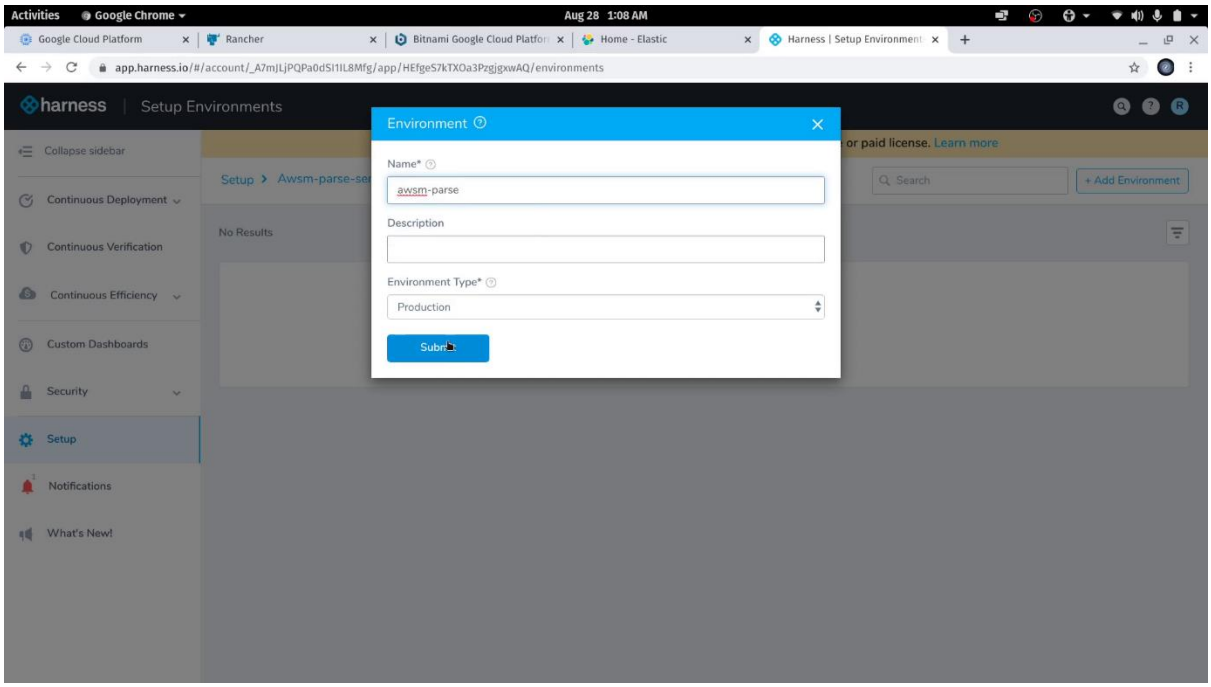
- Now, let's head to **Environments**.



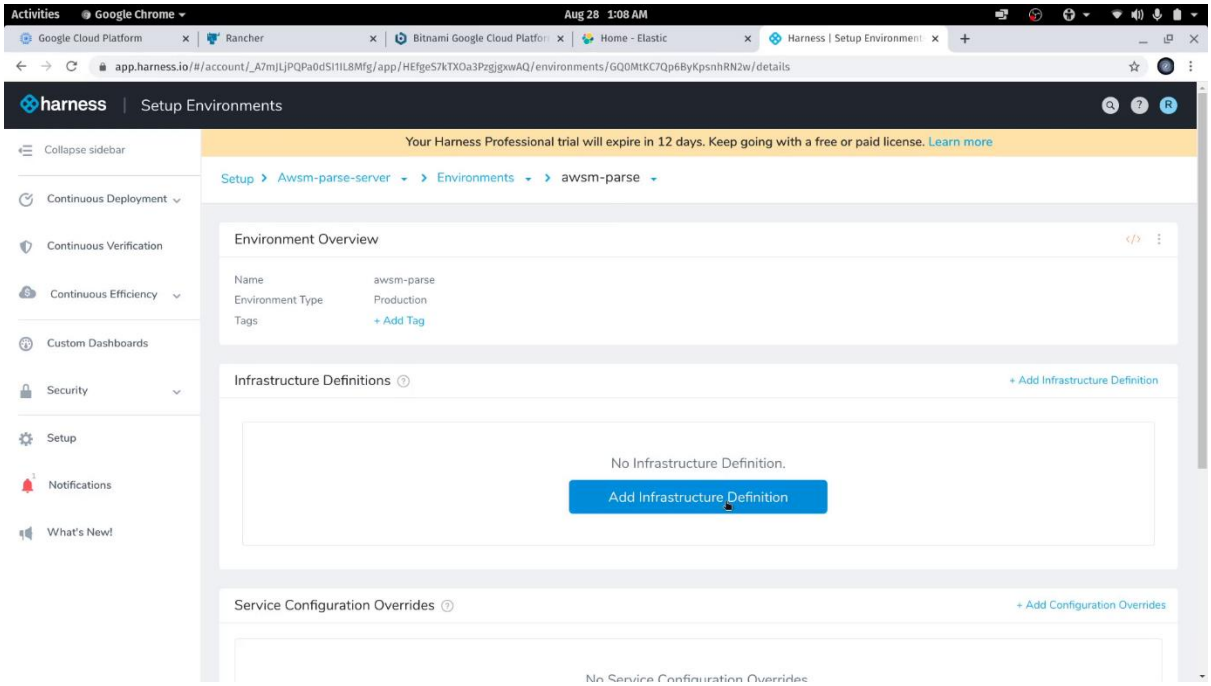
- Next, we will **Add Environment**.



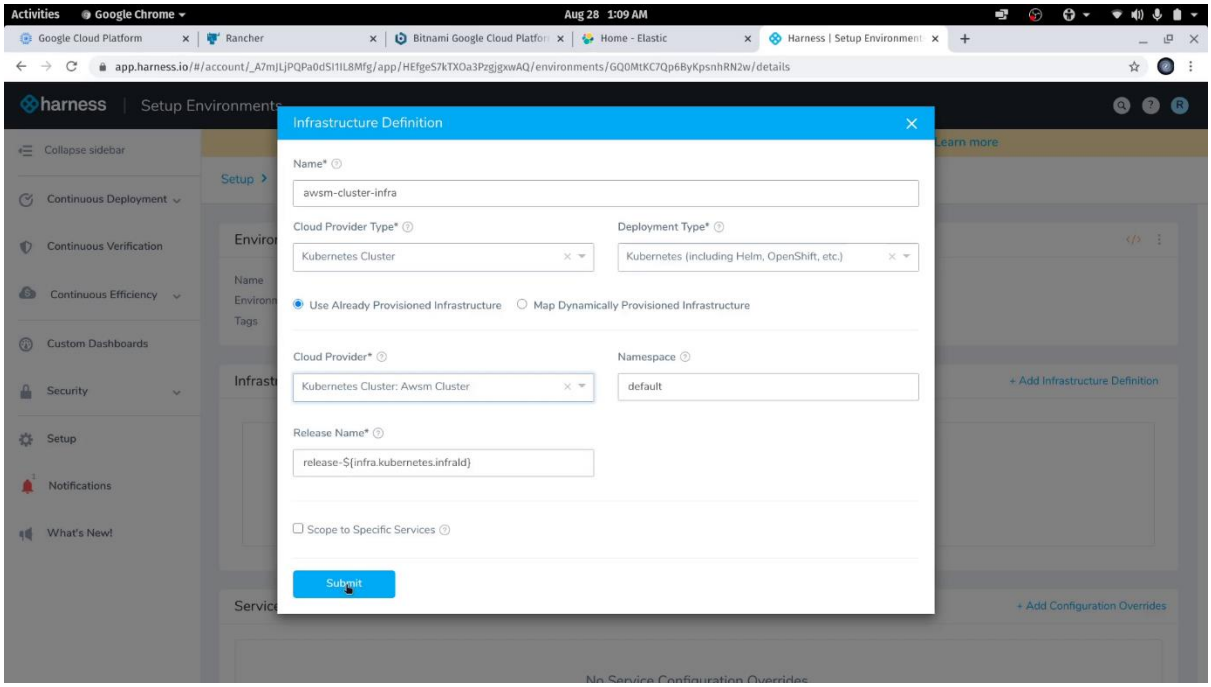
- Select **Environment Type** as **Production** and name the Environment.
- Then, **Submit**.



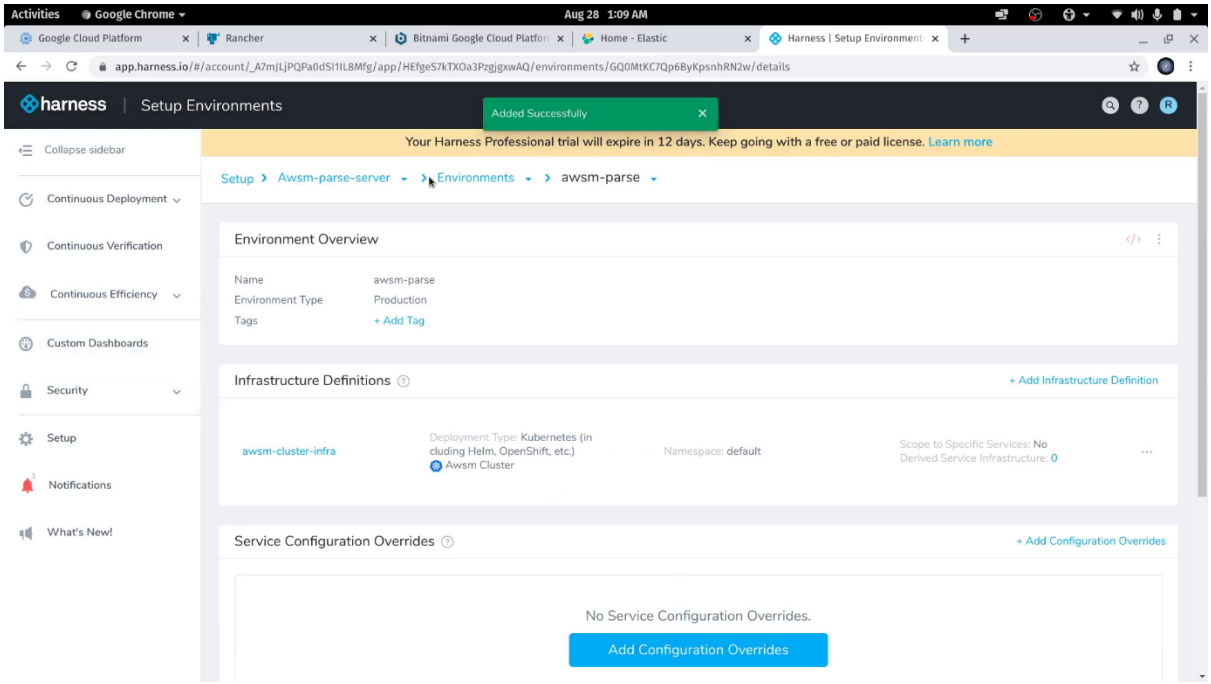
- Now, let's **Add Infrastructure Definition**.



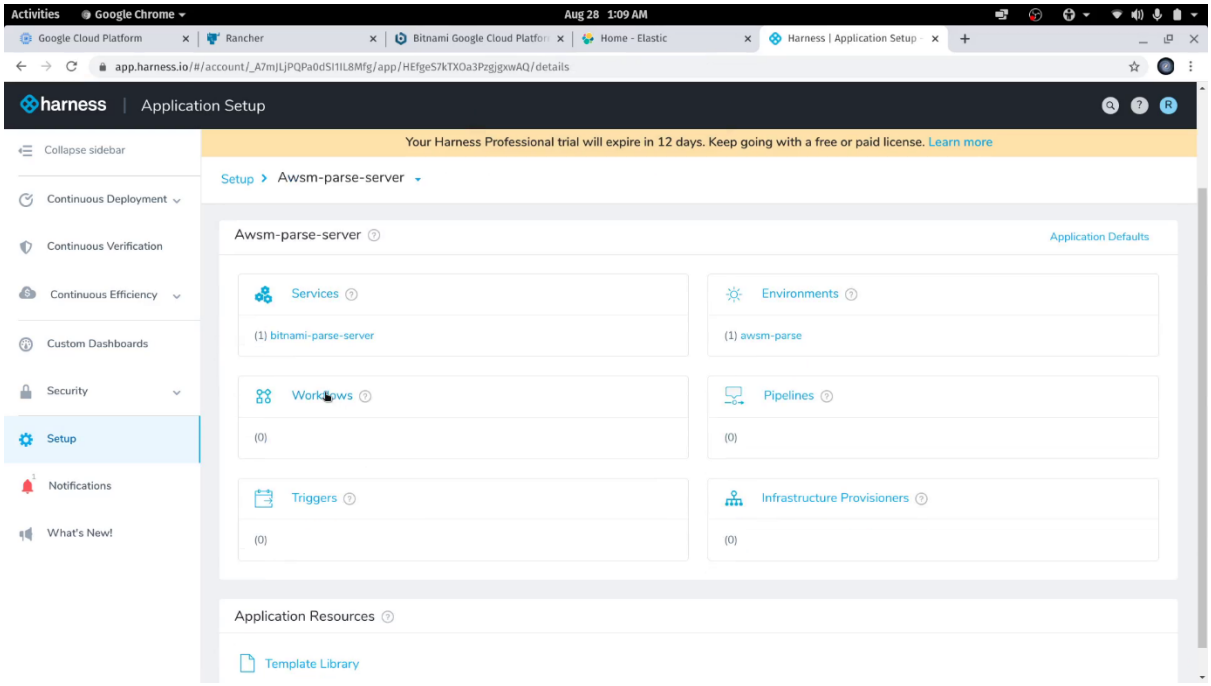
- Fill in the following details in the **Infrastructure Definition** form and **Submit**.



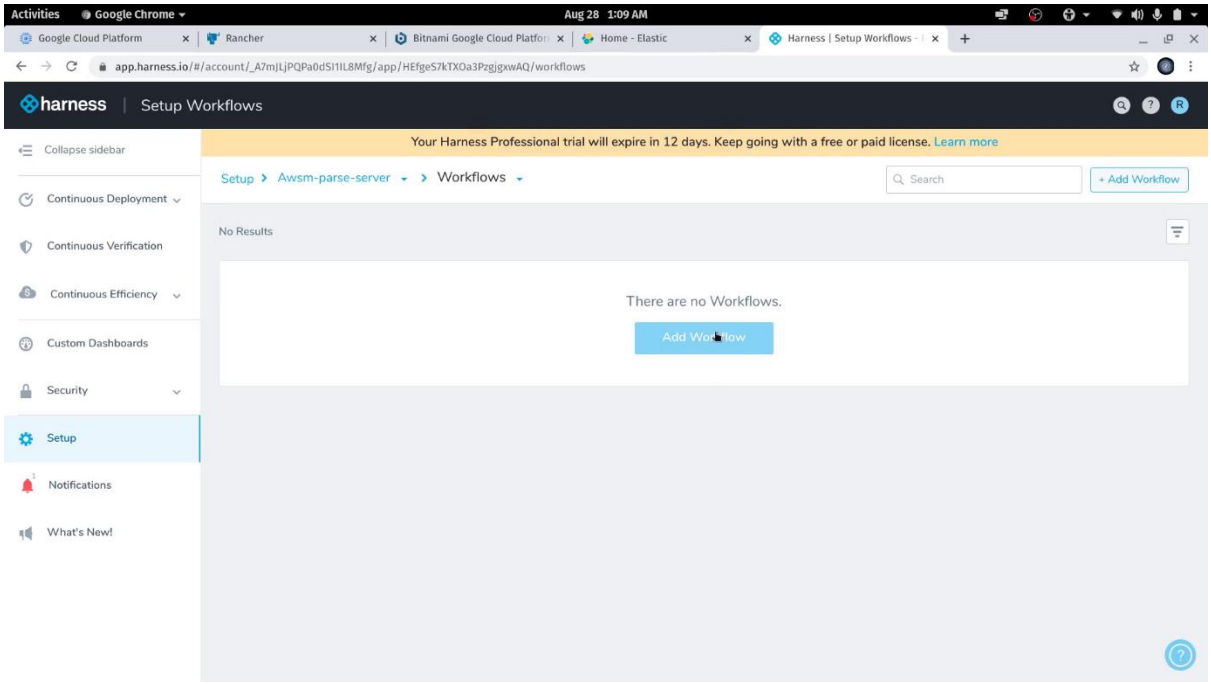
- We have successfully added our **Infrastructure Definition**.



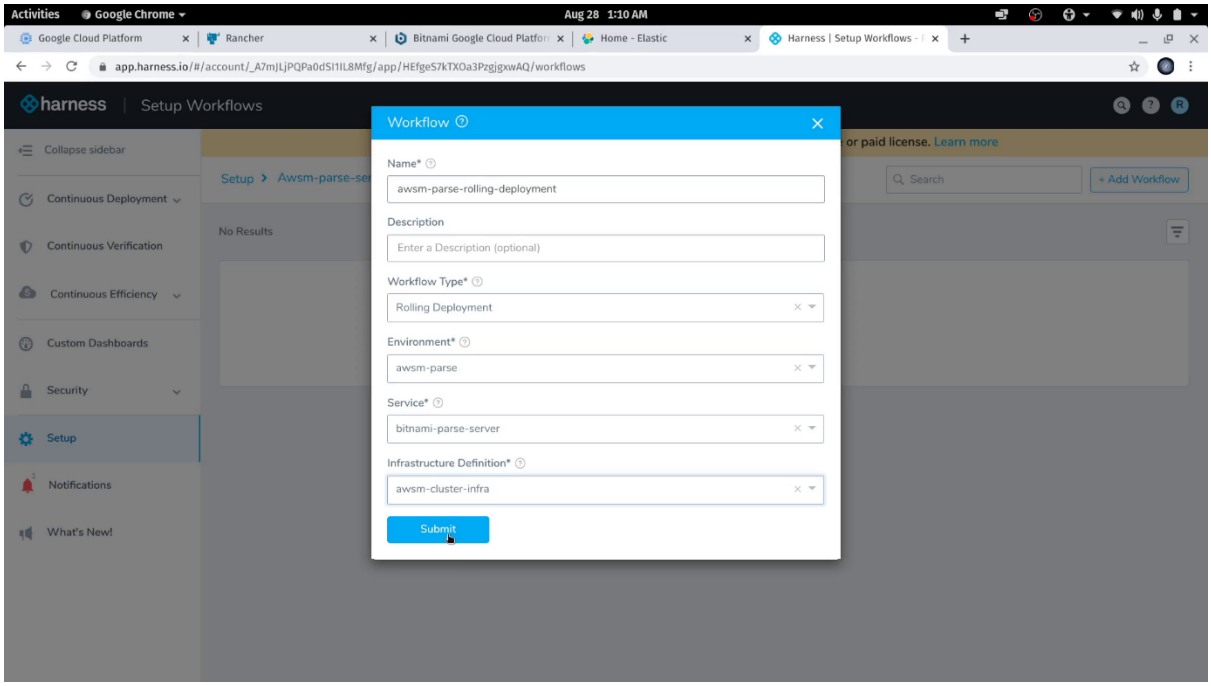
- Next, **Awsm-parse-server > Workflows**



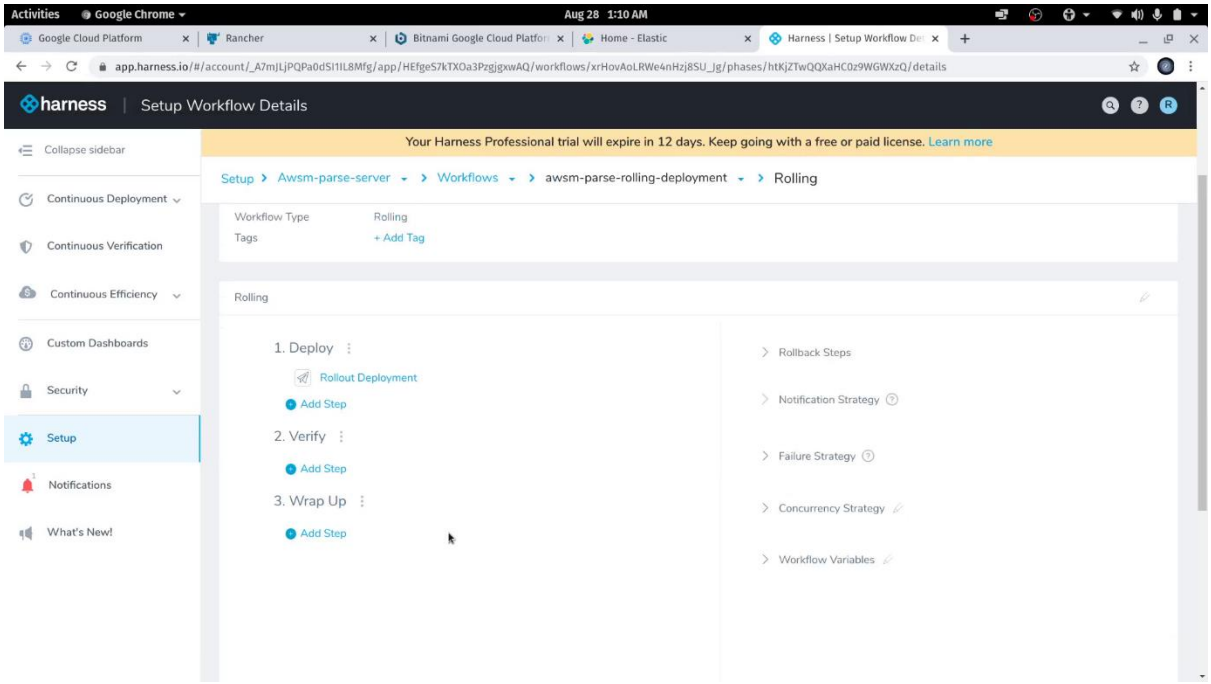
- **Add Workflow**



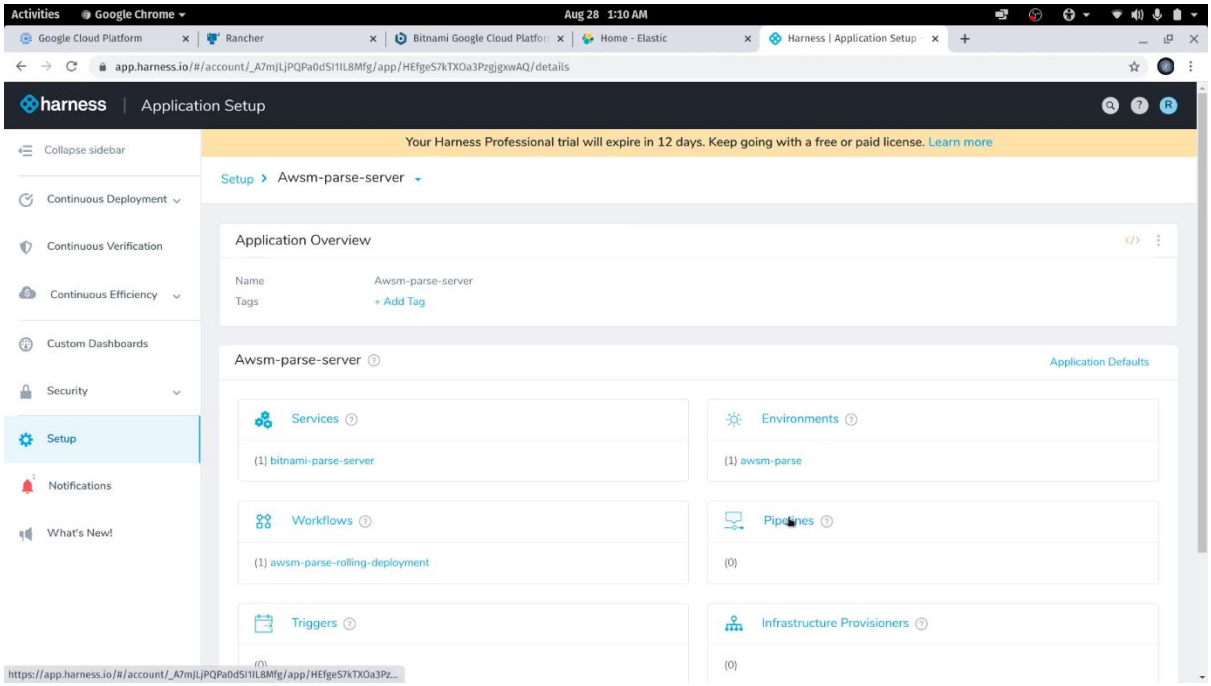
- Select **Workflow Type** as **Rolling Deployment**.
- Fill the rest of the form.
- Then, **Submit**.



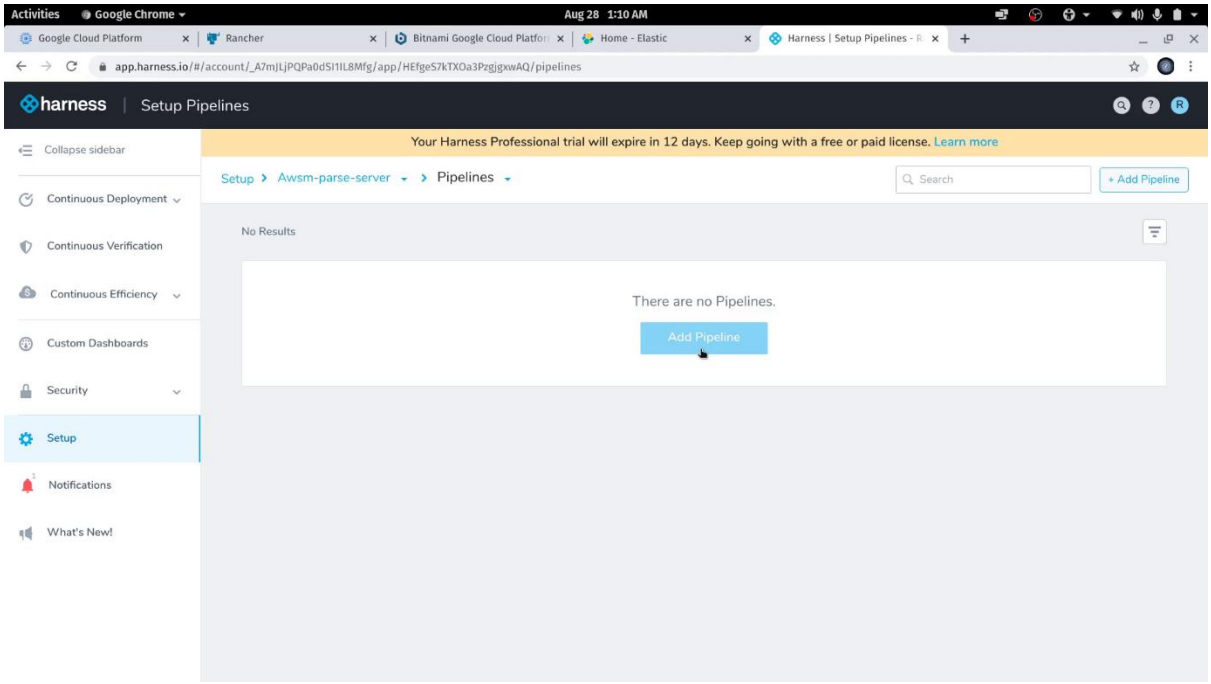
- Workflow successfully added.



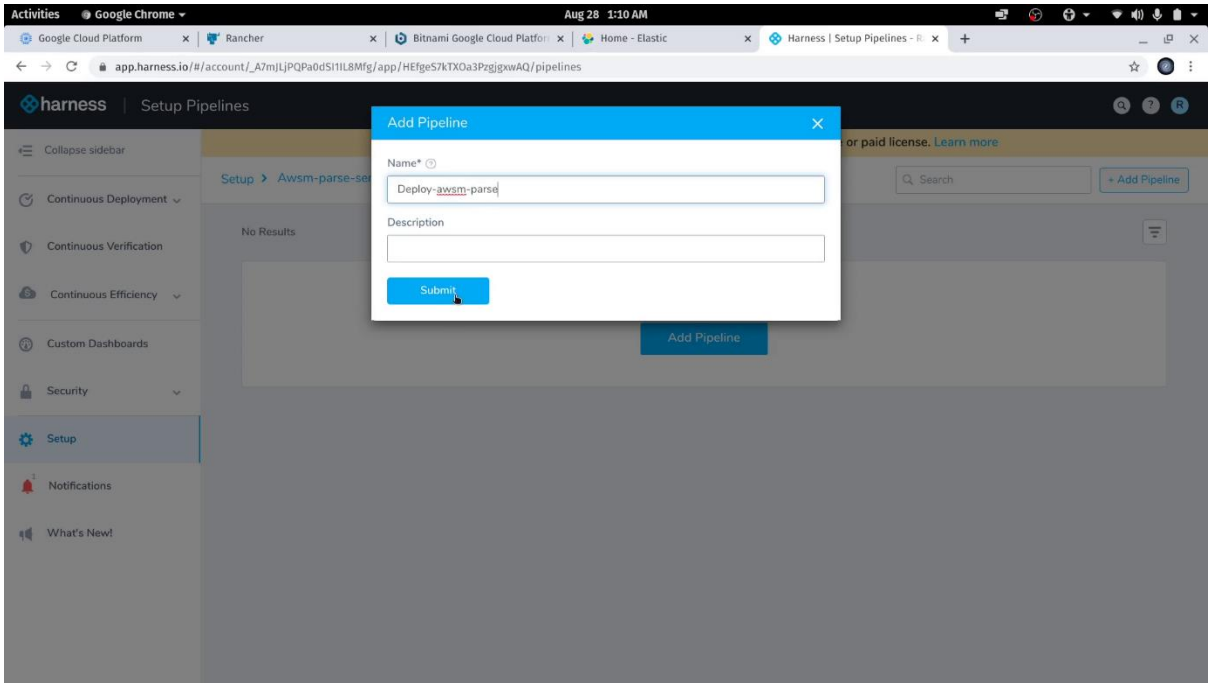
- Finally, let's head to **Awsm-parse-server > Pipelines**



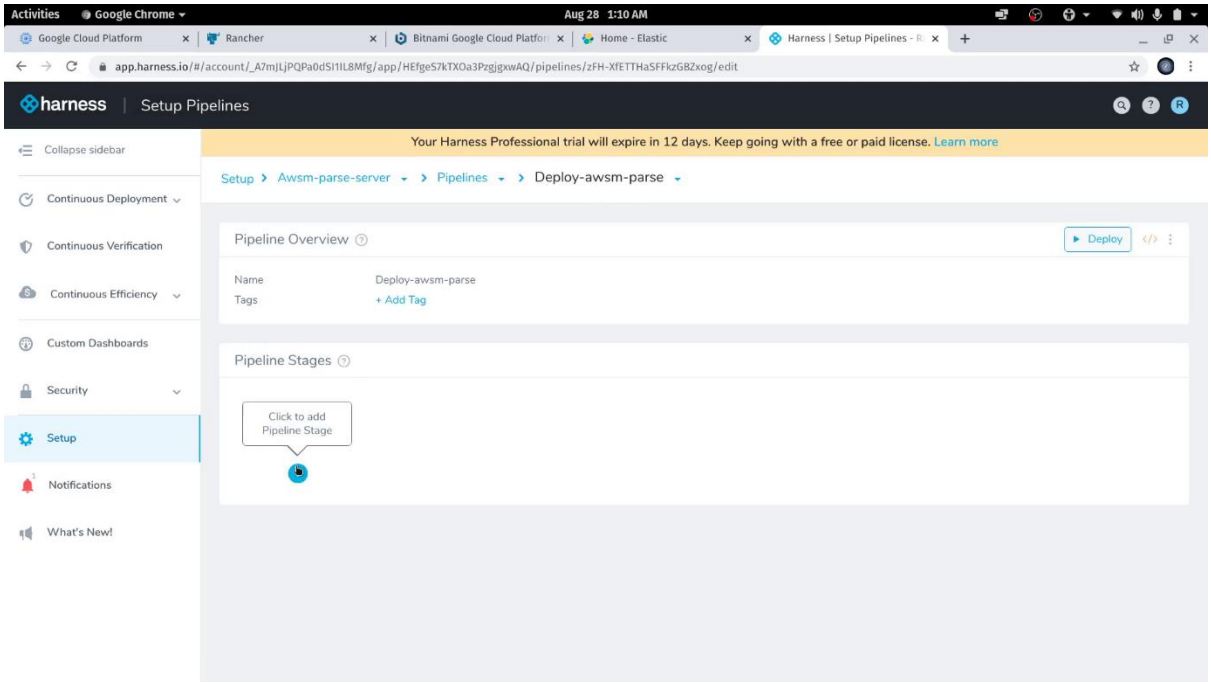
- Now, let's **Add Pipeline**.



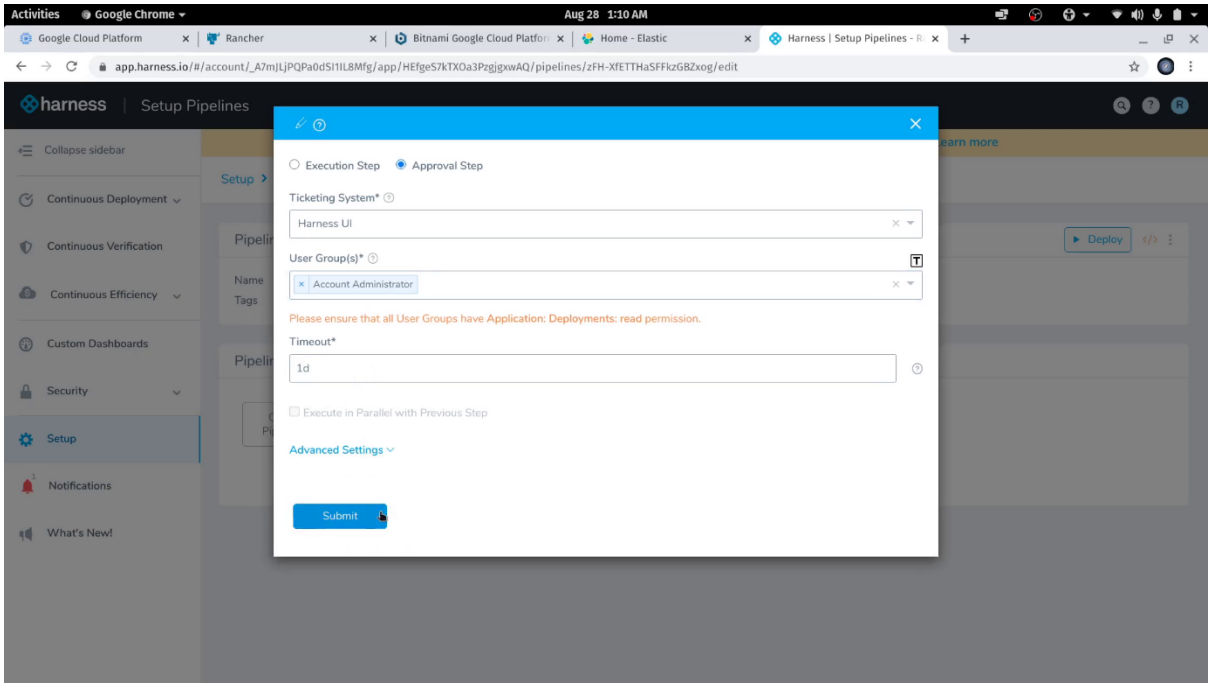
- Enter a name for our **Pipeline** and **Submit**.



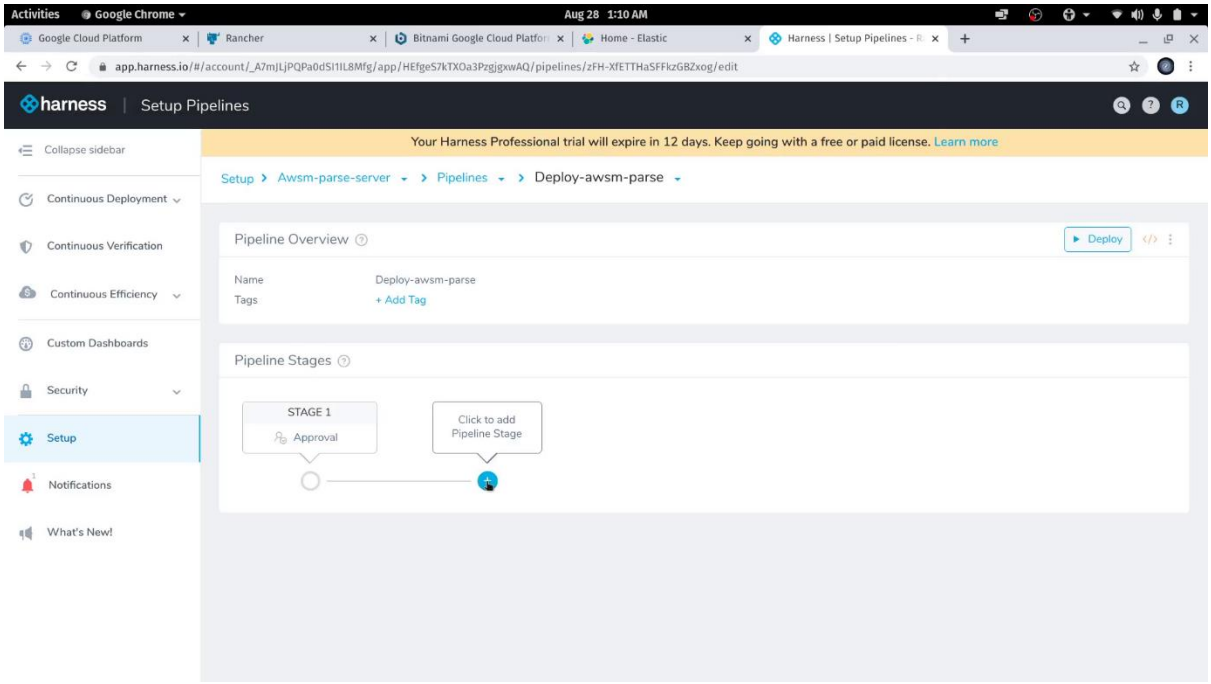
- Next, we are going to add an **Approval Stage**.



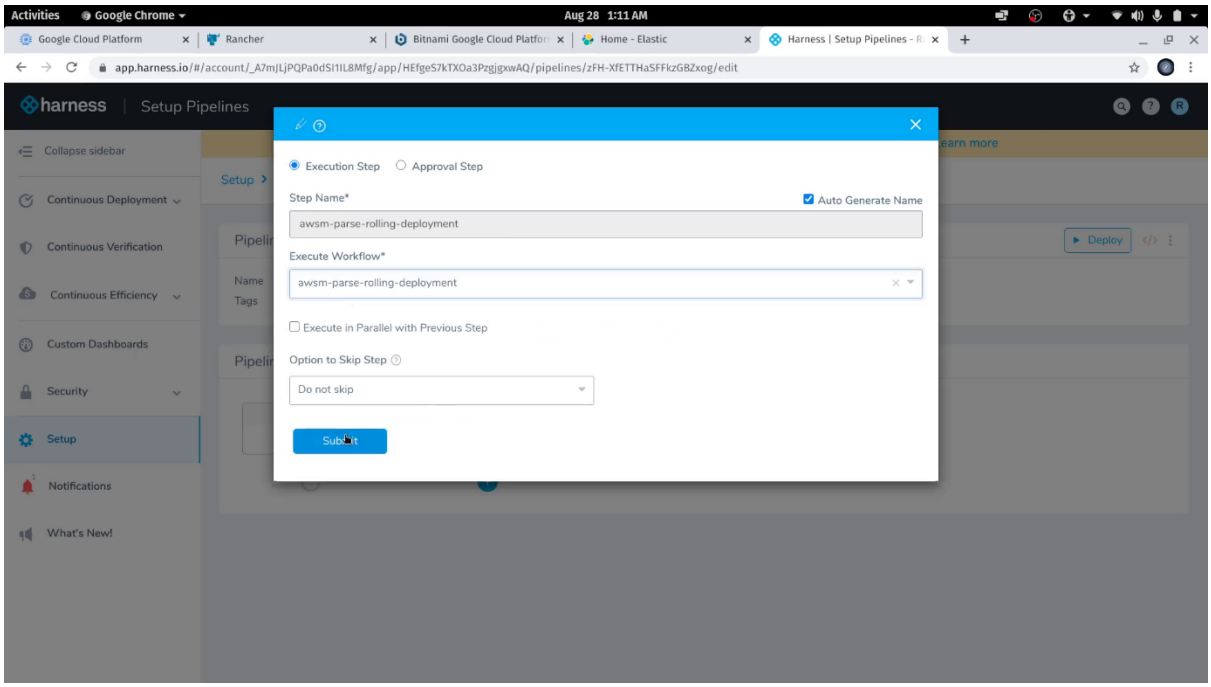
- Harness provides us with User Group(s) for enforcing RBAC (Role Based Access Control).
- Click **Submit**.



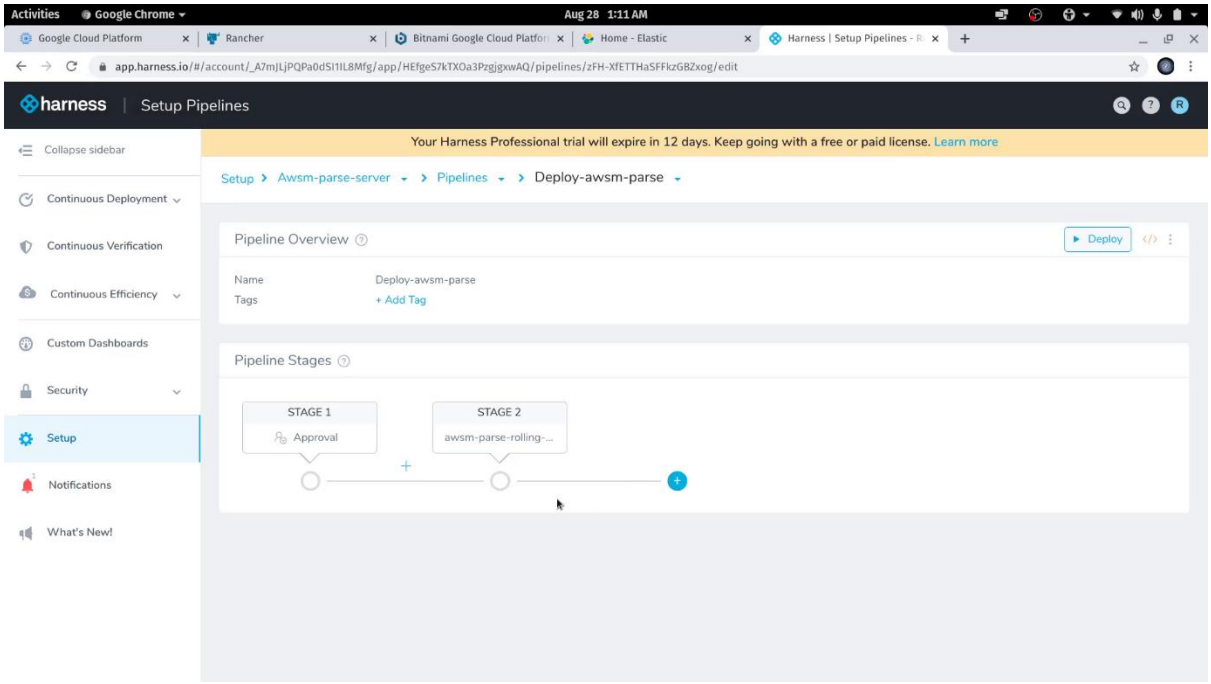
- Now let’s add the last stage of our pipeline and it is the **Execution Step**.



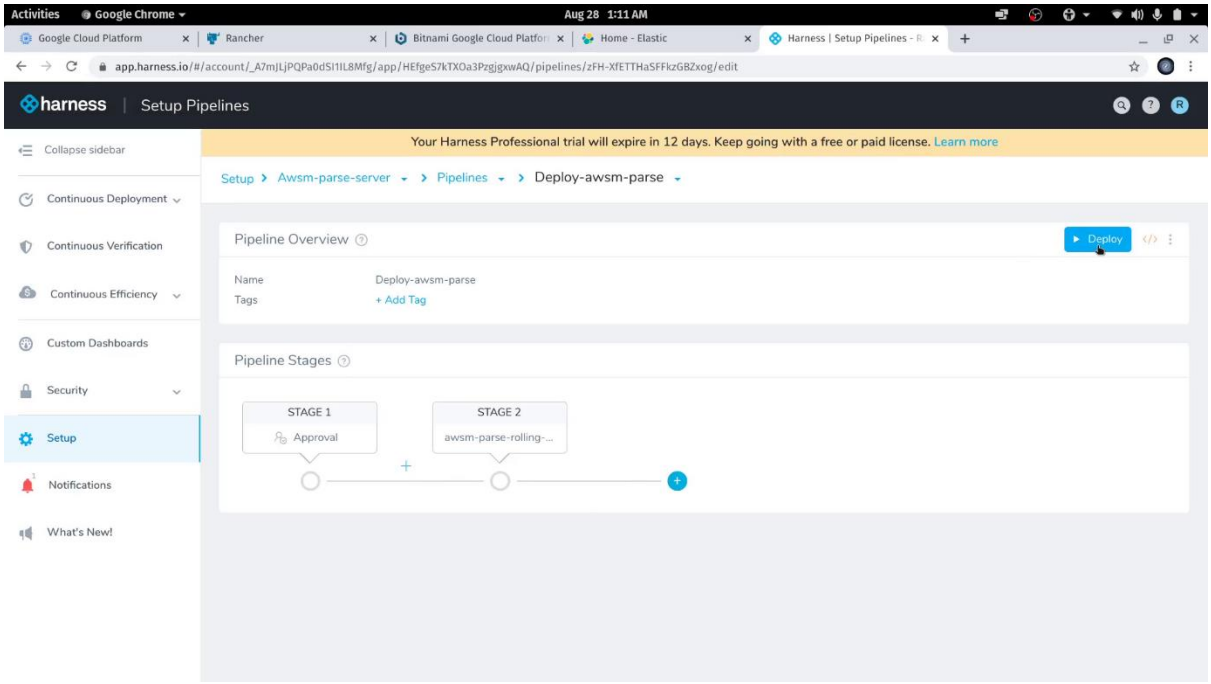
- Select our rolling deployment workflow which we created under **Execute Workflow**.
- Then, **Submit**.



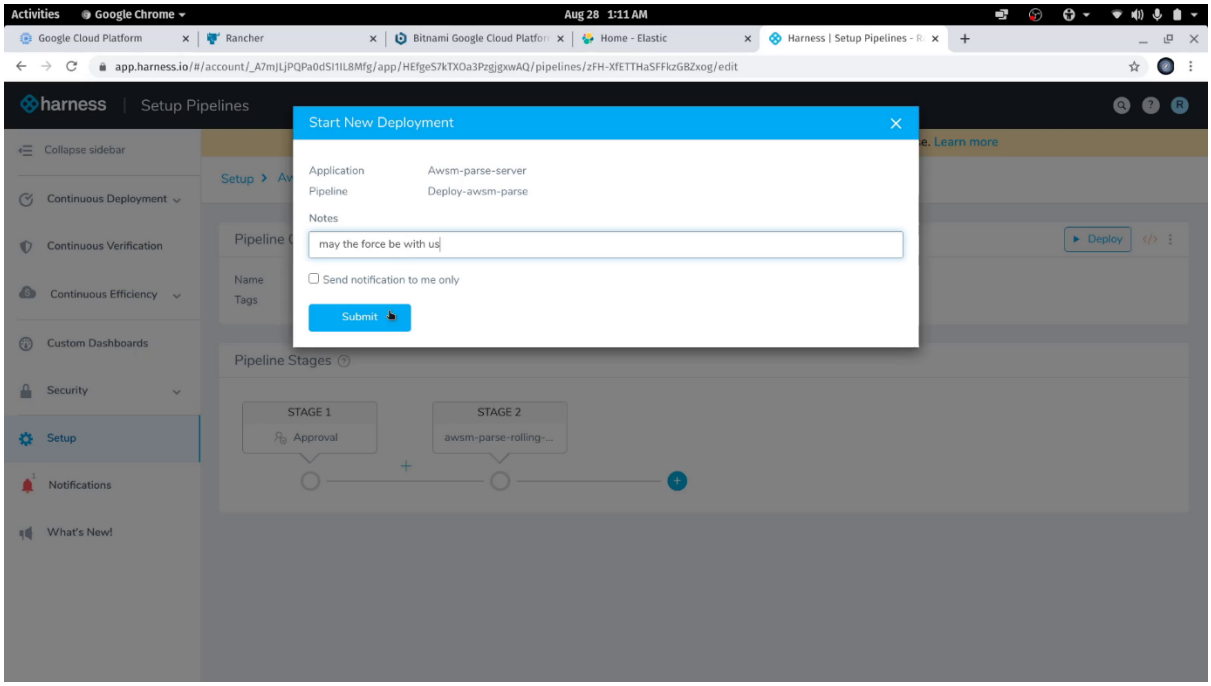
- We have successfully created our CD pipeline using Harness.



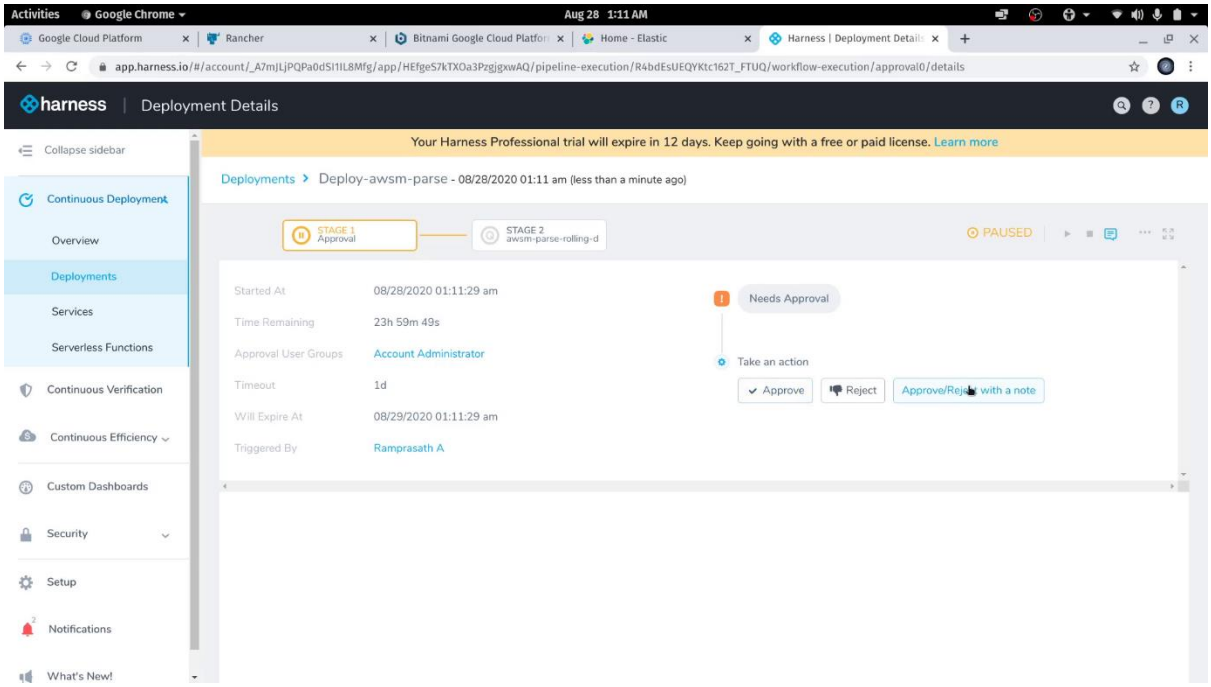
16. Now, let's deploy our Parse Server Application using rolling update strategy i.e. incremental update strategy.
    - Click **Deploy**.



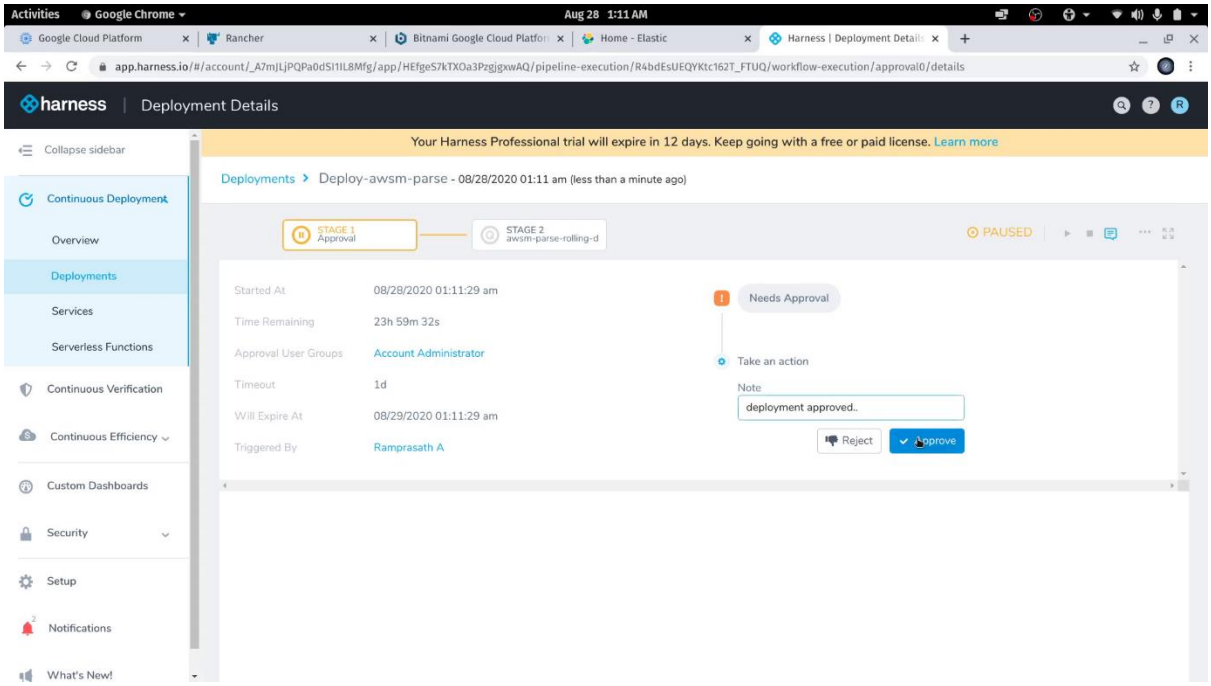
- Add a note if required and **Submit**.



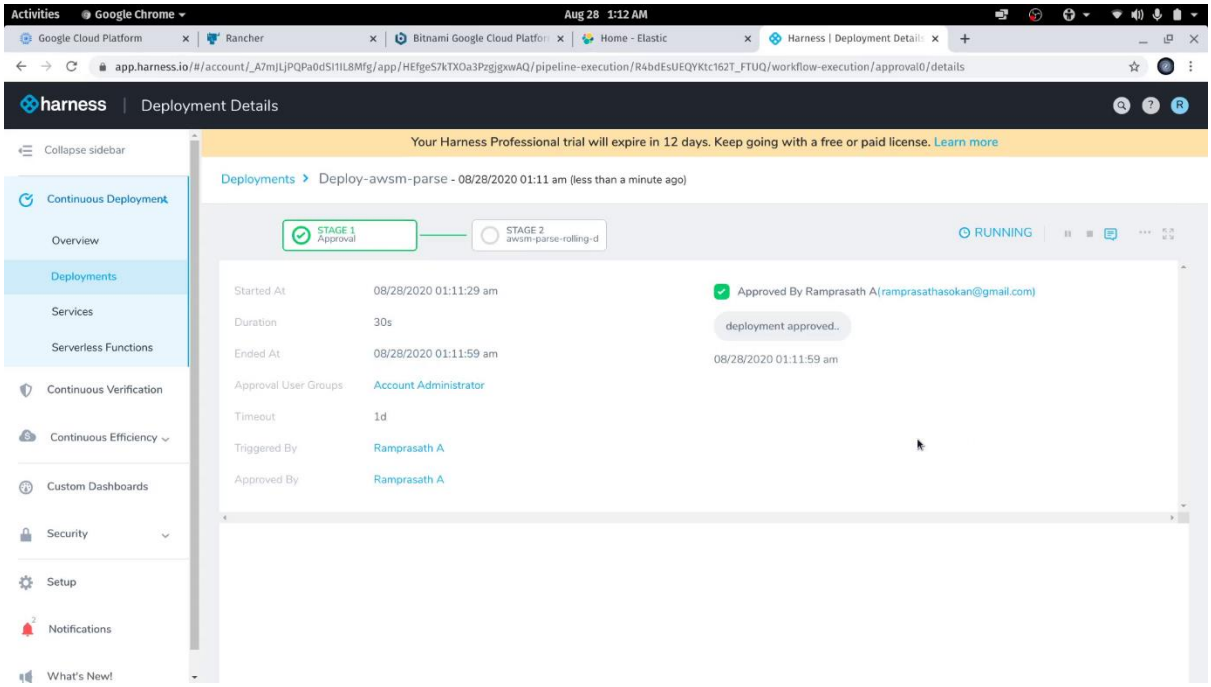
- Let’s approve with a note.
- Note: Once we trigger **Deploy**, an email will be sent by Harness notifying regarding deployment and its approval for the Deployment.



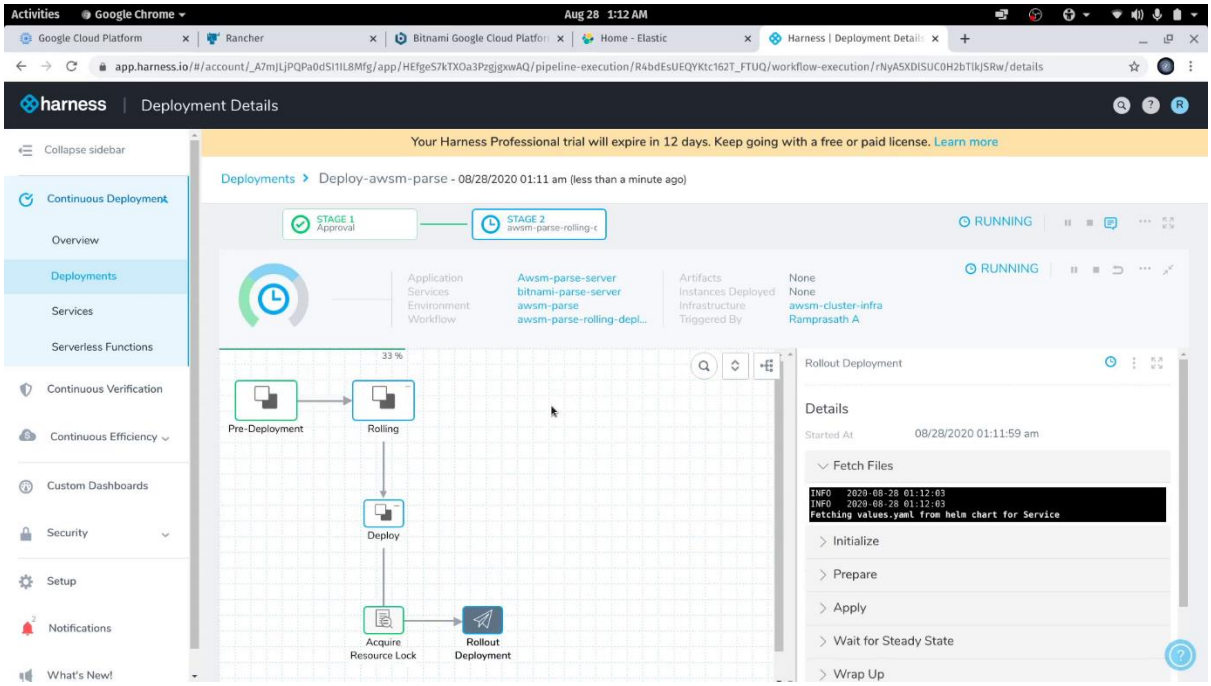
- Click **Approve**.



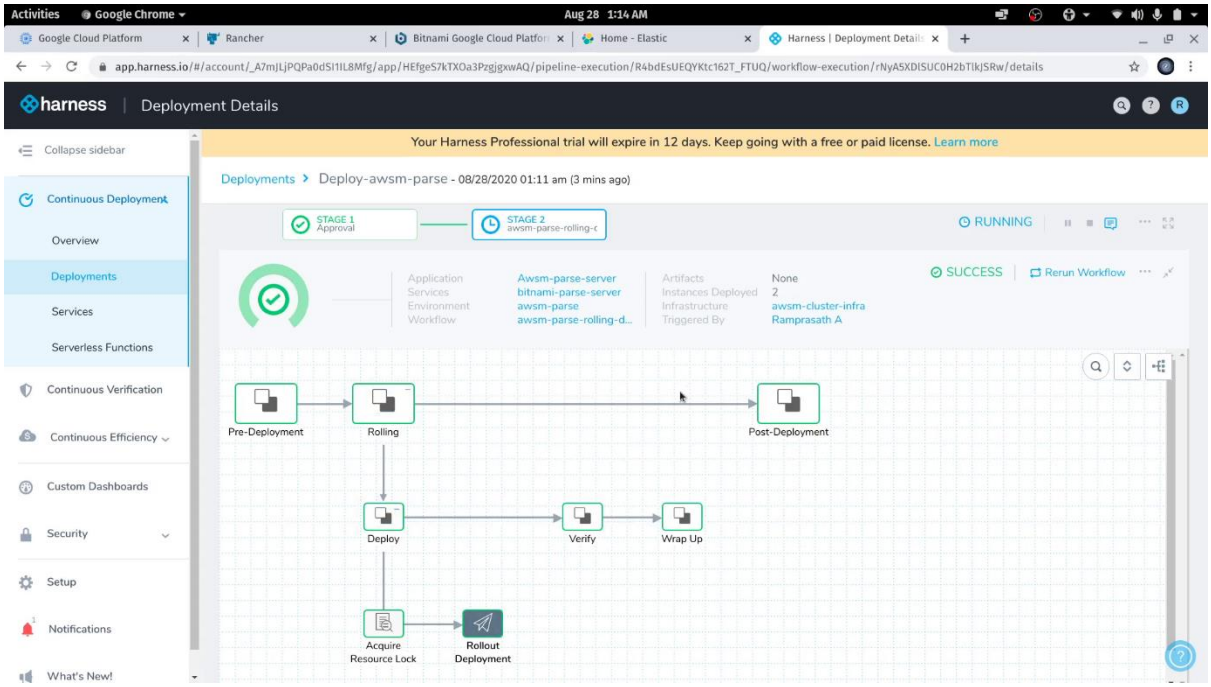
- Deployment has been approved.



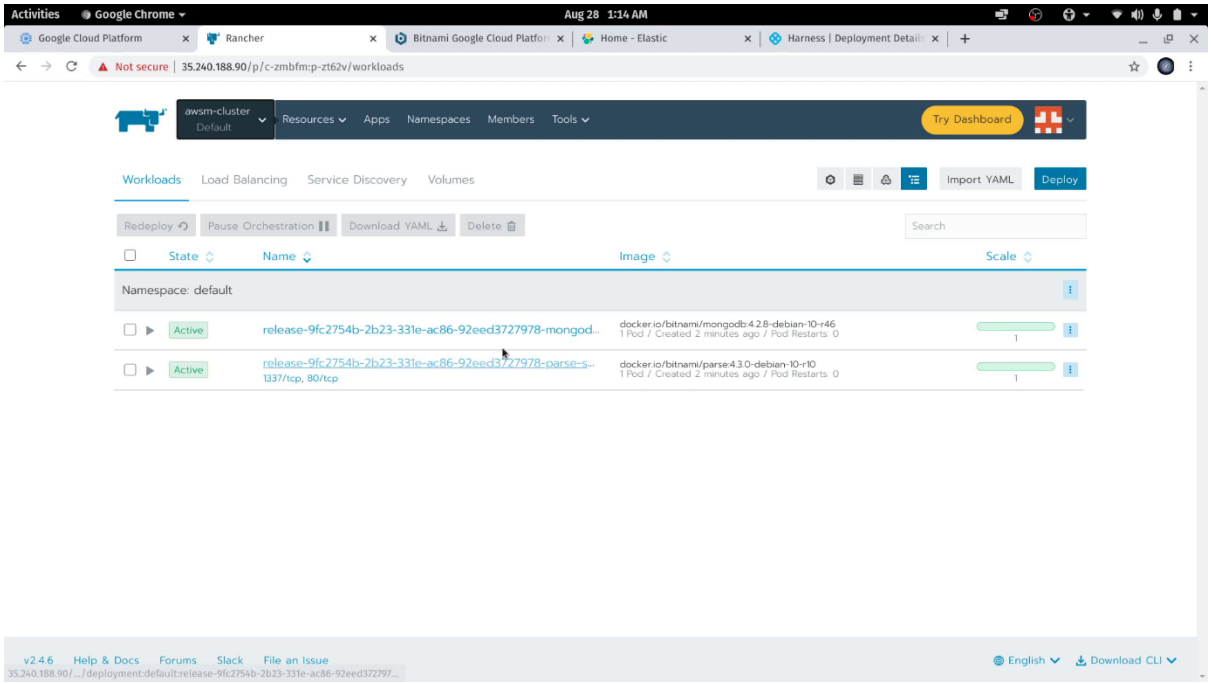
- Deployment in progress.



- Deployment successful.



- Let’s verify our deployment from **Rancher > Cluster > awsm-cluster > Workloads**



- We have successfully deployed our parse server application using rolling update strategy or incremental update strategy.
- We can rollback our deployment from Rancher if needed.

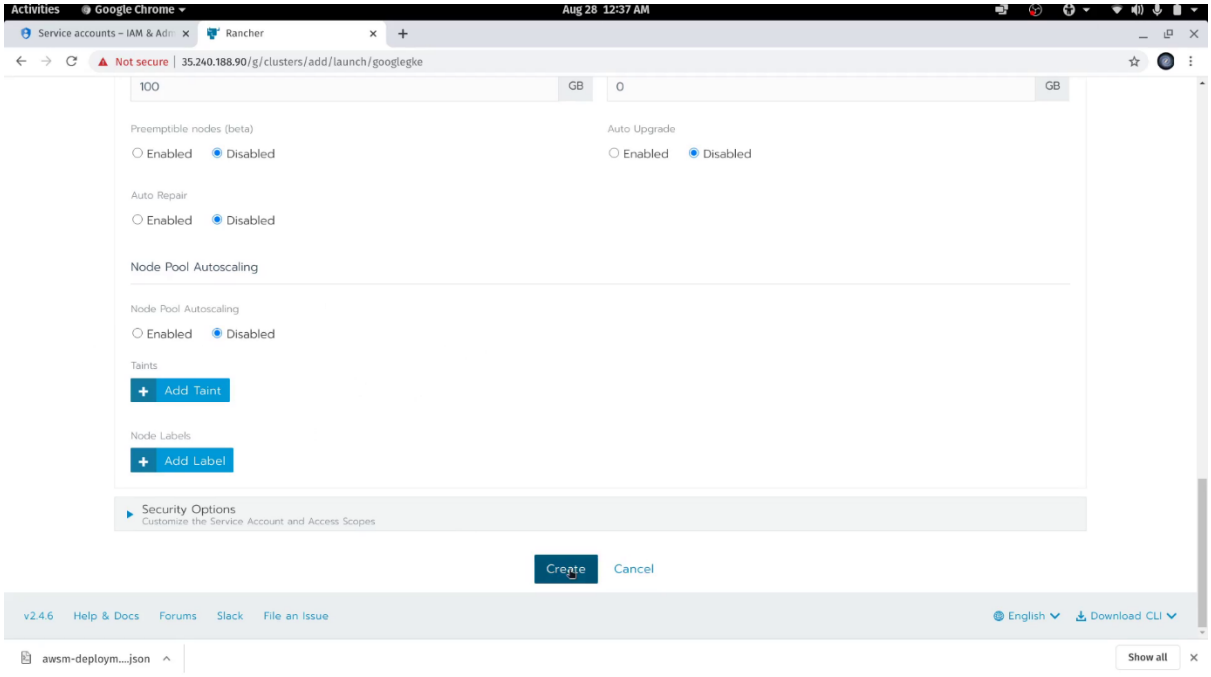
17. Note: Due to extensive time spent on learning new concepts and new tools from scratch, I was not able to setup Velero for backing up data.

Performance and Scaling

As far as horizontal scaling is concerned, we can do it manually after provisioning the Kubernetes cluster or enable node pool autoscaling feature while adding a new cluster from GKE. I was not able to enable node pool autoscaling due to the fact that I have a maximum quota of 8vCPUs per region in GCP and I have used all of them by provisioning 4 nodes.

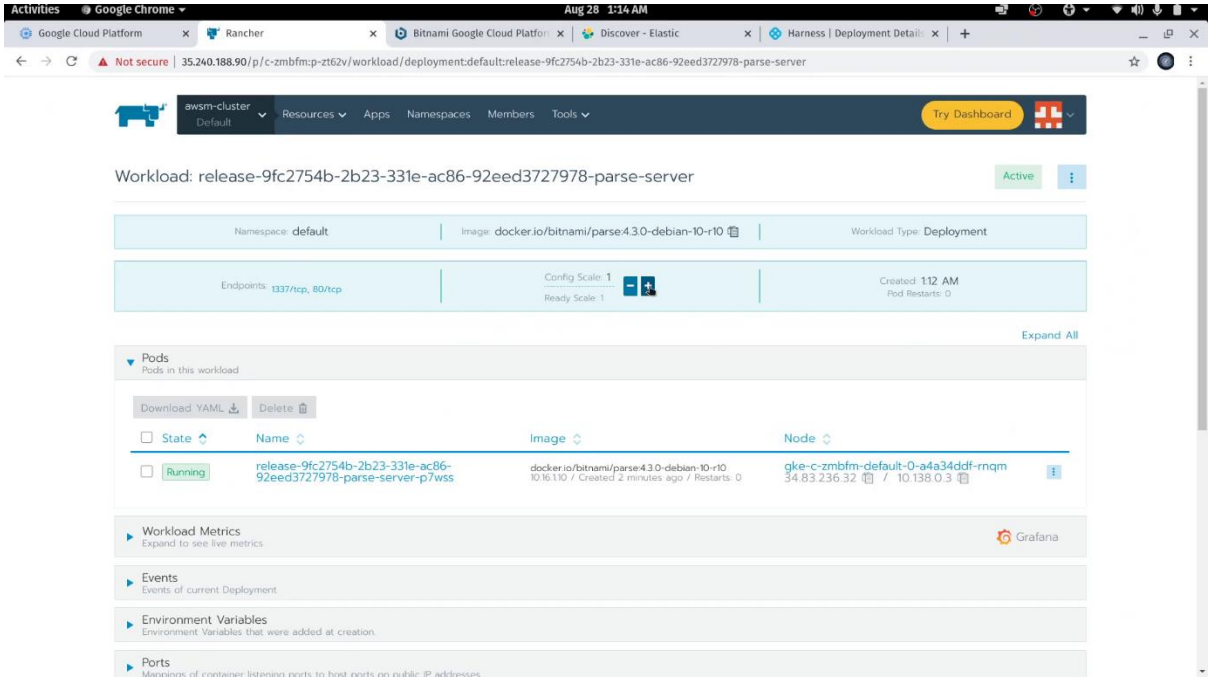
What node pool autoscaling feature does is increases or decreases the size of the node pool automatically, based on the resource requests (rather than actual resource utilization) of Pods running on that node pool's nodes.

To enable node pool autoscaling, select enable under node pool autoscaling

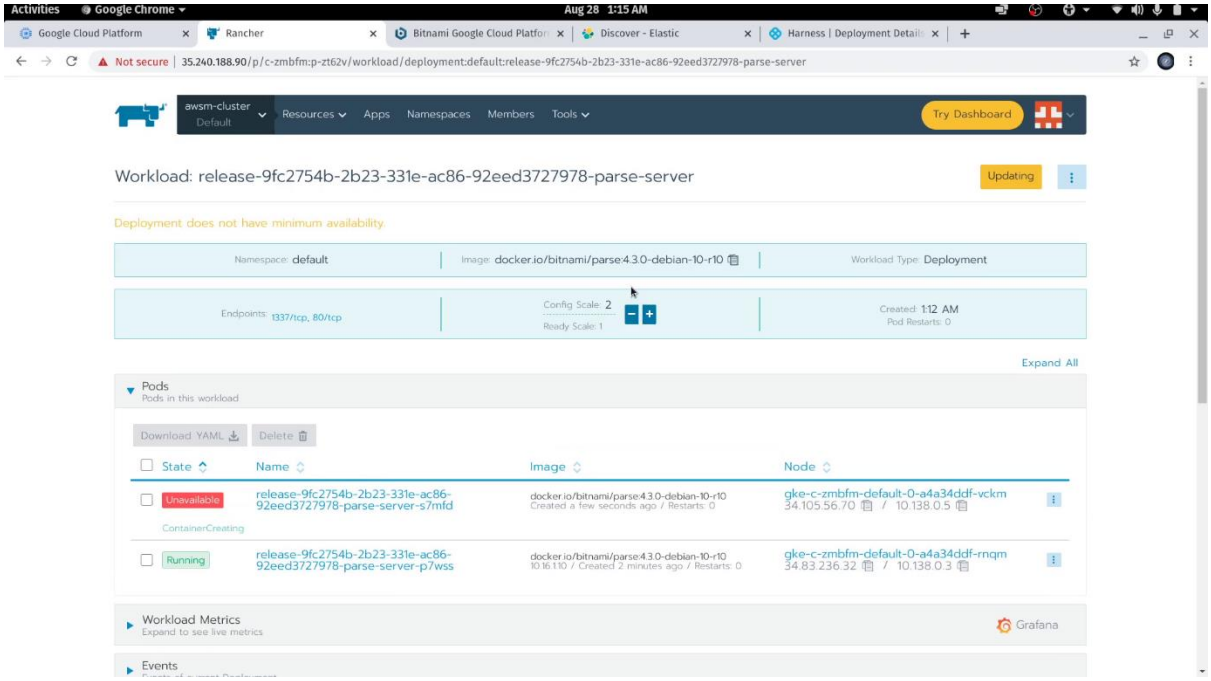


To manually scale a deployed app, we can increase the number of instances by the following steps:

- Go to **Rancher > Cluster > awsm-cluster > default (namespace) > workloads > release-id-parse-server**



- We can see that a new instance of a container is being created



Unresolved Issues / Recommended Approaches

1. Running Rancher in a Kubernetes cluster using RKE for high availability and scalability.
2. Running Harness delegate in a Kubernetes cluster for high availability and scalability.
3. Running ELK Stack in a self-hosted instance in GCP instead of using Bitnami Launchpad.
4. Enable node pool autoscaling while adding a new cluster for horizontal scaling.
5. Using self-hosted version of Harness to comply with privacy guidelines and to protect IP.
6. Incorporating OWASP ModSecurity WAF rules by adding annotation in Ingress if NGINX-Ingress is used as L4 load balancer.
7. Application source code protection can be achieved by using code obfuscation.

Other tools that I tried

1. GitLab
2. RKE (Rancher Kubernetes Engine)
3. Weave Cloud

References

1. https://rancher.com/docs/rancher/v2.x/en/
2. https://docs.harness.io/
3. https://www.elastic.co/guide/index.html
4. https://helm.sh/docs/