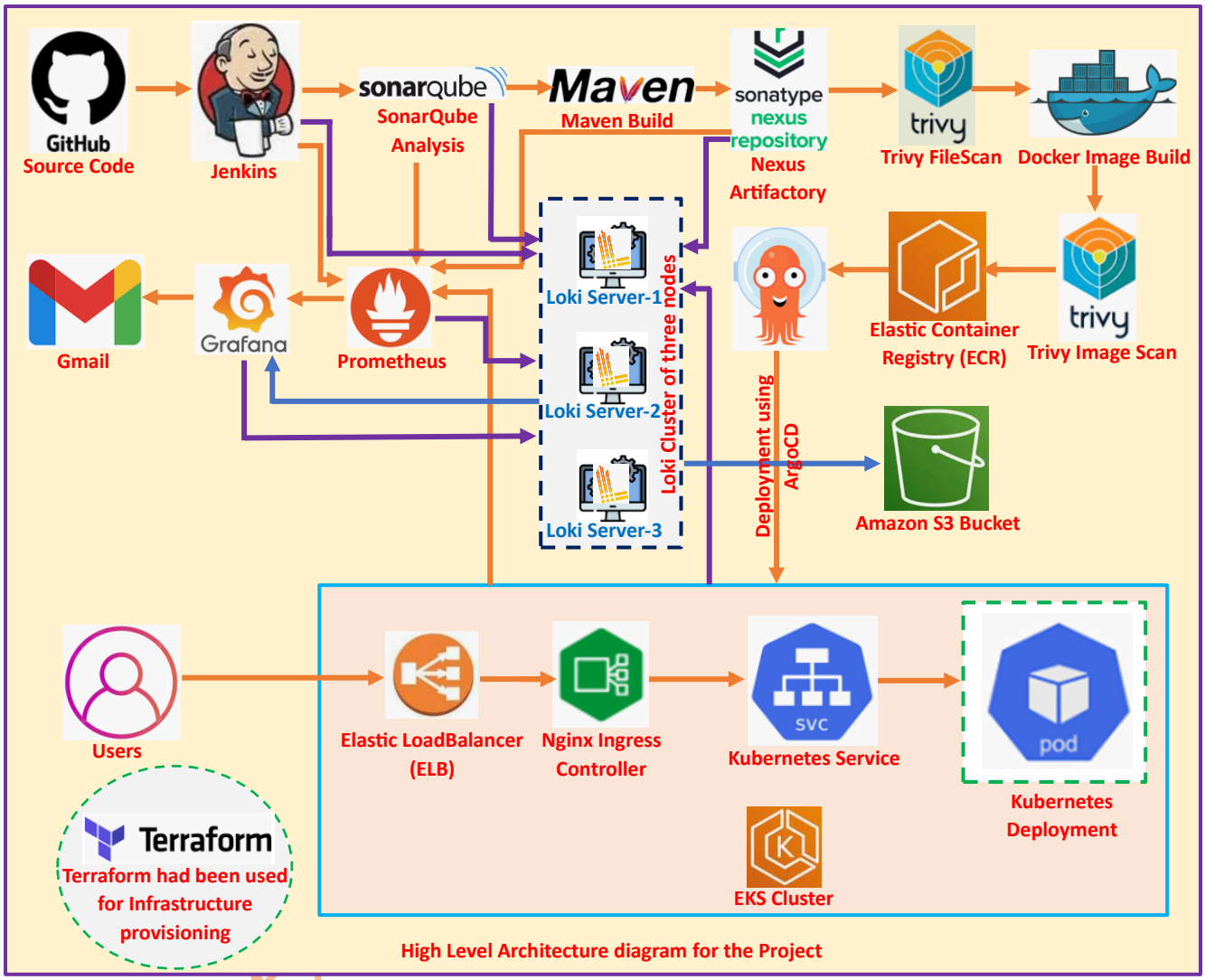
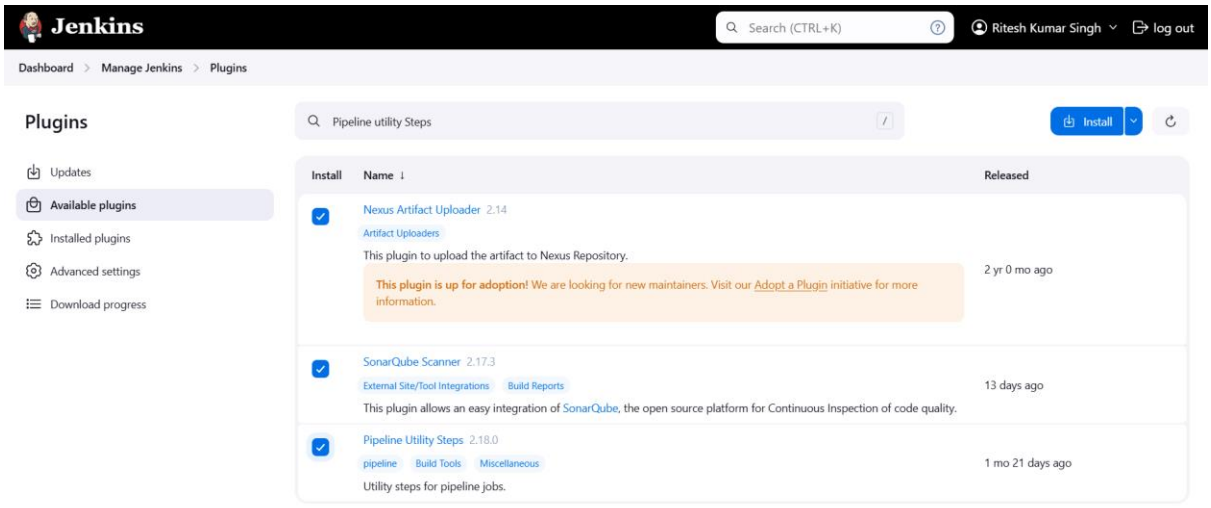
DevOps Project Blogging App Deployment Monitoring using Prometheus and Grafana and Log Aggregation using Loki, Promtail and Grafana



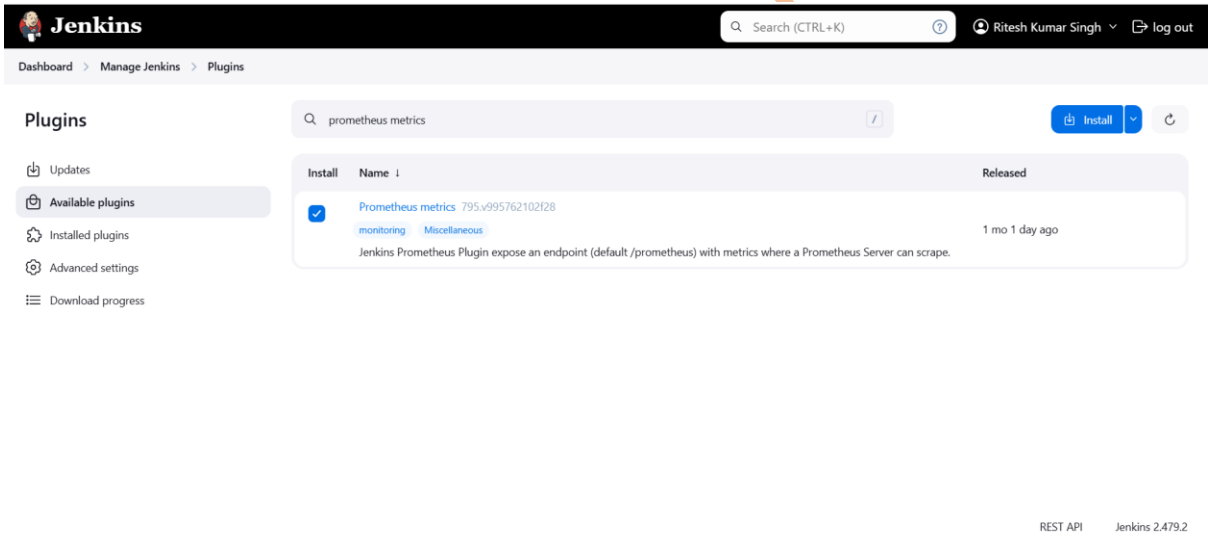
High Level Architecture diagram for the Project

This DevOps Project deals with creation of Infrastructure using Terraform and setup of CICD Pipeline using Jenkins, Monitoring using Prometheus and Grafana and Log Aggregation using Loki, Promtail and Grafana. SonarQube was used for Code-Analysis and Maven was used as the Build Tool. Nexus Artifactory was used to keep the Artifacts as shown in the Architecture diagram above. Trivy was used for FileScan and Docker Image Scan. The Docker Image was kept in the Elastic Container Registry (ECR) and which was deployed to EKS Cluster using the ArgoCD as shown in the high-level architecture diagram above. User was able to access the Application through the Ingress and hence the Kubernetes Service. For this project the source code was present in the GitHub Repository https://github.com/singhritesh85/Blogging-App.git.

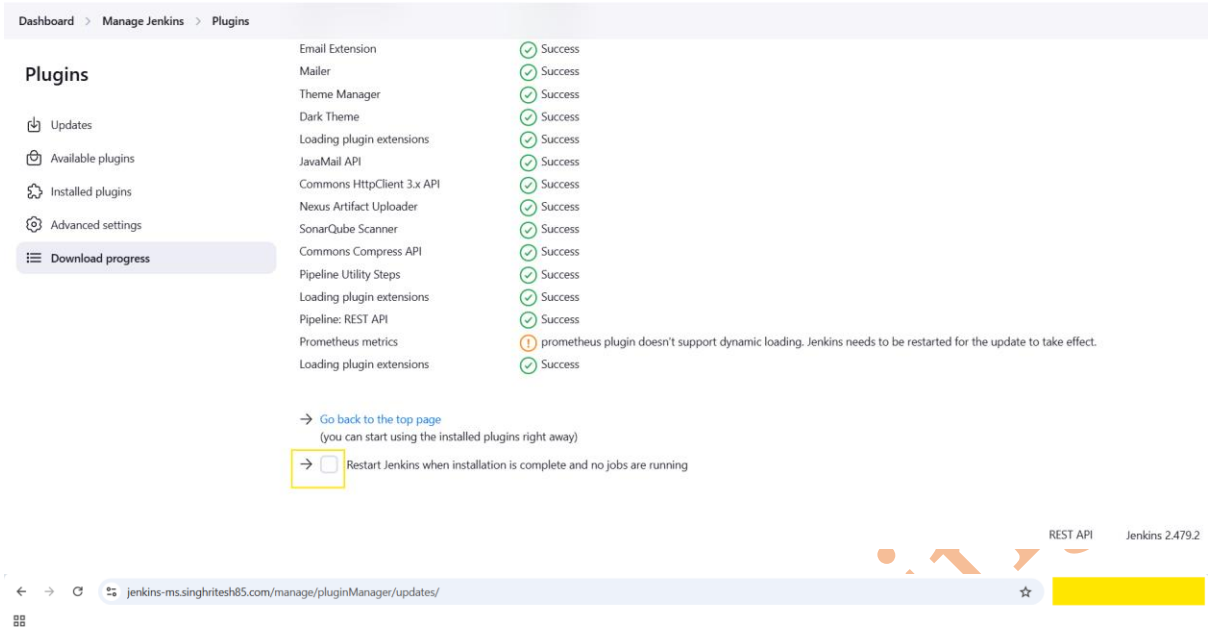
For this project I had installed Nexus Artifact Uploader, SonarQube Scanner and Pipeline Utility Step Plugin in the Jenkins as shown in the screenshot attached below.

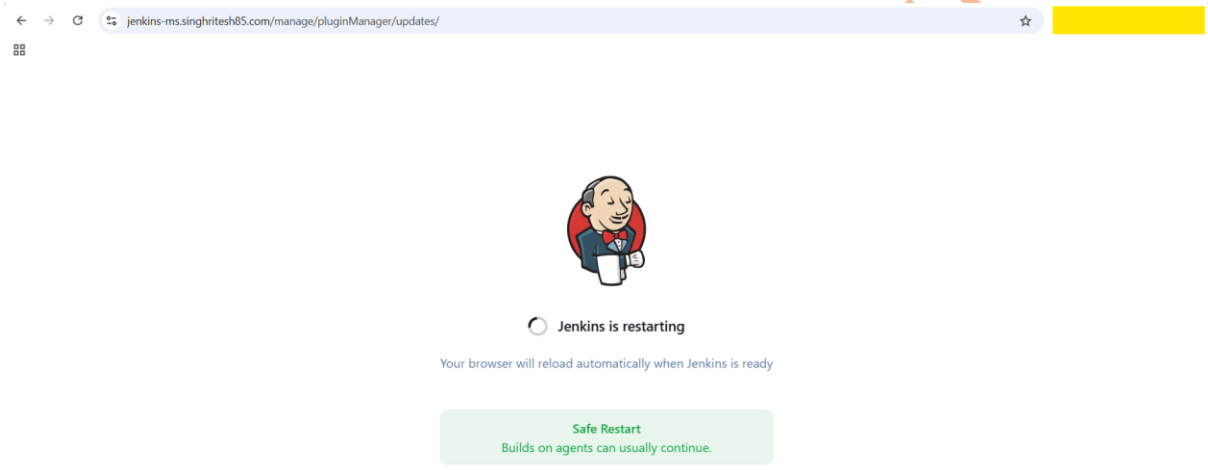


I had installed Prometheus metrics plugin in the Jenkins to monitor Jenkins Job through Prometheus and Grafana as shown in the screenshot attached below.

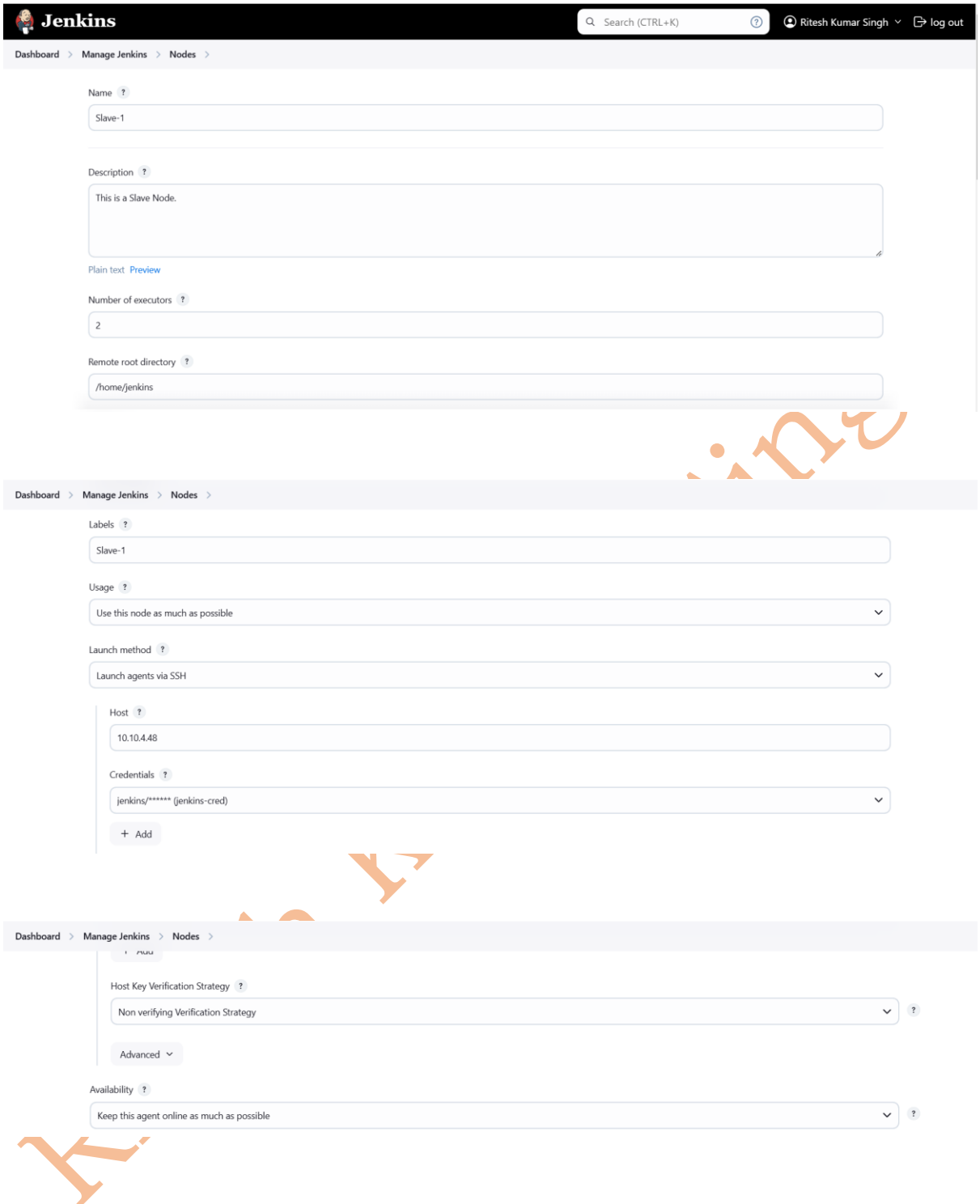


After Installation of Prometheus metrics as explained above in Jenkins I restarted the Jenkins as shown in the screenshot attached below.

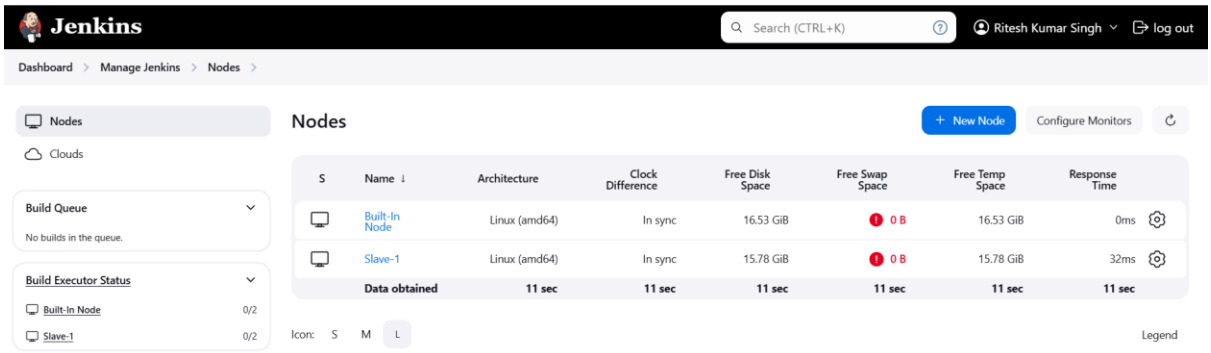




To Add Jenkins Slave with the Jenkins Master, follow the steps as written here. Go to **Manage Jenkins > Nodes**. Then provide the further details as mentioned in the screenshot attached below.



Finally, the Jenkins Slave was added to the Jenkins Master as shown in the screenshot attached below.



Installation of node-exporter and promtail had been done using the helm chart in the EKS Cluster as written below.

helm repo add prometheus-community https://*prometheus-community.github.io/helm-charts*

kubectl create ns node-exporter

helm install my-prometheus-node-exporter prometheus-community/prometheus-node-exporter --version 4.37.1 --set service.type=LoadBalancer -n node-exporter

```
[root@                ~]# helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
[root@                ~]# kubectl create ns node-exporter
[root@                ~]# helm install my-prometheus-node-exporter prometheus-community/prometheus-node-exporter --version 4.37.1 --set service.type=LoadBalancer -n node-exporter
```

Below screenshot shows the Kubernetes Service which was created for node-exporter using the helm chart.

```
[root@ip-10-10-4-48 ~]# kubectl get svc -n node-exporter
NAME                          TYPE           CLUSTER-IP      EXTERNAL-IP                                                              PORT(S)          AGE
my-prometheus-node-exporter   LoadBalancer   172.      .139   a                                      2.us-east-2.elb.amazonaws.com   9100:30875/TCP   10m
```

I had updated the prometheus configuration file, **/etc/prometheus/prometheus.yml** as shown in the screenshot attached below.

```
  - job_name: "EKS"
    static_configs:
      - targets: ["a                                      2.us-east-2.elb.amazonaws.com:9100"]
```

After updating the prometheus configuration file I restarted the prometheus service and checked the prometheus service status as shown in the screenshot attached below.

```
[root@                ~]# systemctl restart prometheus.service
[root@                ~]# systemctl status prometheus.service
● prometheus.service - Prometheus
   Loaded: loaded (/etc/systemd/system/prometheus.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon                                  s ago
```

I installed the promtail using the helm chart as shown in the screenshot attached below. After cloning helm chart from GitHub, I updated the values.yaml file of promtail helm chart with Loki Servers Private IP Addresses as shown in the screenshot attached below.

```
# -- The log level of the Promtail server
# Must be reference in `config.file` to configure `server.log_level`
# See default config in `values.yaml`
logLevel: info
# -- The log format of the Promtail server
# Must be reference in `config.file` to configure `server.log_format`
# Valid formats: `logfmt, json`
# See default config in `values.yaml`
logFormat: logfmt
# -- The port of the Promtail server
# Must be reference in `config.file` to configure `server.http_listen_port`
# See default config in `values.yaml`
serverPort: 3101
# -- The config of clients of the Promtail server
# Must be reference in `config.file` to configure `clients`
# @default -- See `values.yaml`
clients:
  - url: http://10.        :3100/loki/api/v1/push
  - url: http://10.        :3100/loki/api/v1/push
  - url: http://10.        :3100/loki/api/v1/push

# -- Configures where Promtail will save it's positions file, to resume reading after restarts.
# Must be referenced in `config.file` to configure `positions`
positions:
  filename: /run/promtail/positions.yaml
# -- The config to enable tracing
enableTracing: false
# -- A section of reusable snippets that can be reference in `config.file`.
# Custom snippets may be added in order to reduce redundancy.
```

git clone https://github.com/singhritesh85/helm-chart-promtail.git

kubectl create ns promtail && helm upgrade --install promtail ./helm-chart-promtail -f ./helm-chart-promtail/values.yaml -n promtail

kubectl get pods -n promtail --watch

```
[root@              ~]# git clone https://github.com/singhritesh85/helm-chart-promtail.git
```

```
[root@              ~]# kubectl create ns promtail && helm upgrade --install promtail ./helm-chart-promtail -f ./helm-chart-promtail/values.yaml -n promtail
namespace/promtail created
```

```
[root@              ~]# kubectl get pods -n promtail --watch
NAME                READY   STATUS    RESTARTS   AGE
promtail-m     5    1/1     Running   0          32s
promtail-x     2    1/1     Running   0          32s
```

I had provided restricted access to the deployment user **jenkins** using Service Account, Role and Role Binding as shown in the screenshot attached below. The deployment user had all the accesses in the

namespace **blogapp** but does not had access for the entire EKS cluster. That means deployment user jenkins access was restricted to the namespace **blogapp** in the EKS Cluster.

```
[root@ip-10-10-4-48 ~]# cat sa-role-rolebinding.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins
  namespace: blogapp
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: user-role
  namespace: blogapp
rules:
  - apiGroups: ["*"]
    resources: ["*"]
    verbs: ["*"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: user-rolebinding
  namespace: blogapp
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: user-role
subjects:
- namespace: blogapp
  kind: ServiceAccount
  name: jenkins
```

```
[root@                ~]# kubectl apply -f sa-role-rolebinding.yaml
serviceaccount/jenkins created
role.rbac.authorization.k8s.io/user-role created
rolebinding.rbac.authorization.k8s.io/user-rolebinding created
```

Created Kubernetes Secrets Which Token was utilized in the kubeconfig file (which was shared with the deployment user jenkins) as shown in the screenshot attached below.

```
[root@ ~]# cat secrets.yaml
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: mysecretname
  namespace: blogapp
  annotations:
    kubernetes.io/service-account.name: jenkins
```

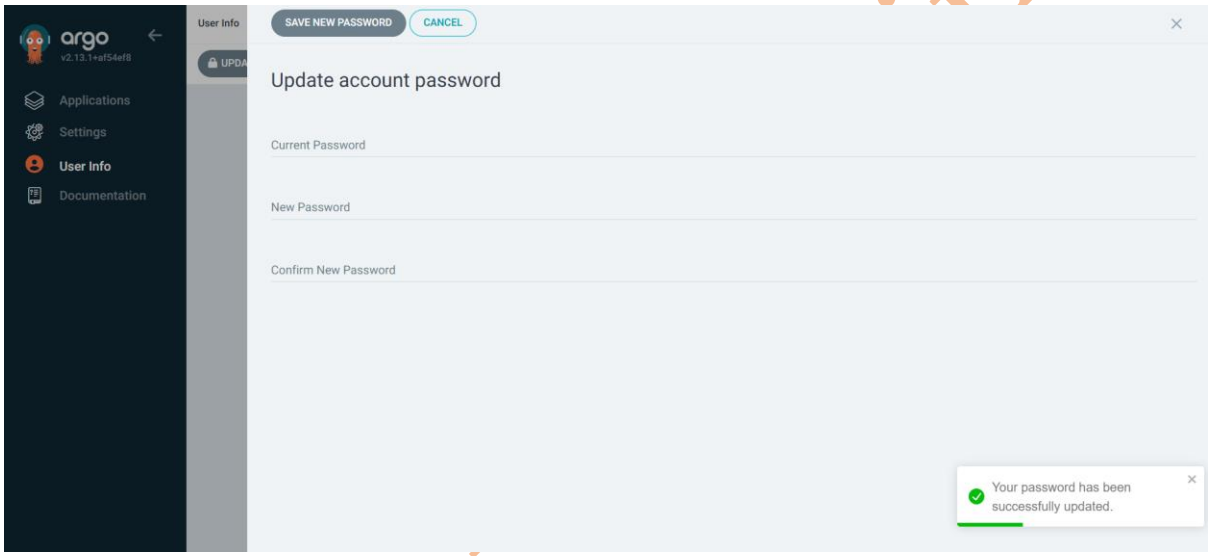
```
[root@ ~]# kubectl apply -f secrets.yaml
secret/mysecretname created
[root@ ~]# kubectl get secrets -n blogapp
NAME           TYPE                                  DATA   AGE
mysecretname   kubernetes.io/service-account-token   3      5s
[root@ ~]# kubectl describe secrets mysecretname -n blogapp
Name:         mysecretname
Namespace:    blogapp
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: jenkins
              kubernetes.io/service-account.uid: 1

Type:  kubernetes.io/service-account-token

Data
====
namespace:  7 bytes
token:      e
ca.crt:     1107 bytes
```

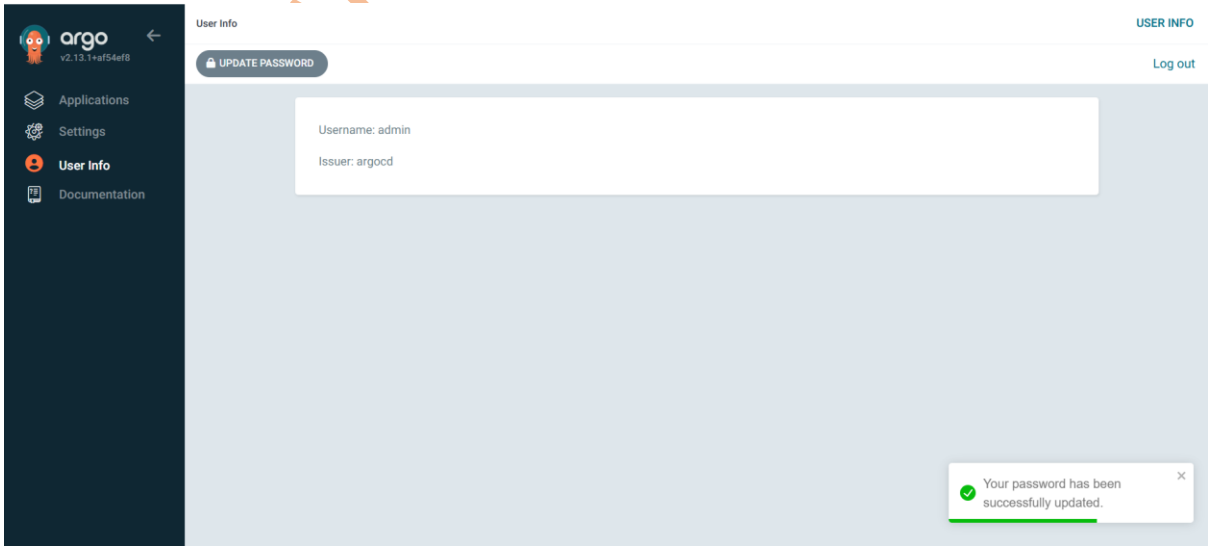
```
[jenkins@ip-10-10-4-48 ~]$ cat ~/.kube/config
clusters:
- cluster:
    certificate-authority-data: L
    server: https://4 F.gr7.us-east-2.eks.amazonaws.com
  name: arn:aws:eks:us-east-2:0 5:cluster/eks-demo-cluster-dev
contexts:
- context:
    cluster: arn:aws:eks:us-east-2:0 6:cluster/eks-demo-cluster-dev
    user: jenkins
  name: dexter
current-context: dexter
kind: Config
preferences: {}
users:
- name: jenkins
  user:
    token: e
```

Kubernetes Ingress for ArgoCD was creates using the Ingress Rule as shown in the screenshot attached below.

```
[root@                ~]# cat argocd-ingress-rule.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  namespace: argocd
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"    ###  You can use this option for this particular case for ArgoCD but not for all
#    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  ingressClassName: nginx
  rules:
  - host: argocd.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: argocd-server   ### Provide your service Name
            port:
              number: 80     #### Provide your service port for this particular example you can also choose 443
```

I had updated the ArgoCD password as shown in the screenshot attached below.



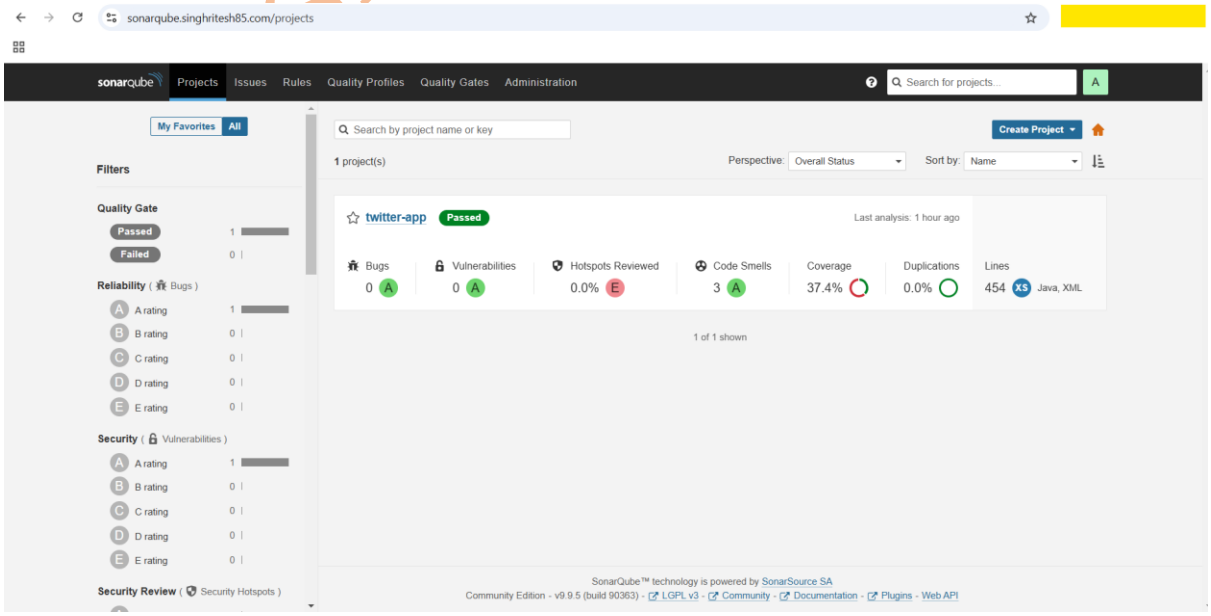


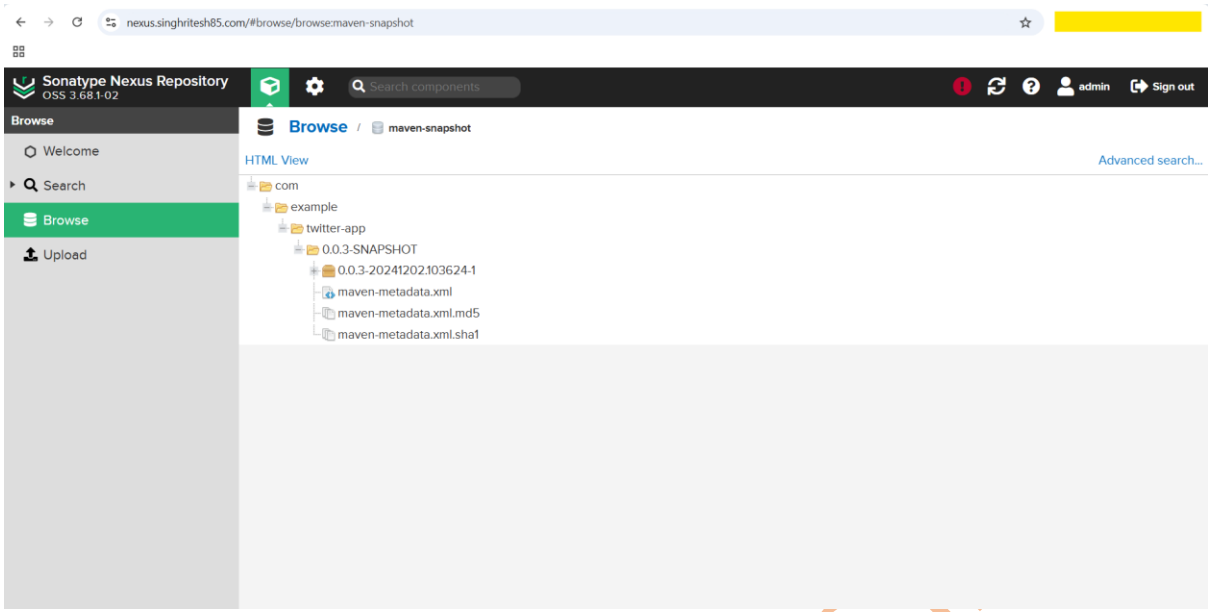
Kubernetes Ingress for Blogging Application was created using the Ingress Rule as shown in the screenshot attached below.

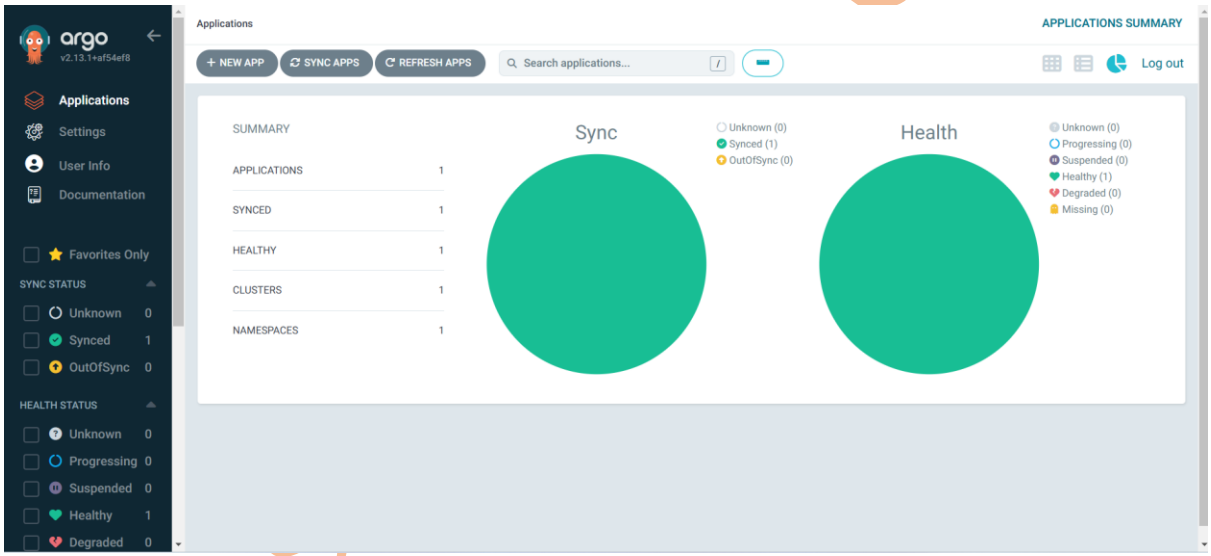
```
[root@                ~]# cat ingress-rule.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: blogapp-ingress
  namespace: blogapp
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: blogapp.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: blogapp-folo
            port:
              number: 80
```

```
[root@            ~]# kubectl get ing -A
NAMESPACE   NAME              CLASS   HOSTS                     ADDRESS                                                  PORTS   AGE
argocd      minimal-ingress   nginx   argocd.singhritesh85.com  a                                  8.us-east-2.elb.amazonaws.com  80      4h9m
blogapp     blogapp-ingress   nginx   blogapp.singhritesh85.com a                                  8.us-east-2.elb.amazonaws.com  80      3h56m
```

The Screenshot for SonarQube, Nexus and ArgoCD after running the Jenkins Job is as shown in the screenshot attached below.





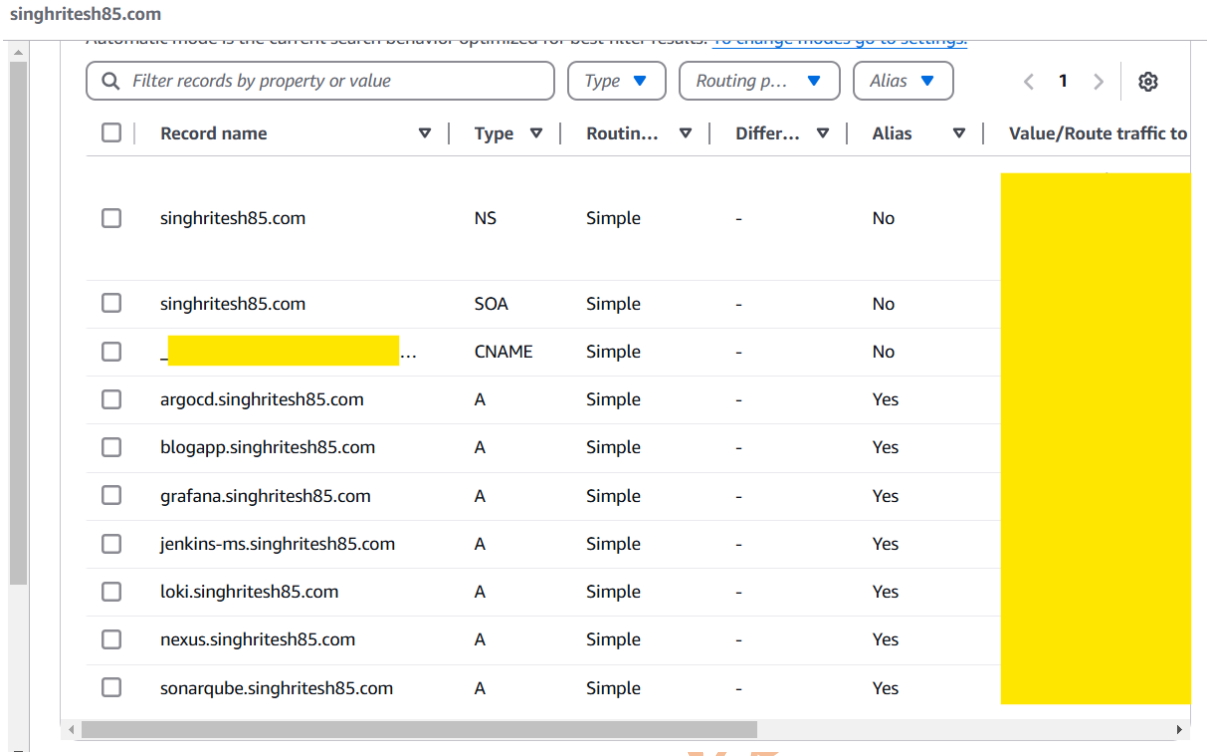


Below screenshot show Jenkins Job after its successful execution.

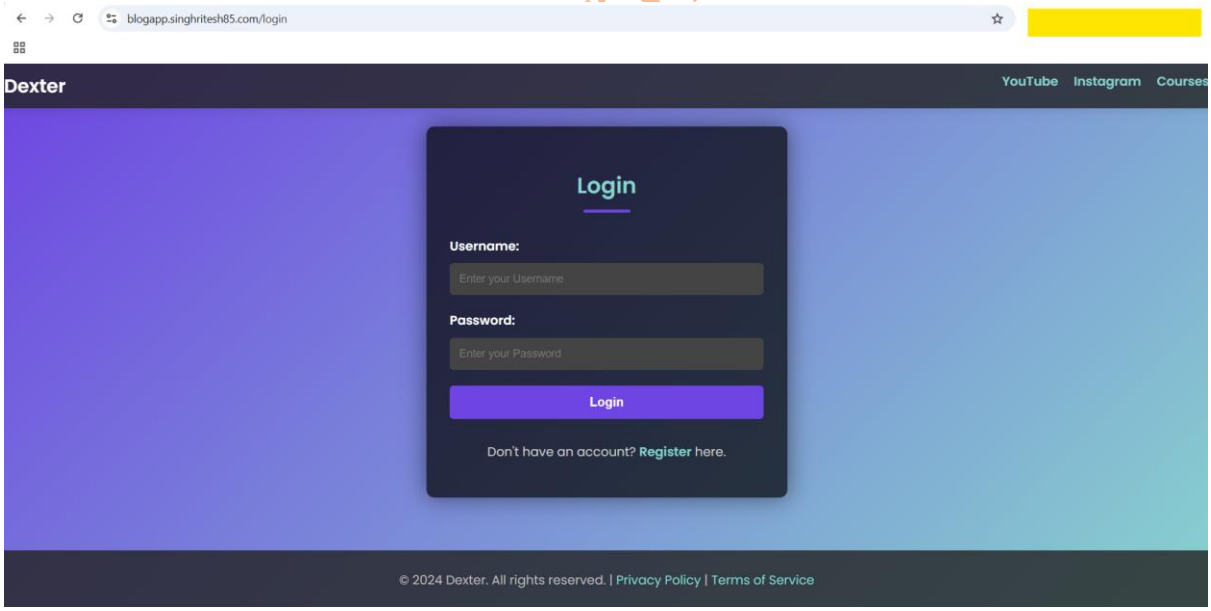
After successful execution of Jenkins Job Kubernetes Pod, Service and Deployment was created.

```
[jenkins@                ~]$ kubectl get all -n blogapp
NAME                                 READY   STATUS    RESTARTS   AGE
pod/blogapp-folo-7                   1/1     Running   0          126m

NAME                   TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/blogapp-folo   ClusterIP   172.    .192   <none>        80/TCP    136m

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/blogapp-folo   1/1     1            1           136m

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/blogapp-folo-7          d   0         0         0       136m
replicaset.apps/blogapp-folo-7          4   1         1         1       126m
[jenkins@                ~]$ kubectl get nodes
Error from server (Forbidden): nodes is forbidden: User "system:serviceaccount:blogapp:jenkins" cannot list resource "nodes" in API group "" at the cluster sc
ope
```

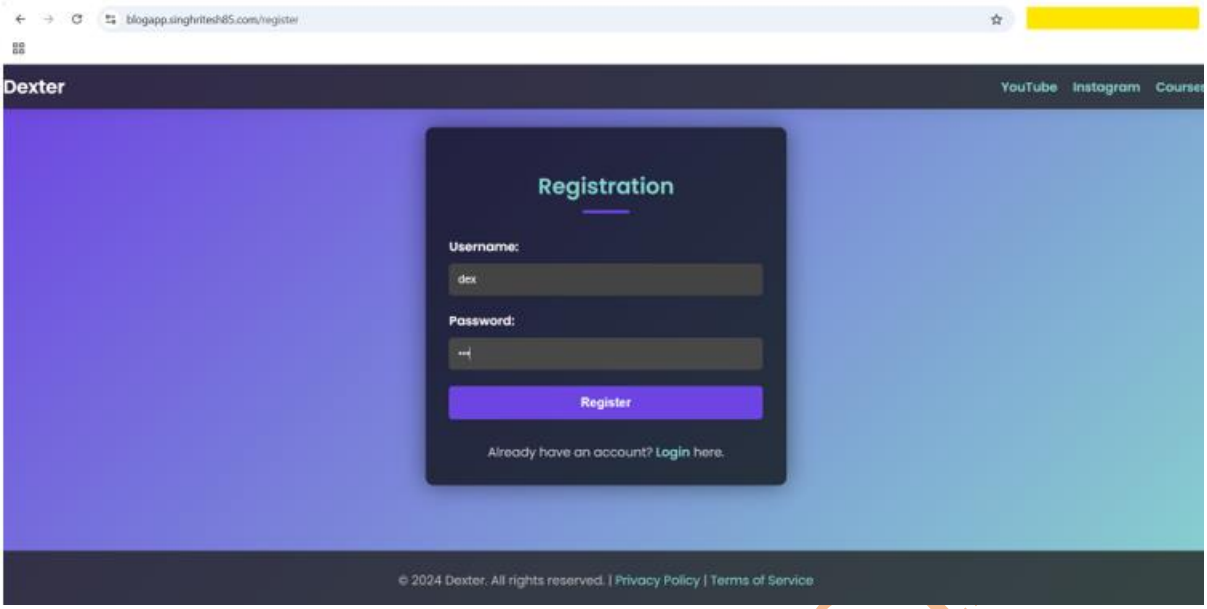
Below screenshot shows the entry for Route53 to create the Record Set.

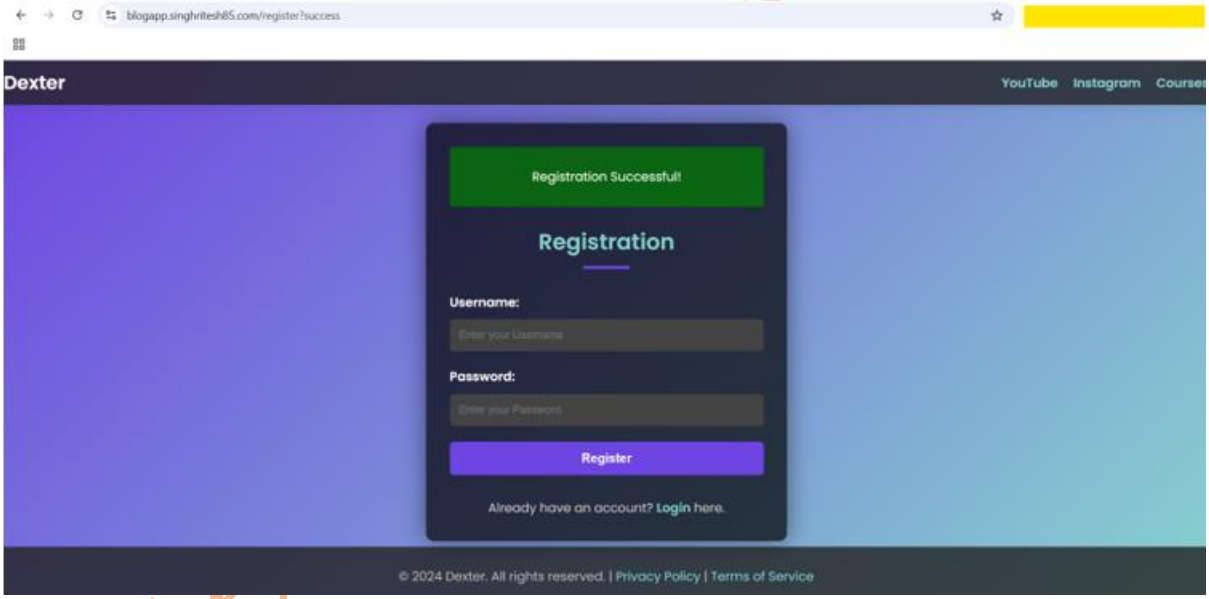


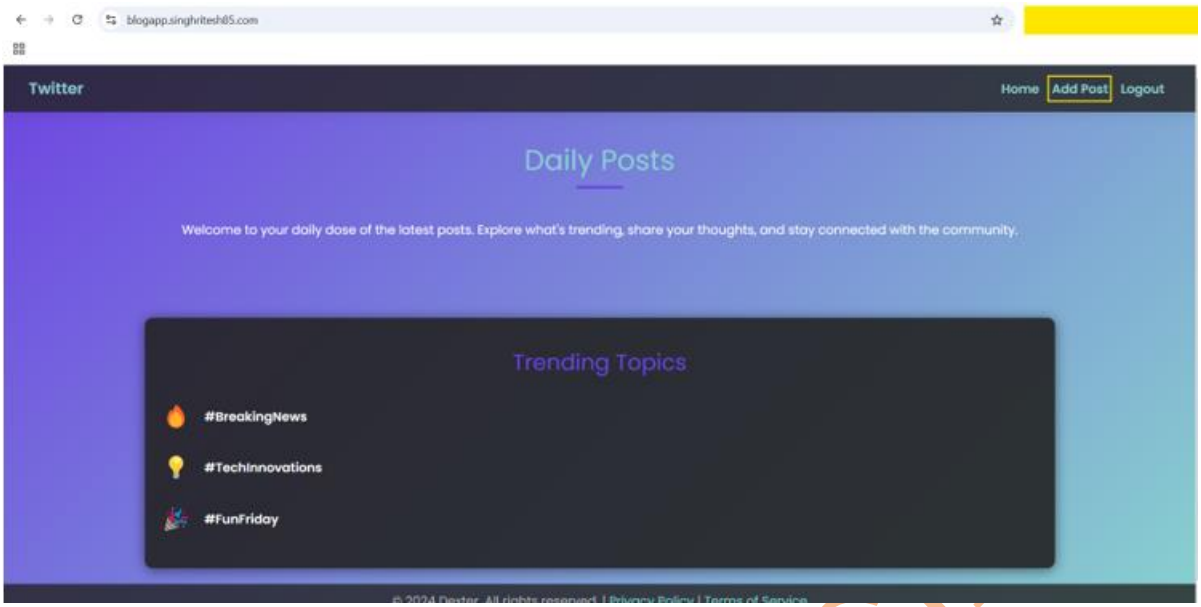
Finally, using the URL I started accessing the Application.

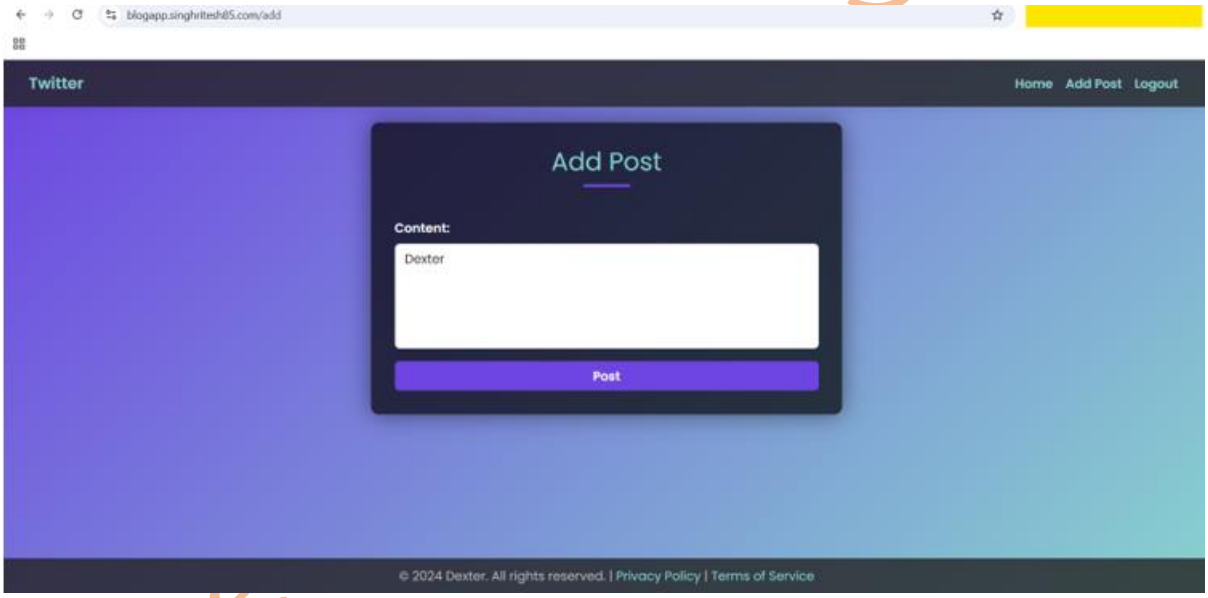


I registered a new user and logged in through that user and checked that I was able to read the blogs or create a new blogs.


blogapp.singhritesh85.com/register
Dexter
YouTube
Instagram
Registration
Username:
dex
Password:
Register
Already have an account? Login here.
© 2024 Dexter. All rights reserved. | Privacy Policy | Terms of Service


blogapp.singhritesh85.com/register?success
Dexter
YouTube
Instagram
Registration Successful!
Registration
Username:
Password:
Register
Already have an account? Login here.
© 2024 Dexter. All rights reserved. | Privacy Policy | Terms of Service

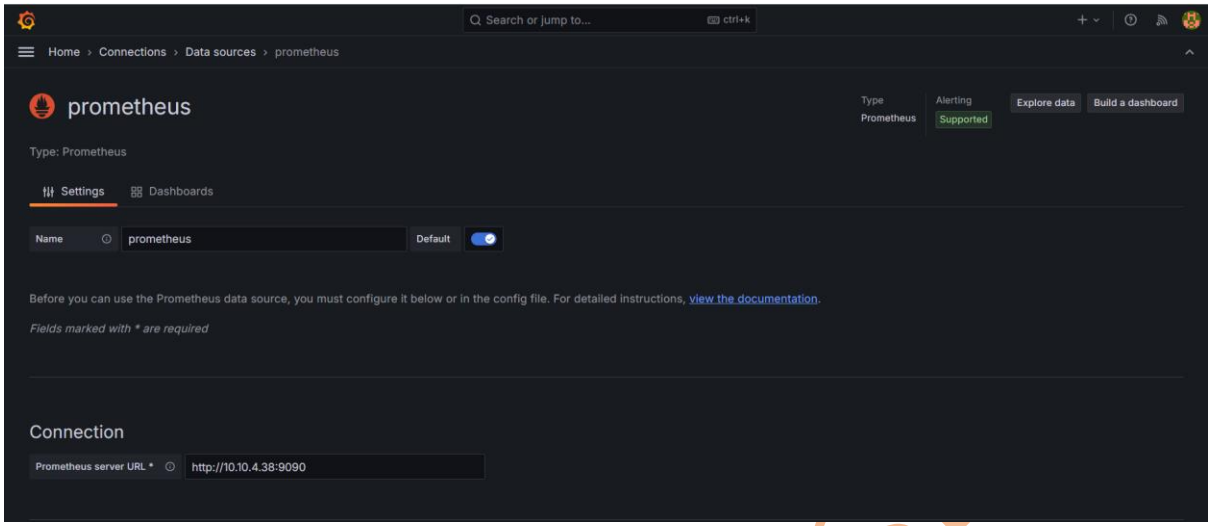


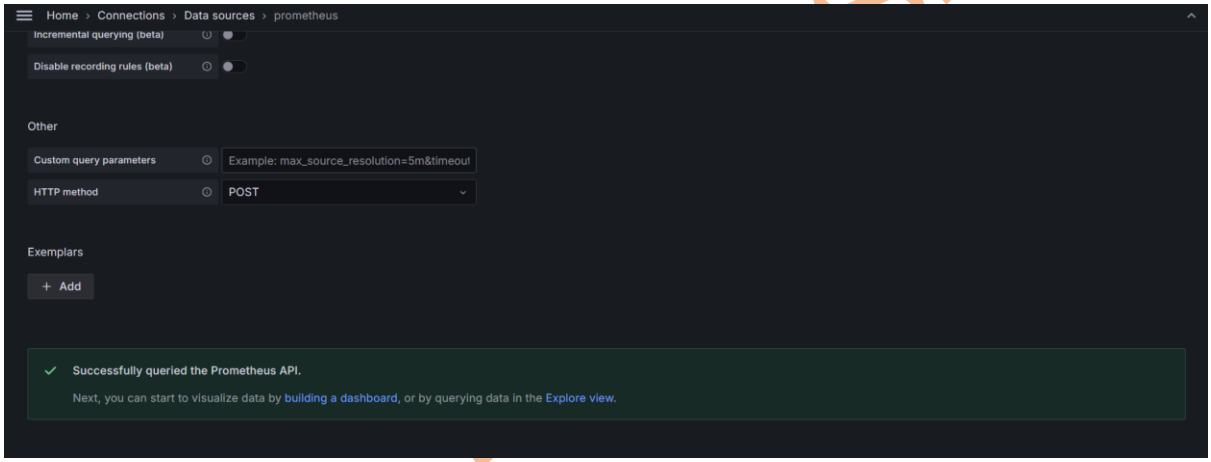


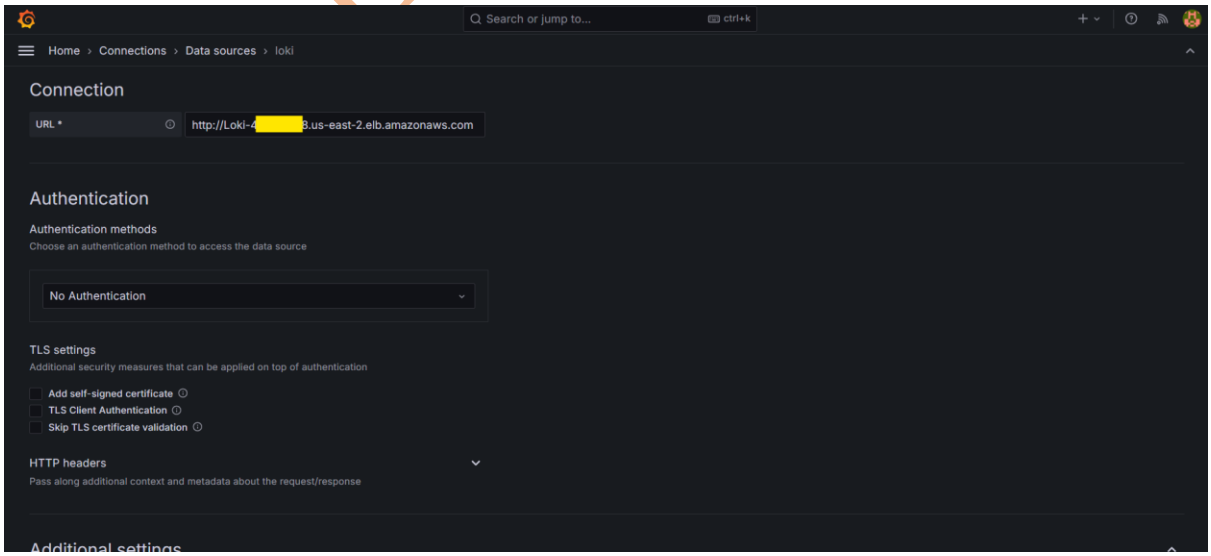
Monitoring Using Prometheus and Grafana and Log Aggregation using Loki

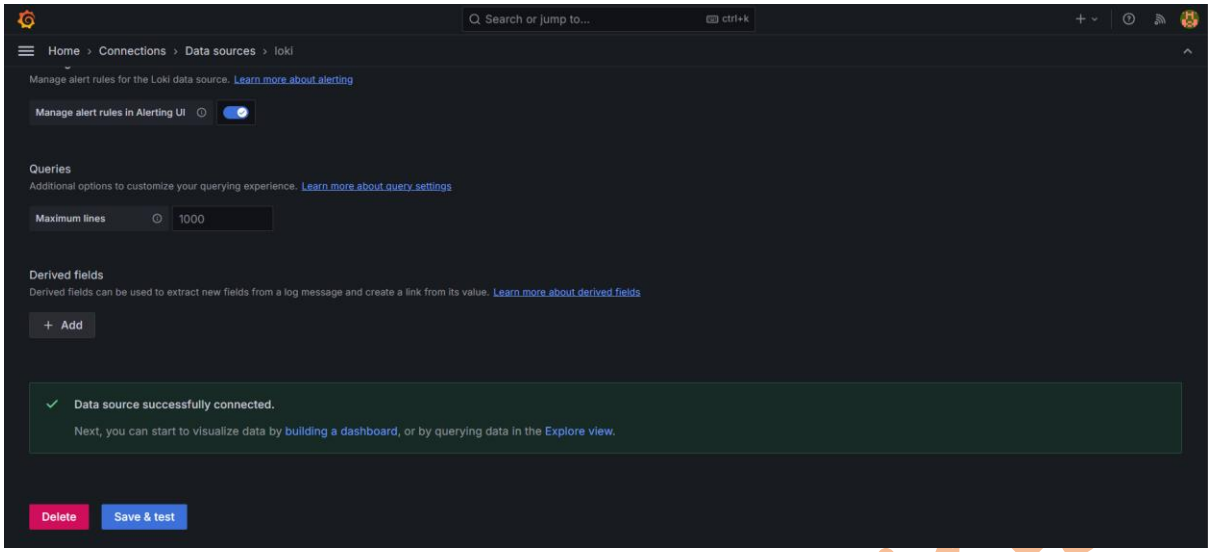
For Monitoring Tool I had used Prometheus and Grafana. To monitor SonarQube I had used SonarQube-Prometheus-Exporter which was installed using terraform at the path **/opt/sonarqube/extensions/plugins**. It was downloaded from the link https://github.com/dmeiners88/sonarqube-prometheus-exporter/releases/download/v1.0.0-SNAPSHOT-2018-07-04/sonar-prometheus-exporter-1.0.0-SNAPSHOT.jar. These steps had been covered in the terraform **user_data_sonarqube.sh**. It is basically a bootstrap script for SonarQube Server. For Monitoring Jenkins, you need to install the plugin Prometheus metrics and then restart Jenkins, these steps already been discussed at the starting. The configuration for prometheus had already been done in the terraform. I had taken sonarqube username and password as **admin** and **Admin123** respectively, you can choose as of your own choice and update the terraform script accordingly (Prometheus needs username and password to extract the metrics from SonarQube). I

had provided the terraform script with this GitHub Repository. Below Screenshot shows how I had integrated Prometheus and Loki with Grafana.

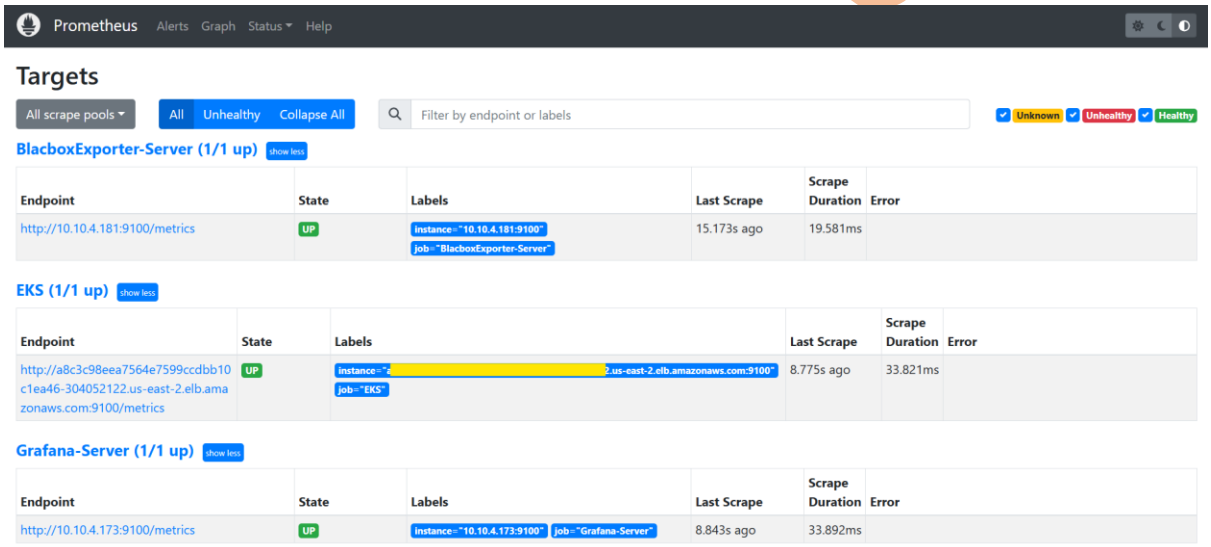




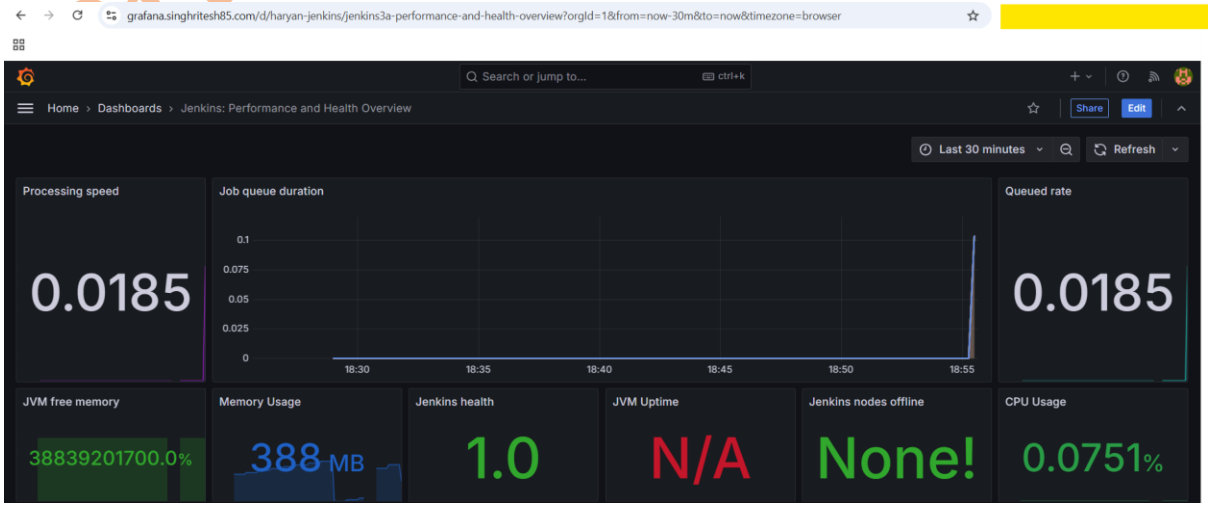




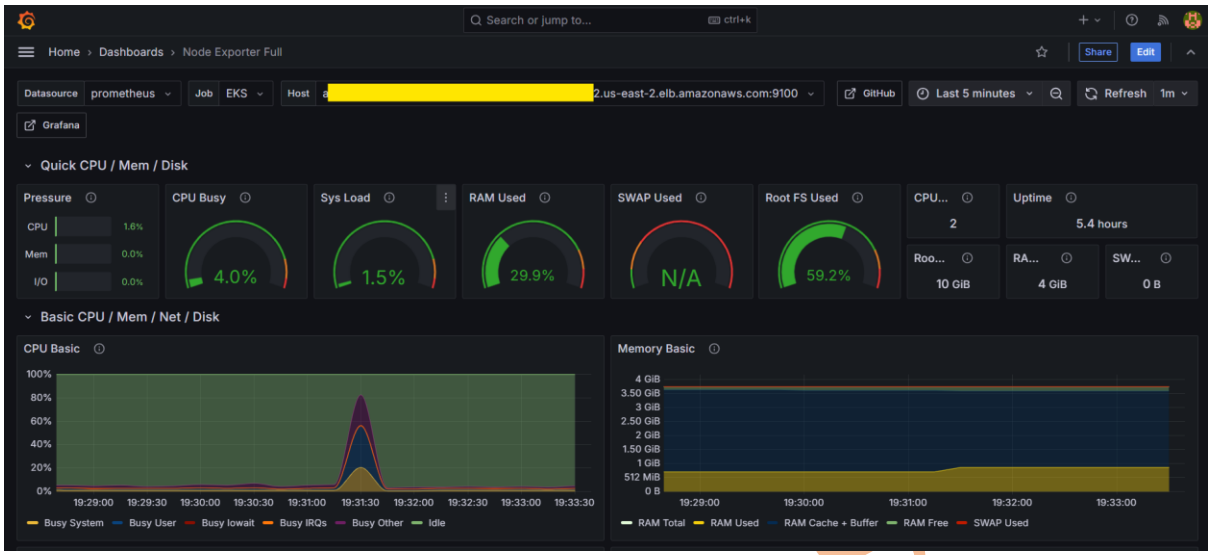
Finally, I checked the Prometheus Console and I was able to see all the Targets was UP.



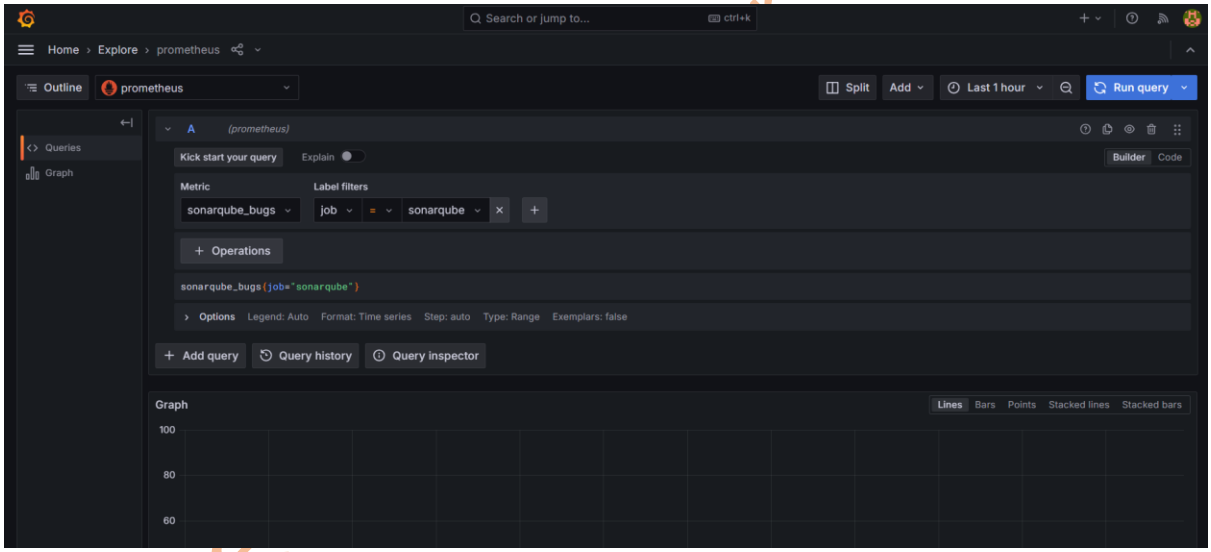
For Monitoring Jenkins Job using Prometheus I created the Grafana Dashboard using the Grafana ID **9964**.

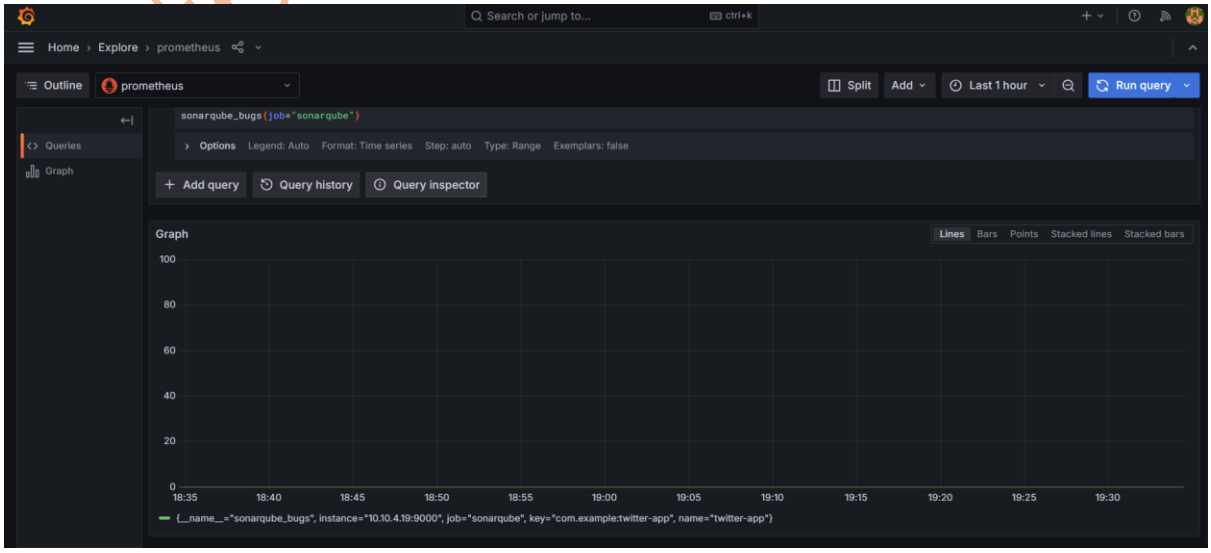


For Monitoring all the Servers and EKS Cluster health using the Node Exporter I used Grafana ID **1860**.

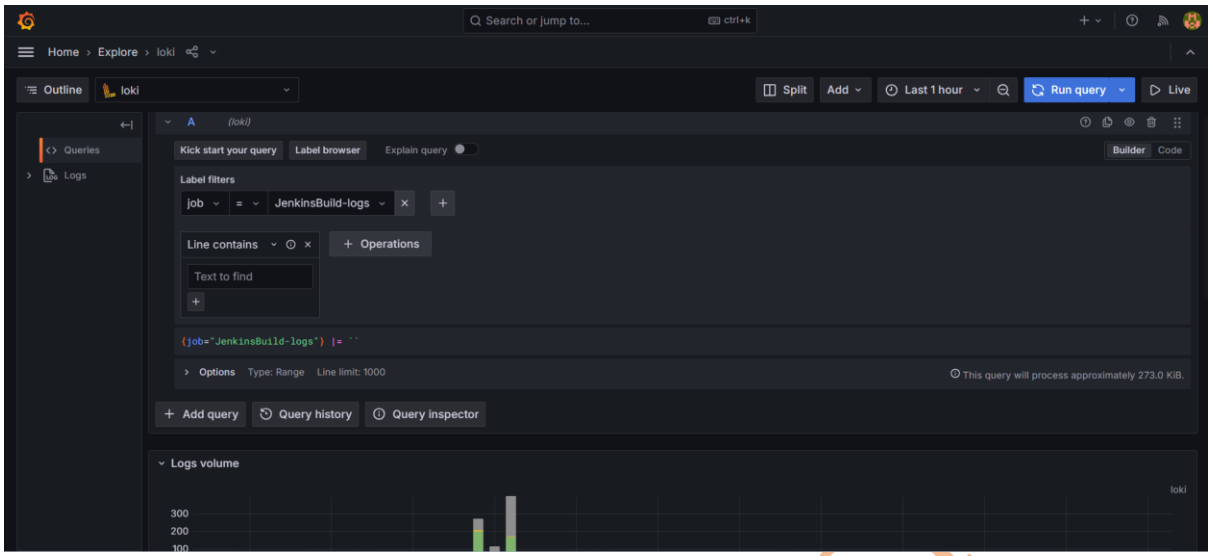


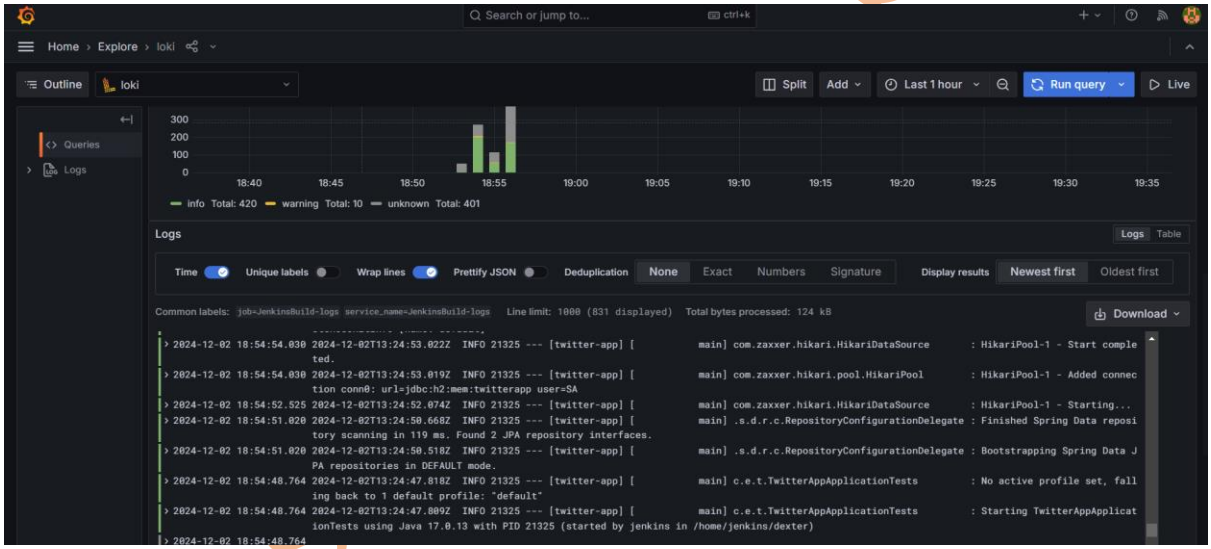
Grafana Metrics I started exploring as shown in the screenshot attached below.





Logs using Loki through Grafana I started exploring as shown in the screenshot attached below.





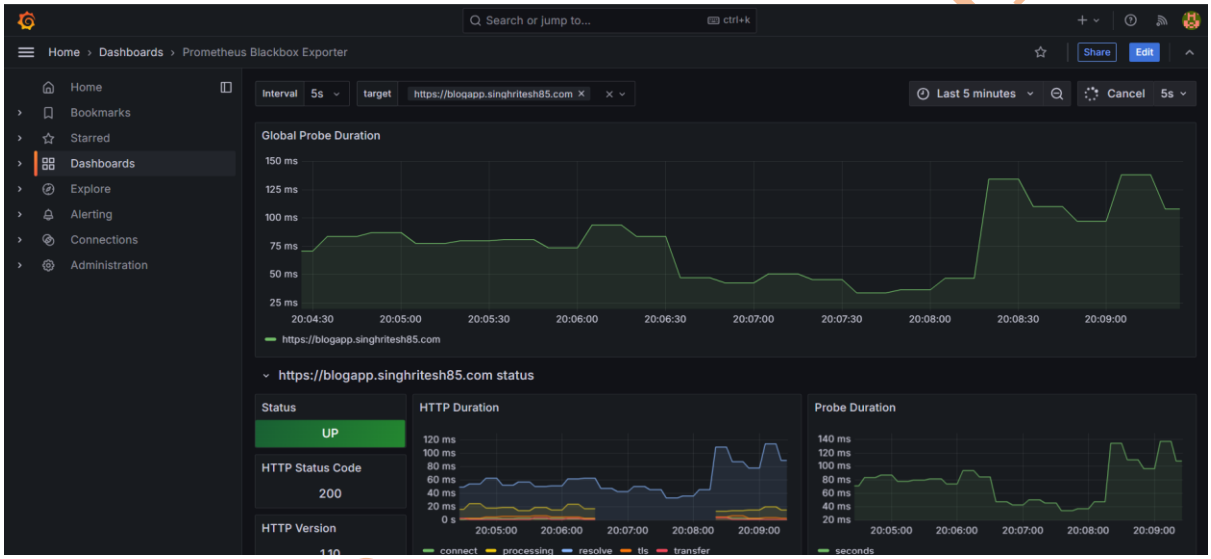
To achieve synthetic monitoring using Prometheus Blackbox Exporter I updated the **/etc/resolv.conf** file for Blackbox Exporter Server as shown in the screenshot attached below. I had used Google's Public DNS Server which is shown in the attached screenshot below.

```
[root@                ~]# cat /etc/resolv.conf
; generated by /usr/sbin/dhclient-script
search us-east-2.compute.internal
options timeout:2 attempts:5
nameserver 8.8.8.8    #10.10.0.2
```

Finally, I was able to perform the synthetics monitoring on the Blogging Application URL as shown in the screenshot attached below. Application URL **https://blogapp.singhritesh85.com** had been monitored using blackbox exporter.

I had installed Blackbox Exporter on a different server and not on the Prometheus Server. **The module name** is monitor_website.yml present of the blackbox exporter server at the path (/opt/blackbox_exporter_linux_amd64/monitor_website.yml). Prometheus blackbox operator is used for endpoint monitoring (Synthetic Monitoring) across the protocol http, https, TCP and ICMP. In this project I am monitoring the Application URL **https://blogapp.singhritesh85.com** with the help of Prometheus Blackbox-Exporter. Prometheus blackbox exporter will send the metrics to Prometheus. For this project Prometheus acts as a DataSource for Grafana and send metrics to Grafana which we can see with the help of Charts and Graphs.

To create the Grafana Dashboard for Application URL Monitoring using blackbox exporter I had used the Grafana ID **7587** and below is the created Dashboard.

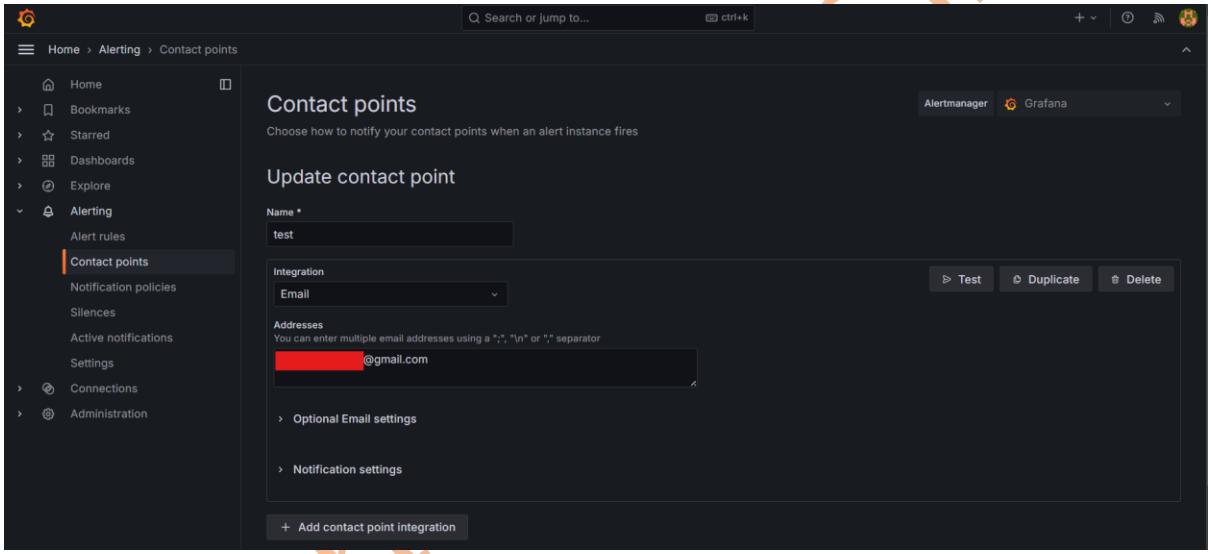


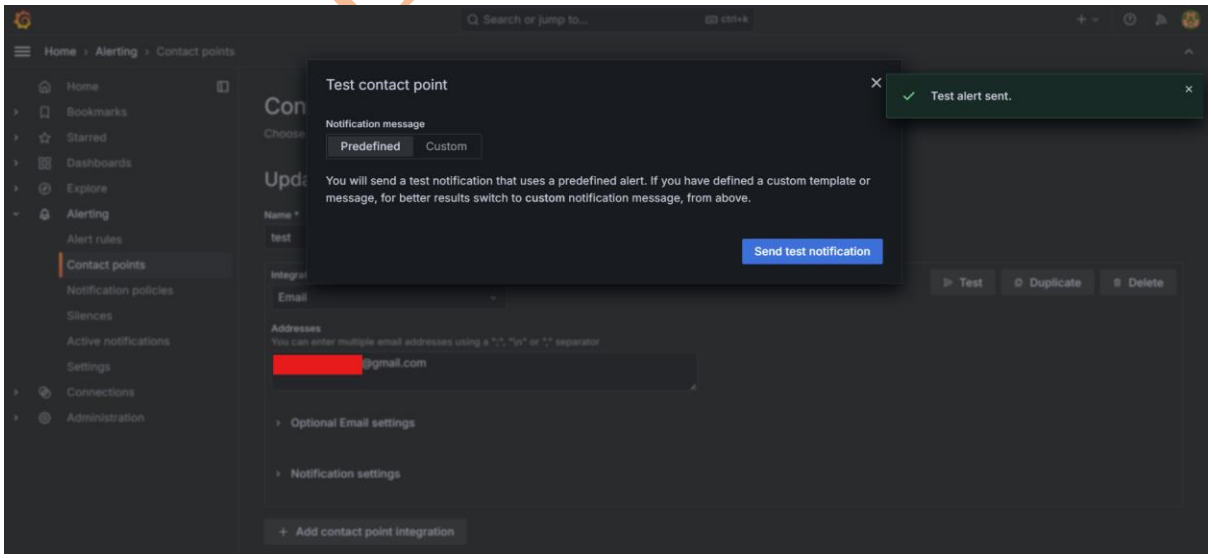
Configuration of Alerts in Grafana

To configure Alerts in Grafana, first I created **contact points** with the Email ID and changed smtp settings in the configuration file /etc/grafana/grafana.ini of Grafana as shown in the screenshot attached below.

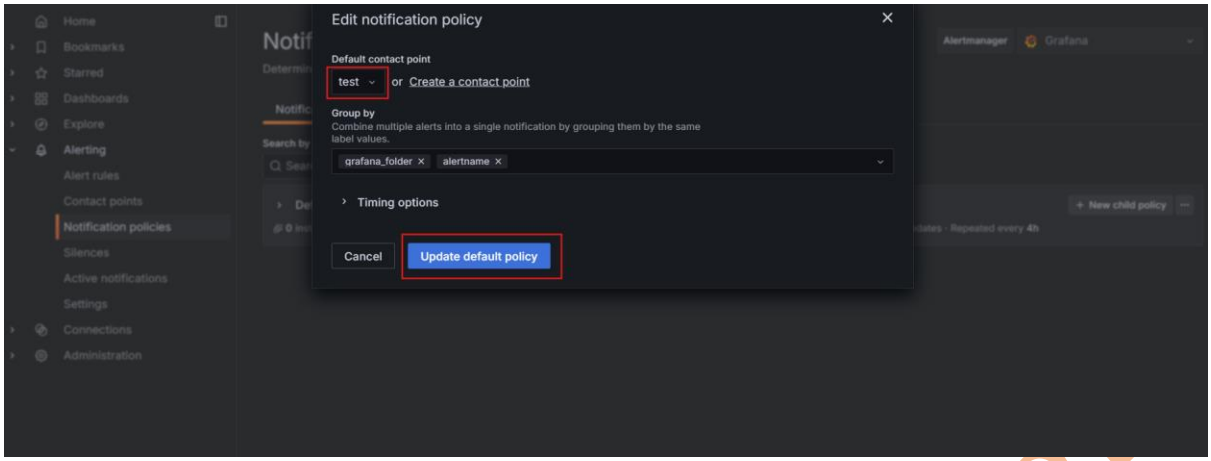
```
[root@[redacted] ~]# cat /etc/grafana/grafana.ini
```

```
################################### SMTP / Emailing ##########################
[smtp]
enabled = true
host = smtp.gmail.com:587
user = [redacted]@gmail.com
# If the password contains # or ; you have to wrap it with triple quotes. Ex """#password;"""
password = [redacted]
;cert_file =
;key_file =
skip_verify = true
from_address = [redacted]@gmail.com
from_name = Grafana
# EHLO identity in SMTP dialog (defaults to instance_name)
;ehlo_identity = dashboard.example.com
# SMTP startTLS policy (defaults to 'OpportunisticStartTLS')
;startTLS_policy = NoStartTLS
# Enable trace propagation in e-mail headers, using the 'traceparent', 'tracestate' and (optionally) 'baggage' fields (defaults to false)
;enable_tracing = false

[smtp.static_headers]
# Include custom static headers in all outgoing emails
;Foo-Header = bar
;Foo = bar

[emails]
;welcome_email_on_sign_up = false
;templates_pattern = emails/*.html, emails/*.txt
;content_types = text/html
```

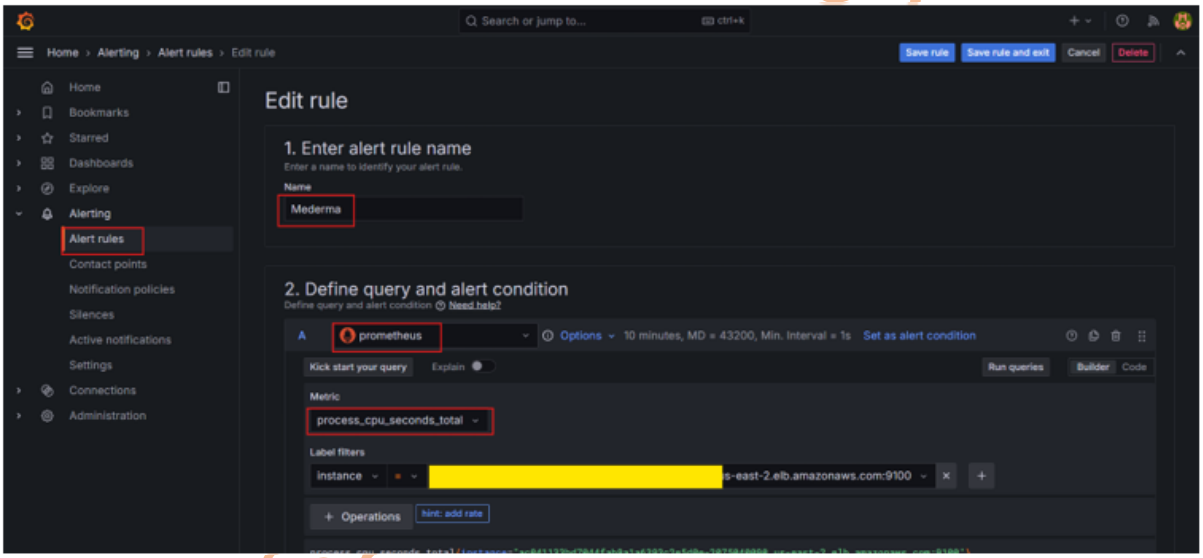


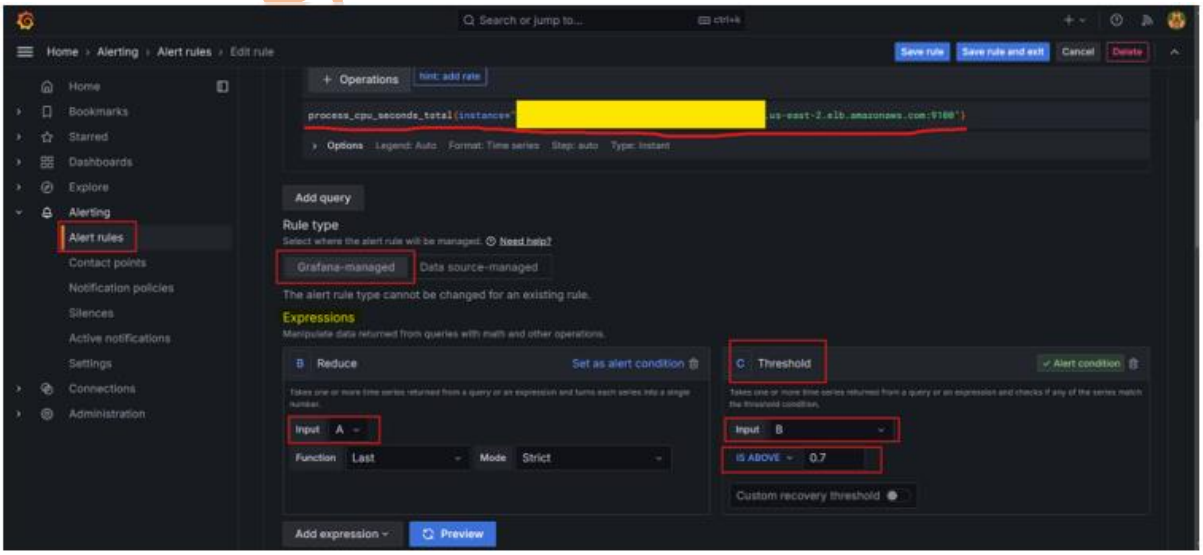


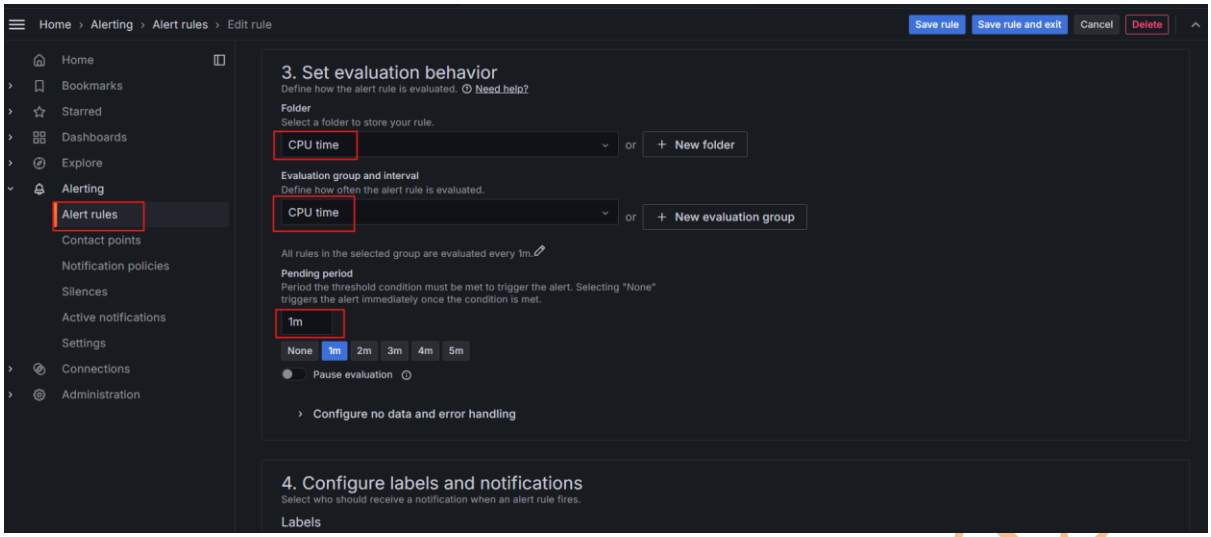
The Default **Notification Policy** had been changed as shown in the screenshot attached below.

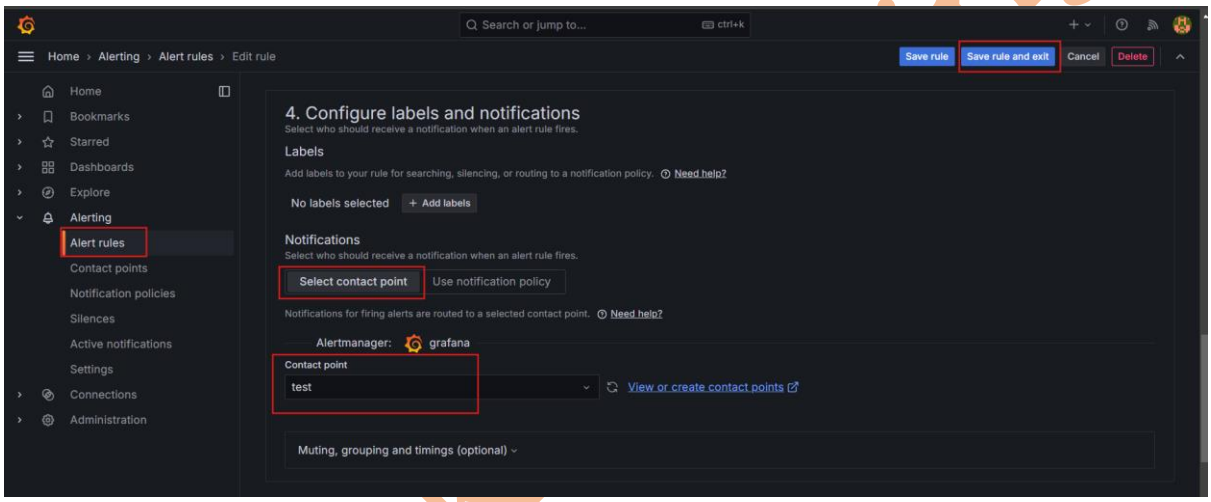


Configure **Alert Rule** as shown in the screenshot attached below.

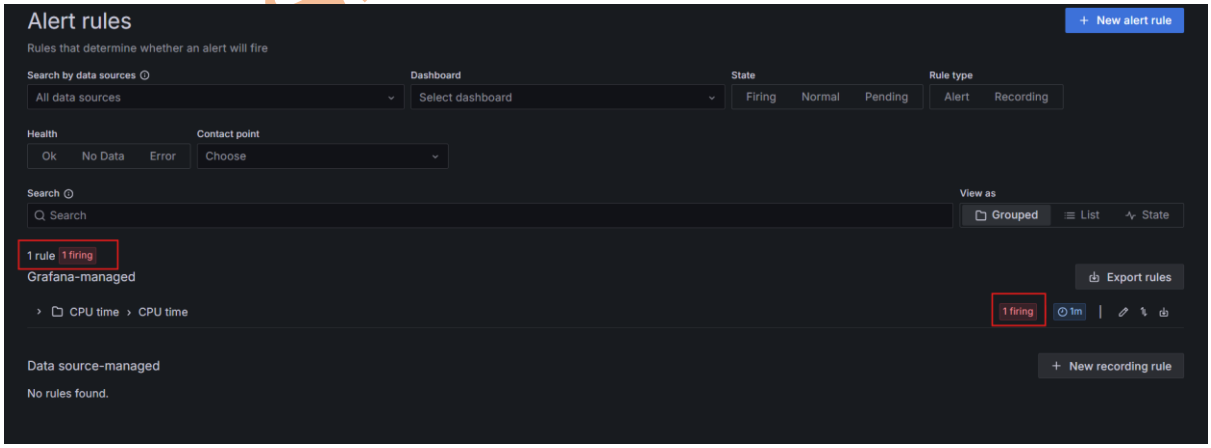








If the Alert Rule is in firing state after condition crosses the threshold condition, then Grafana console screenshot will be showing the same as shown in the screenshot attached below.



An Email will be sent to the Email ID as shown in the screenshot attached below.


Grafana
CPU time › Mederma
1 firing instances
Firing
Mederma
View alert
Values
A= B= C=
Labels
alertname Mederma
grafana_folder CPU time
instance us-east-2.elb.amazonaws.com:9100
job EKS