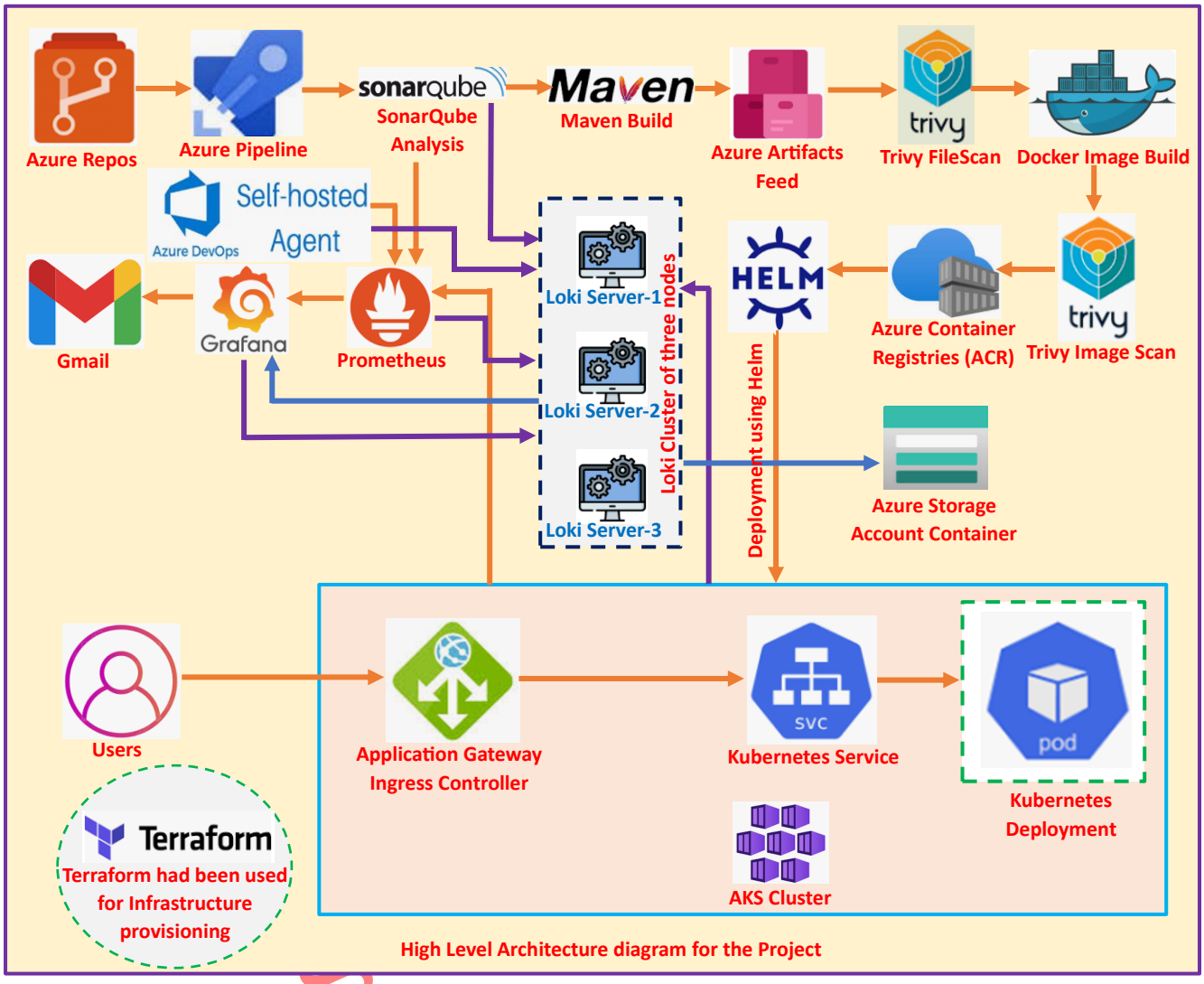
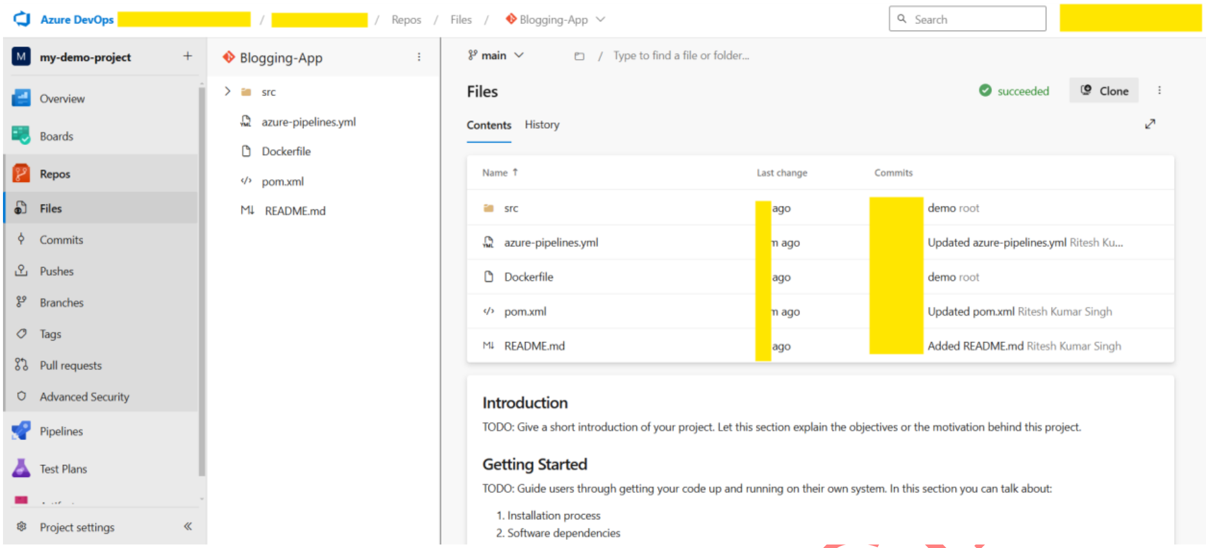
DevOps Project Blogging Application Monitoring using Prometheus and Grafana Log Aggregation using Loki Promtail and Grafana



This DevOps project aims to create the infrastructure using Terraform and to establish the CI/CD pipeline using Azure DevOps. For Monitoring Prometheus and Grafana and for Log Aggregation Loki, promtail and Grafana had been used. The Source Code was present in the Azure Repos and Azure DevOps Pipeline had been used as the CI/CD Tool. SonarQube and Azure Artifacts Feed was used for code Analysis and to keep the artifacts for the project respectively. Maven was used as the Build Tool for the project. Trivy was used for file scan and Docker Image Scan as shown in the screenshot attached above. Finally, Application Pods had been created using the Docker Image which was kept in the Azure Container Registries (ACR). For Monitoring Prometheus and Grafana and for Log aggregation Loki, Promtail and Grafana had been used. Node exporter was installed on of each Azure VMs and on the AKS Cluster which extracted the metrics from Azure VMs and AKS Cluster and forwarded to prometheus which finally send them to Grafana where we had visualised with the help of Graphs. For Log Aggregation promtail had been installed on all the Azure VMs and AKS Cluster which extracted the Logs and send to Loki and finally, to Grafana as explained in the High-Level Architecture Diagram drawn above. For this project Helm was used for Deployment to AKS Cluster.

The Source Code was kept in the Azure Repos as shown in the screenshot attached below.



For Azure DevOps Pipeline I had used Self-hosted-Agent and followed the below procedure to install it.

```
[root@devopsagent-vm ~]# cd /opt && mkdir myagent && cd myagent
```

```
[root@devopsagent-vm myagent]# wget https://vstsagentpackage.azureedge.net/agent/[redacted]/vsts-agent-linux-x64-[redacted].tar.gz
```

```
[root@devopsagent-vm myagent]# tar -xvf vsts-agent-linux-x64-[redacted].tar.gz
```

```
[root@devopsagent-vm myagent]# rm -f vsts-agent-linux-x64-[redacted].tar.gz
```

```
[root@devopsagent-vm myagent]# ./bin/installdependencies.sh
```

```
[demo@devopsagent-vm myagent]$ ./config.sh
```

```
        agent v[redacted]                              [redacted]
```

```
[demo@devopsagent-vm myagent]$ ./env.sh
[demo@devopsagent-vm myagent]$
[demo@devopsagent-vm myagent]$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/opt/sonar-scanner/bin:/opt/apache-maven/bin:/opt/node-v16.0.0/bin:/opt/dependency-check/bin:/usr/local/bin
```

```
[demo@devopsagent-vm myagent]$ sudo ./svc.sh install
```

```
[demo@devopsagent-vm myagent]$ sudo ./svc.sh start
```

Kubernetes Secrets had been created as shown in the screenshot attached below to provide privilege to receive the Docker Image from the Azure Container Registries (ACR).

```
[root@[redacted] ~]# kubectl create secret docker-registry bloggingapp-auth --docker-server=https://blogappcontainer24registry.azurecr.io --docker-username=blogappcontainer24registry --docker-password=[redacted] -n blogapp
```

Kubernetes Secrets had been created as shown in the screenshot attached below for TLS of the kubernetes ingress.

```
[root@[redacted] ~]# kubectl create secret tls ingress-secret --key mykey.key --cert STAR_singhritesh85_com.crt -n blogapp
secret/ingress-secret created
```

kubectl create secret docker-registry bloggingapp-auth --docker-server=https://blogappcontainer24registry.azurecr.io --docker-username=blogappcontainer24registry --docker-password=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX -n blogapp

kubectl create secret tls ingress-secret --key mykey.key --cert STAR_singhritesh85_com.crt -n blogapp

Kubernetes Ingress had been created using the Ingress Rule as shown in the screenshot attached below.

```
[root@                ~]# kubectl get ing -n blogapp
NAME              CLASS                       HOSTS                       ADDRESS          PORTS     AGE
blogapp-ingress   azure-application-gateway   blogapp.singhritesh85.com   48.              80, 443   10s
[root@                ~]# cat ingress-rule.yaml
# kubectl create secret tls ingress-secret --key mykey.key --cert STAR_singhritesh85_com.crt -n blogapp
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: blogapp-ingress
  namespace: blogapp
  annotations:
    appgw.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  ingressClassName: azure-application-gateway
  tls:
  - secretName: ingress-secret
  rules:
  - host: blogapp.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: blogapp-folo
            port:
              number: 80
```

cat ingress-rule.yaml

```
# kubectl create secret tls ingress-secret --key mykey.key --cert STAR_singhritesh85_com.crt -n blogapp
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: blogapp-ingress
  namespace: blogapp
  annotations:
    appgw.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  ingressClassName: azure-application-gateway
  tls:
  - secretName: ingress-secret
  rules:
  - host: blogapp.singhritesh85.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: blogapp-folo
            port:
              number: 80
```

Azure DevOps Pipeline had been created using azure-pipelines.yaml as provided below.

azure-pipelines.yaml

```
trigger:
- main

pool:
  name: demo
  demands:
  - agent.name -equals demo

variables:
  imagePullSecret: 'bloggingapp-auth'

stages:
- stage: "Build"
  displayName: Build
  jobs:
  - job: "Build"
    displayName: Build
    steps:
    - task: SonarQubePrepare@6
      inputs:
        SonarQube: 'SonarQube'
        scannerMode: 'Other'
        extraProperties: |
          # Additional properties that will be passed to the scanner,
          # Put one key=value per line, example:
          # sonar.exclusions=**/*.bin
          sonar.projectName=bloggingapp
          sonar.projectKey=bloggingapp
          sonar.qualitygate.wait=true
    - task: SonarQubePublish@6
      inputs:
```

```
      pollingTimeoutSec: '300'
    - task: sonar-buildbreaker@8
      inputs:
        SonarQube: 'SonarQube'
    - task: MavenAuthenticate@0
      inputs:
        artifactsFeeds: 'Maven'
        mavenServiceConnections: 'Maven'
    - task: Maven@4
      inputs:
        mavenPomFile: 'pom.xml'
        goals: 'deploy sonar:sonar'
        publishJUnitResults: false
        javaHomeOption: 'JDKVersion'
        mavenVersionOption: 'Default'
        mavenAuthenticateFeed: false
        effectivePomSkip: false
        sonarQubeRunAnalysis: false
    - task: CmdLine@2
      inputs:
        script: 'trivy fs . > /home/demo/trivy-filescan.txt'
- stage: DockerImageBuild
  displayName: DockerImageBuild
  dependsOn: "Build"
  jobs:
  - job: DockerImageBuild
    displayName: DockerImageBuild
    steps:
    - checkout: none
    - task: CmdLine@2
      inputs:
```

```
        script: |
          docker system prune -f --all
          docker build -t demoimage:1.05 .
          docker tag demoimage:1.05 blogappcontainer24registry.azurecr.io/samplewebapp:$(Build.Buil
dId)
          trivy image --exit-code 0 --
severity MEDIUM,HIGH blogappcontainer24registry.azurecr.io/samplewebapp:$(Build.BuildId)
          #trivy image --exit-code 1 --
severity CRITICAL blogappcontainer24registry.azurecr.io/samplewebapp:$(Build.BuildId)
      - task: Docker@2
        inputs:
          containerRegistry: 'Docker-Registry'
          repository: 'samplewebapp'
          command: 'buildAndPush'
          Dockerfile: '**/Dockerfile'
  - stage: KubernetesDeployment
    displayName: KubernetesDeployment
    dependsOn: DockerImageBuild
    jobs:
    - deployment: KubernetesDeployment
      displayName: KubernetesDeployment
      environment: "dev"
      strategy:
        runOnce:
          deploy:
            steps:
            - checkout: none
            - task: HelmDeploy@1
              inputs:
                connectionType: 'Azure Resource Manager'
                azureSubscription: 'Azure DevOps Service Connection'
                azureResourceGroup: 'blogapp-rg'
```

kubernetesCluster: 'blogapp-cluster'

namespace: 'blogapp'

command: 'upgrade'

chartType: 'FilePath'

chartPath: '/home/demo/helm-repo-for-ArgoCD/folo'

releaseName: 'blogapp'

overrideValues: 'imagePullSecrets[0].name=bloggingapp-auth,image.repository=blogappcontainer24registry.azurecr.io/samplewebapp,image.tag=$(Build.BuildId),replicaCount=1,service.type=ClusterIP,service.port=80'

I had provided restricted access to the deployment user **demo** in the AKS Cluster using service account, Role and Role Binding as shown in the screenshot attached below. The deployment user **demo** had all the access in the namespace blogapp but did not have entire access over the AKS Cluster. That means for the deployment user demo, access was restricted to the namespace blogapp.

```
[root@                ~]# cat sa-role-rolebinding.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: demo
  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: demo

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

cat sa-role-rolebinding.yaml

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: demo
  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: demo
```

Finally, Kubernetes Secrets with the name of **mysecret** had been created and its token was used in the kubeconfig file as shown in the screenshot attached below.

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


[root@                ~]# kubectl apply -f secret.yaml
secret/mysecret created
[root@                ~]# kubectl get secrets -n blogapp
NAME       TYPE                                  DATA   AGE
mysecret   kubernetes.io/service-account-token   3      26s
```

cat secret.yaml

---

apiVersion: v1

kind: Secret

type: kubernetes.io/service-account-token

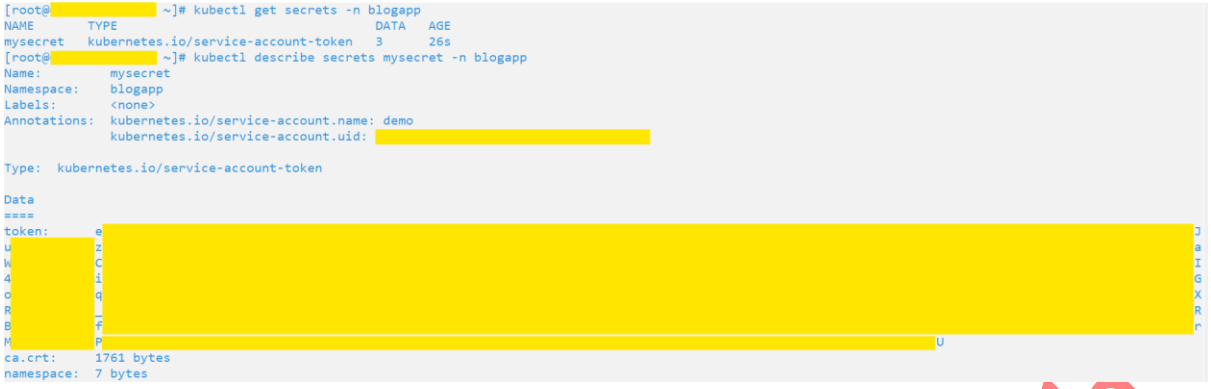
metadata:

name: mysecret

namespace: blogapp

annotations:

kubernetes.io/service-account.name: demo



kubectl describe secrets mysecret -n blogapp

Name: mysecret

Namespace: blogapp

Labels: <none>

Annotations: kubernetes.io/service-account.name: demo

kubernetes.io/service-account.uid: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

Type: kubernetes.io/service-account-token

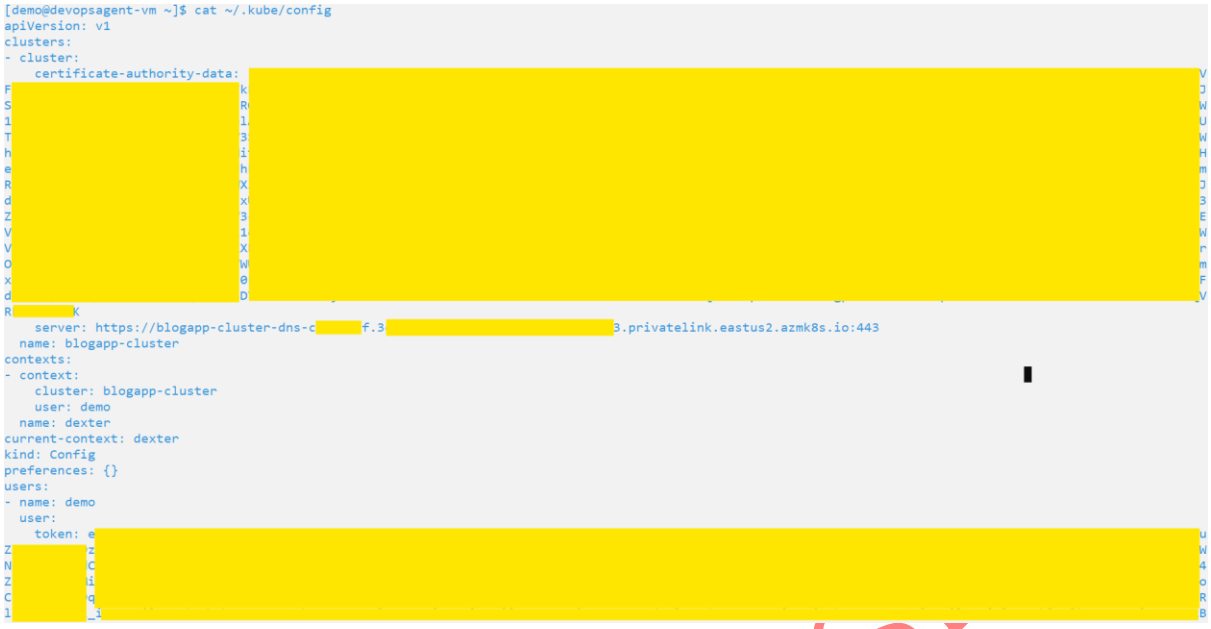
Data

====

token:
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ca.crt: 1761 bytes

namespace: 7 bytes

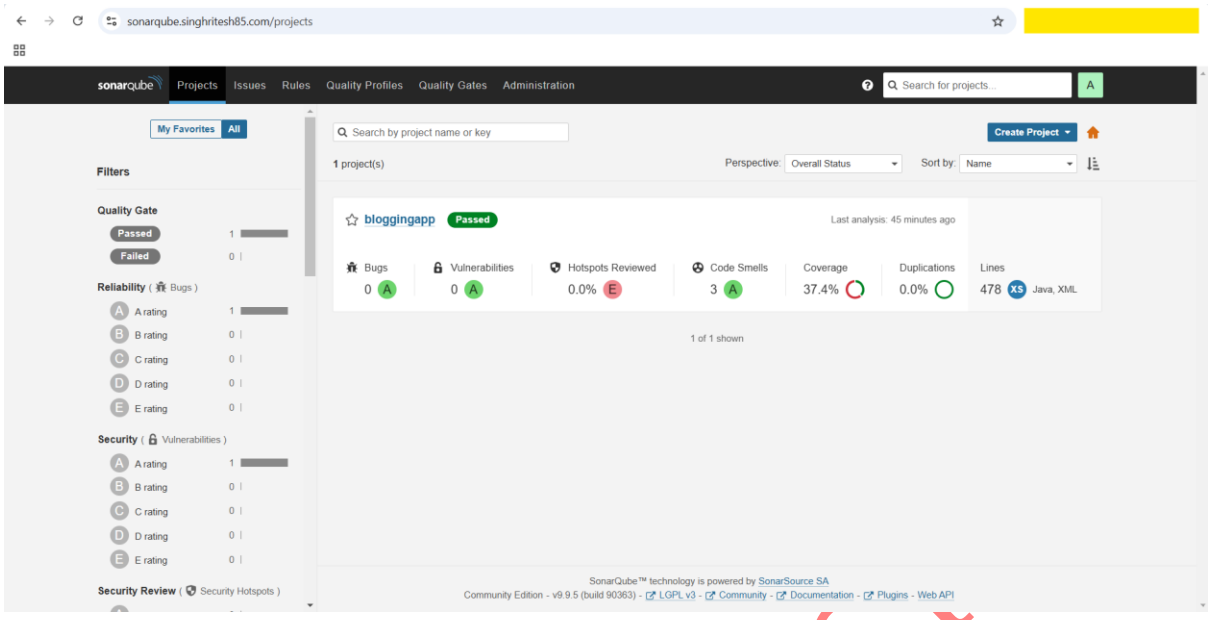


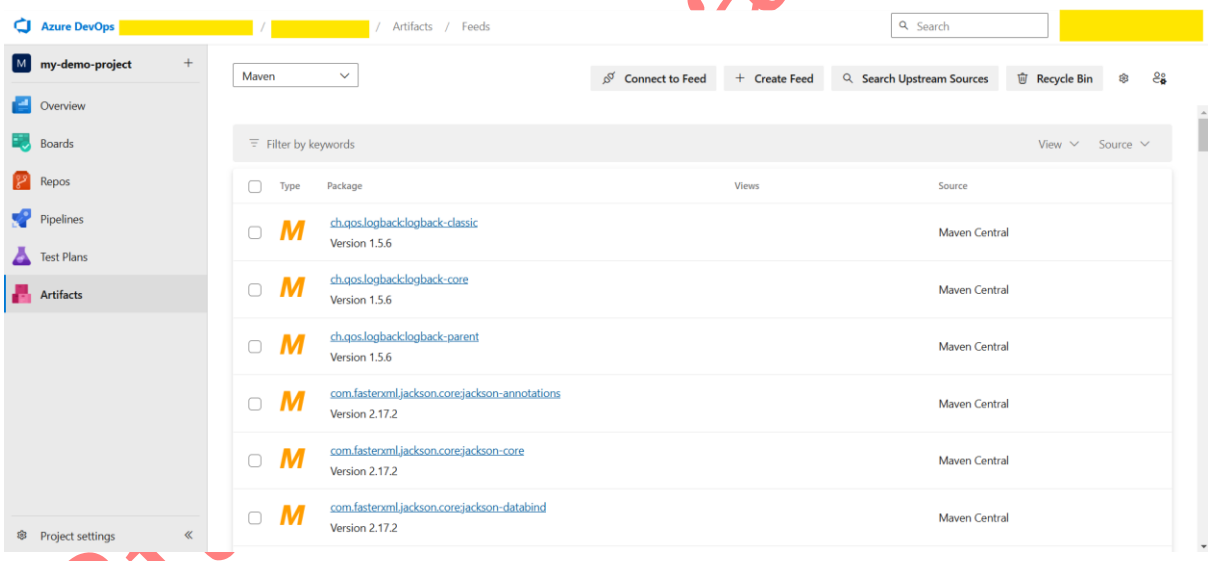
Below kubeconfig file was shared with the deployment user demo.

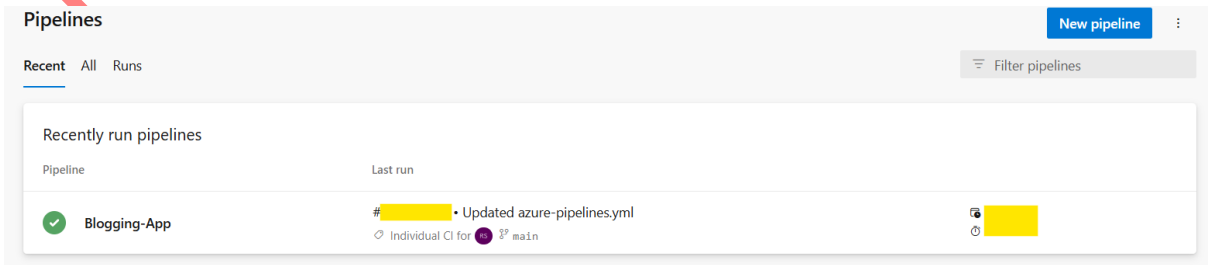
cat ~/.kube/config

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    server: https://blogapp-cluster-dns-
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.privatelink.eastus2.azmk8s.io:443
  name: blogapp-cluster
contexts:
- context:
    cluster: blogapp-cluster
    user: demo
  name: dexter
current-context: dexter
kind: Config
preferences: {}
users:
- name: demo
  user:
    token:
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Below screenshot shows SonarQube, Azure Artifacts Feed and Azure DevOps Pipeline after its successful execution.



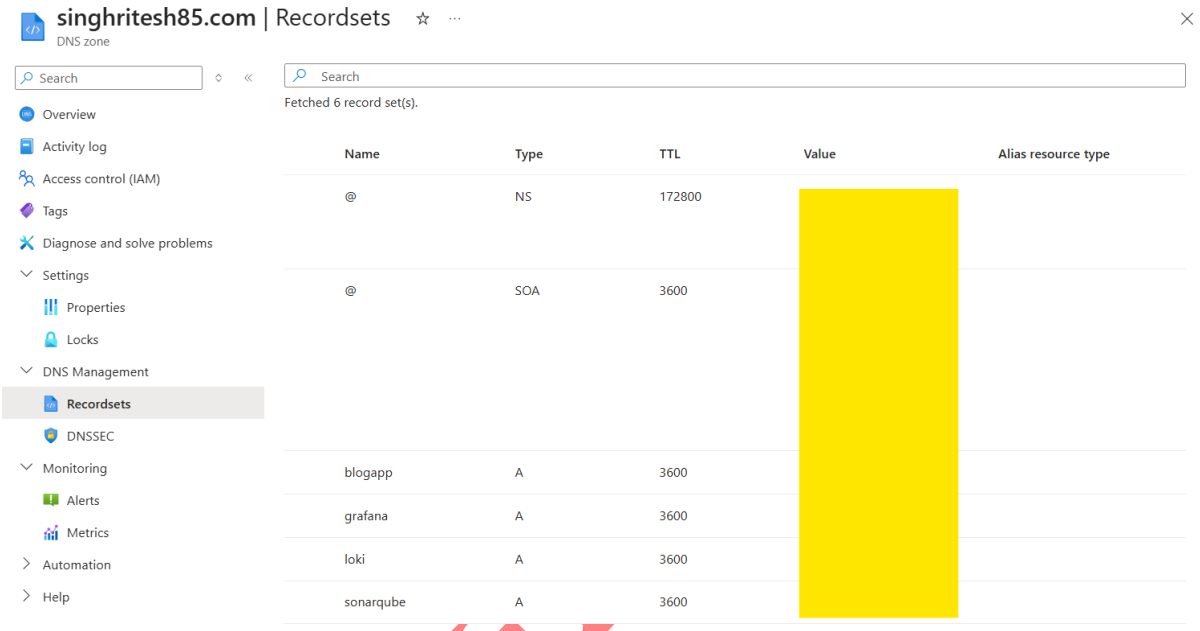




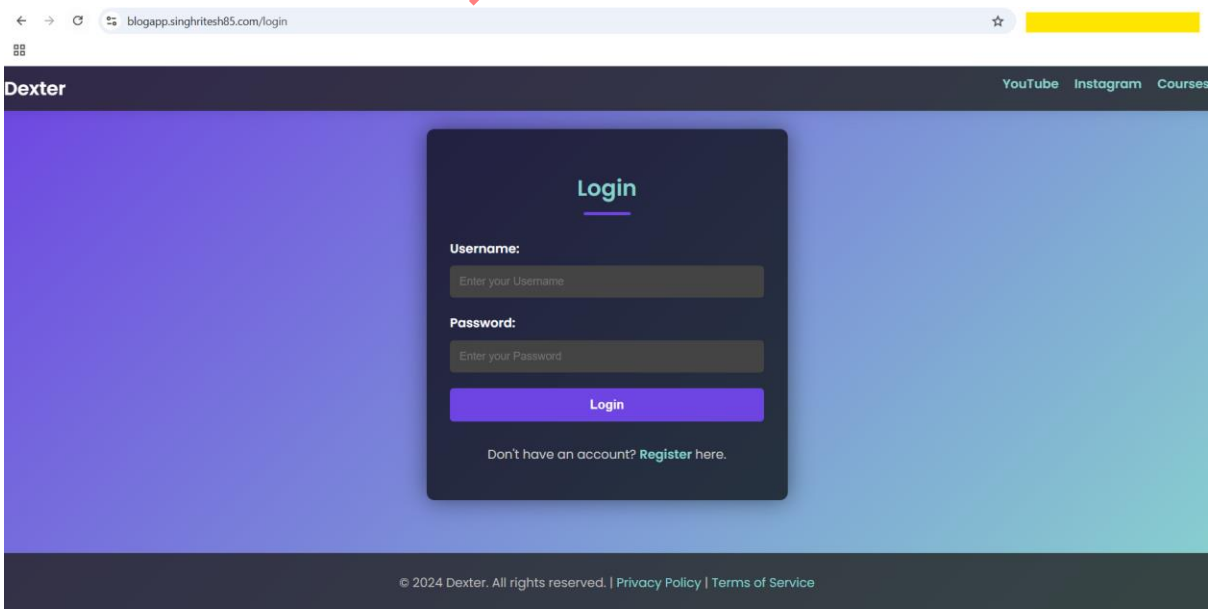
The Kubernetes Pod, Kubernetes Service and Kubernetes Deployment had been created after successful execution of Azure DevOps Pipeline as shown in the screenshot attached below.

```
[demo@devopsagent-vm ~]$ kubectl get all -n blogapp
NAME                              READY   STATUS    RESTARTS   AGE
pod/blogapp-folo-[redacted]       1/1     Running   0          102m

NAME                  TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/blogapp-folo  ClusterIP   10.[redacted]  <none>        80/TCP    104m

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/blogapp-folo      1/1     1            1           104m

NAME                                         DESIRED   CURRENT   READY   AGE
replicaset.apps/blogapp-folo-[redacted]      1         1         1       104m
[demo@devopsagent-vm ~]$ kubectl get nodes
Error from server (Forbidden): nodes is forbidden: User "system:serviceaccount:blogapp:demo" cannot list resource "nodes" in API group "" at the cluster scope
```

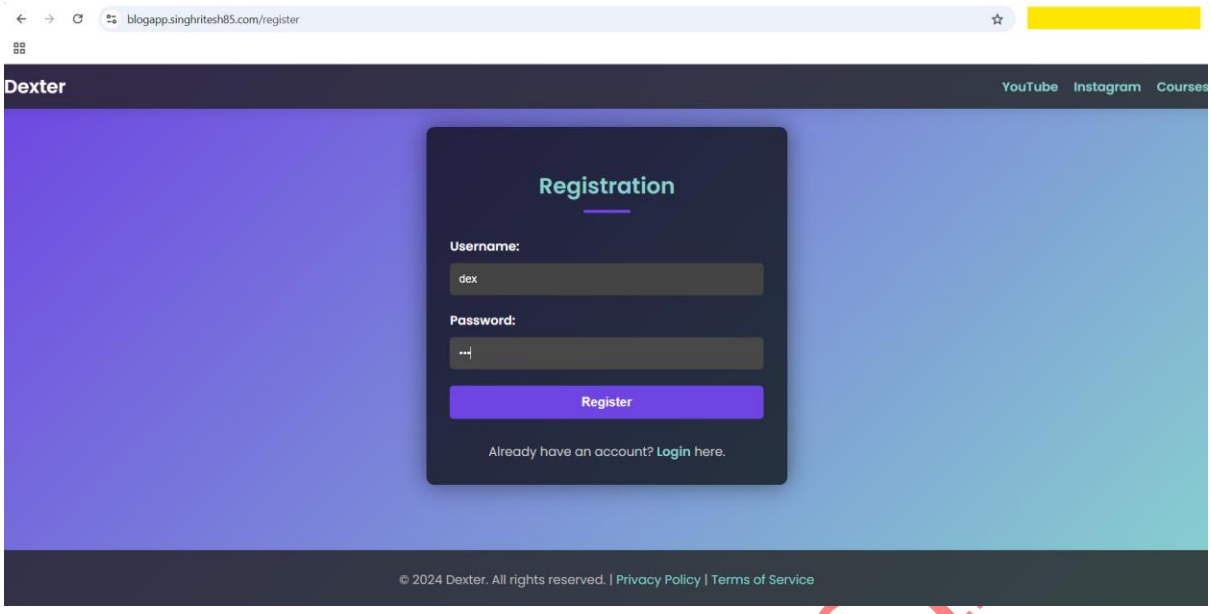
The screenshot for Azure DNS Zone and Record Sets had been shown in the screenshot attached below.



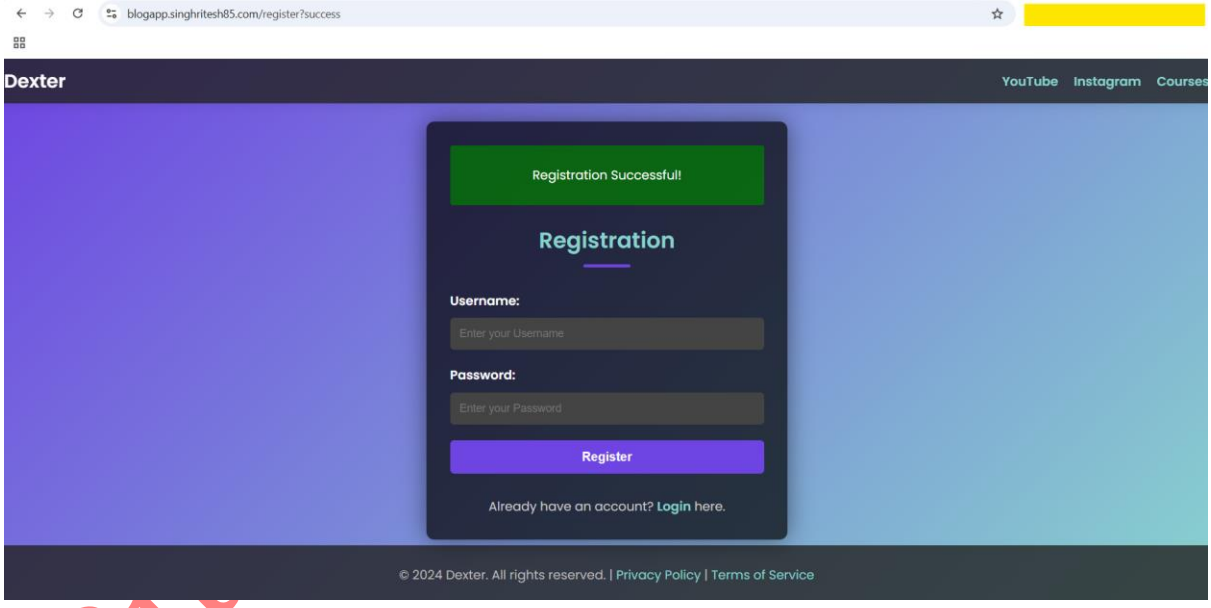
Finally accessed the Application using the URL as shown in the screenshot attached below.



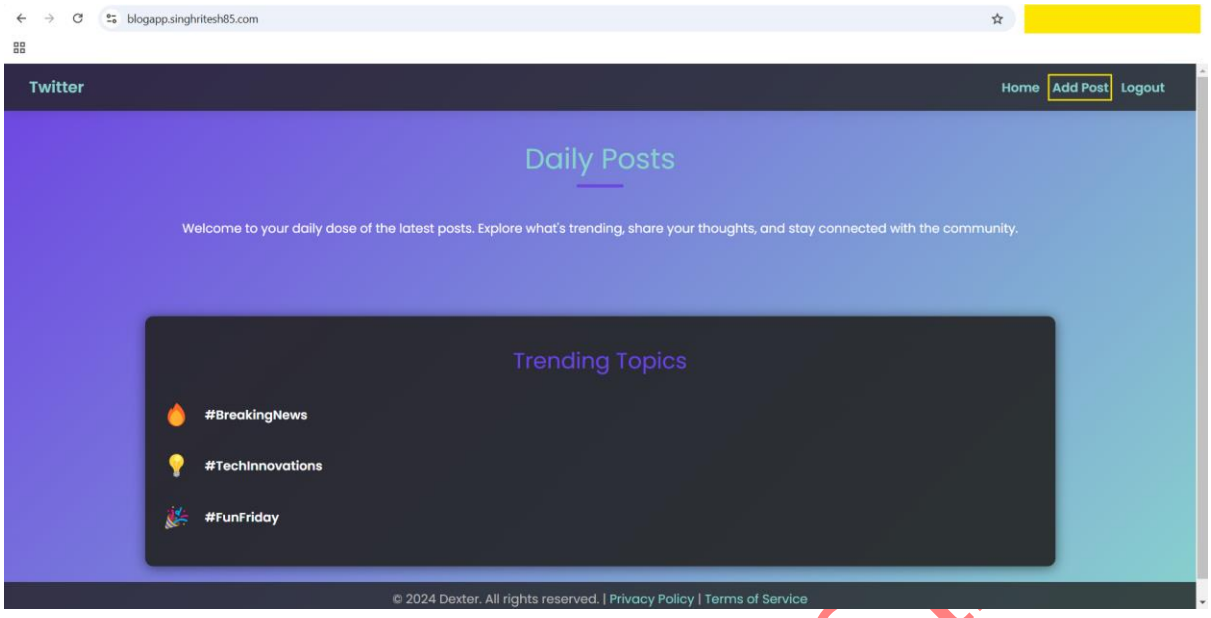
I did the Registration of New User as shown in the screenshot attached below.

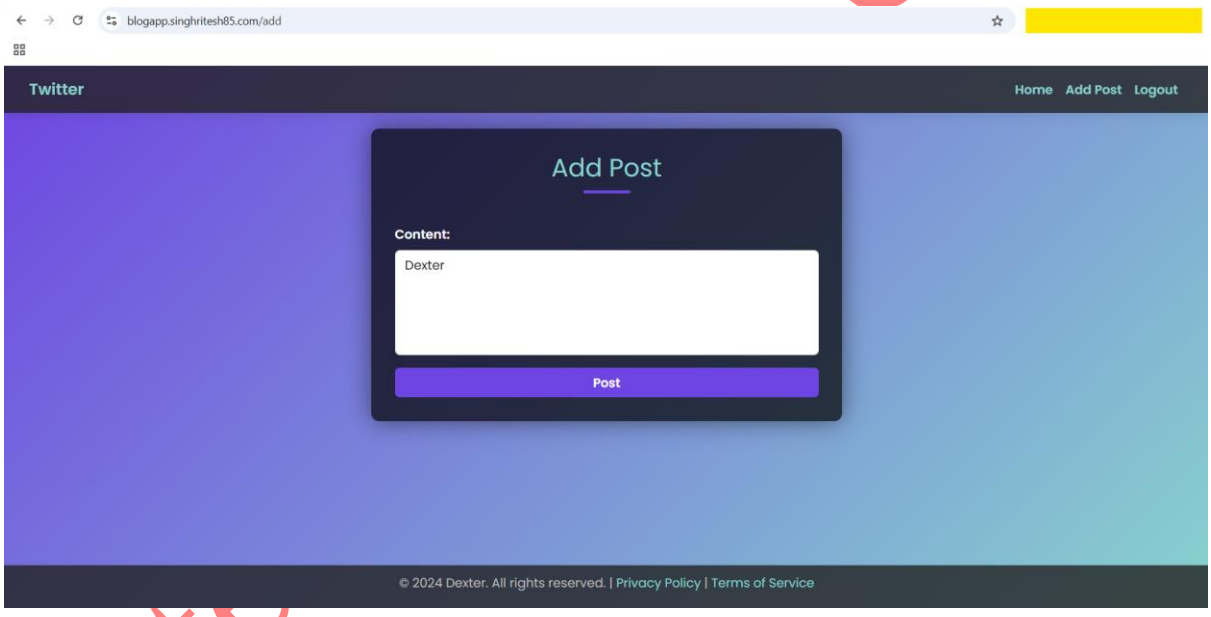


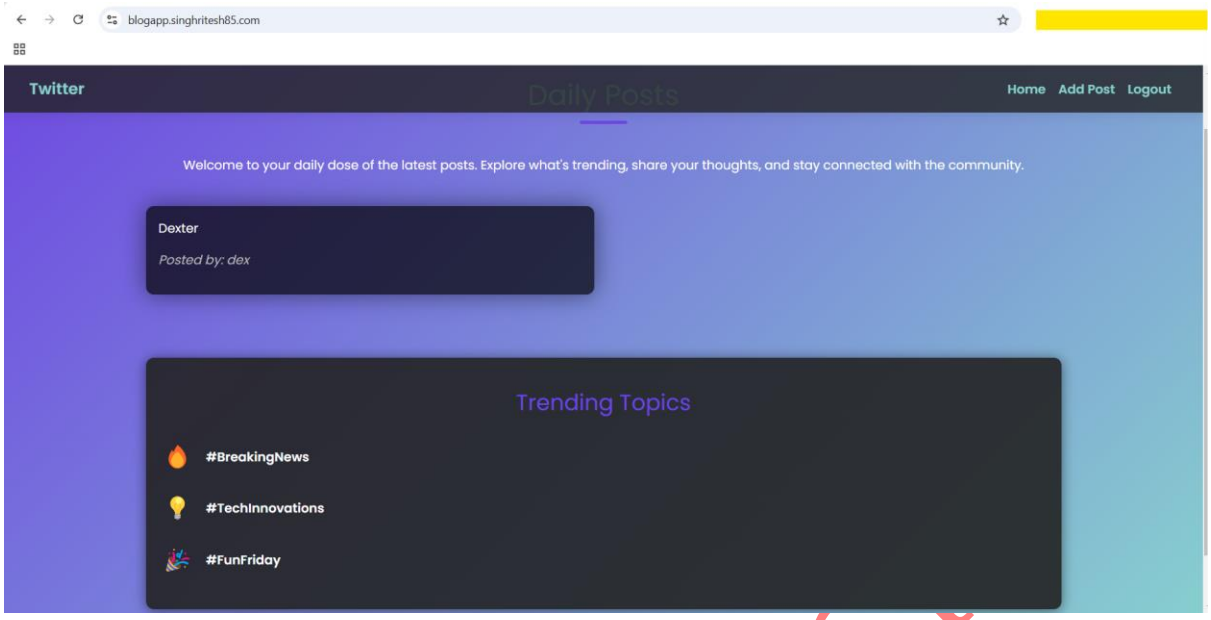
Below screenshot showed the successful registration of the User.

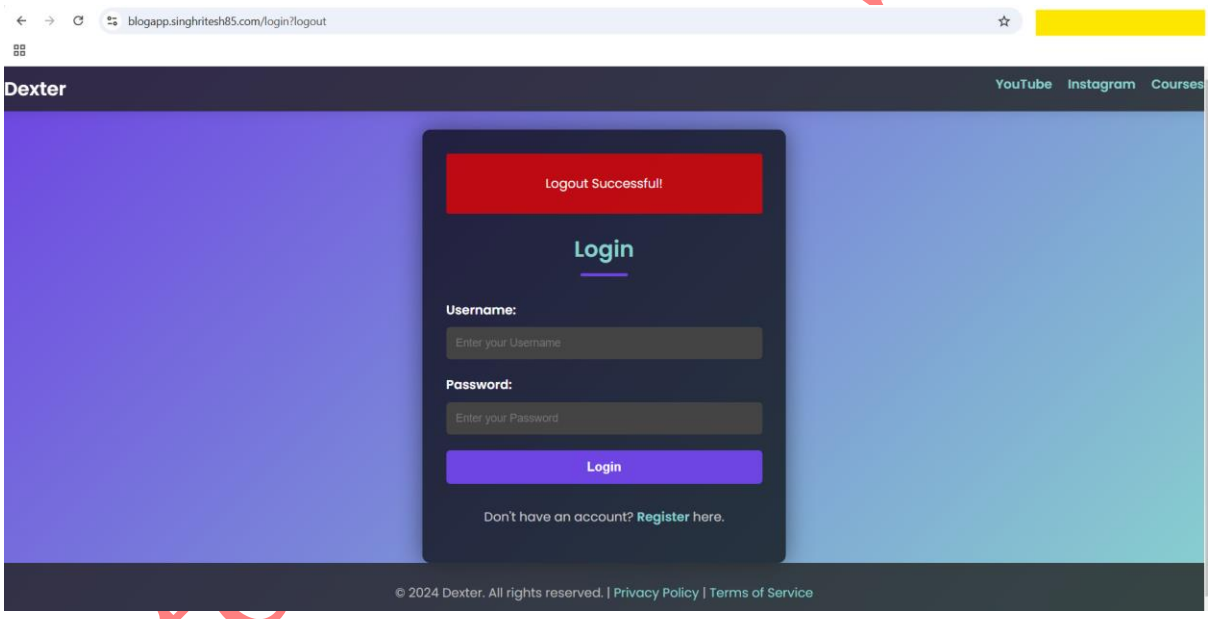


Newly created user wrote and read the blogs as shown in the screenshot attached below.


blogapp.singhritesh85.com
Twitter
Home
Add Post
Logout
Daily Posts
Welcome to your daily dose of the latest posts. Explore what's trending, share your thoughts, and stay connected with the community.
Trending Topics
#BreakingNews
#TechInnovations
#FunFriday
© 2024 Dexter. All rights reserved. | Privacy Policy | Terms of Service

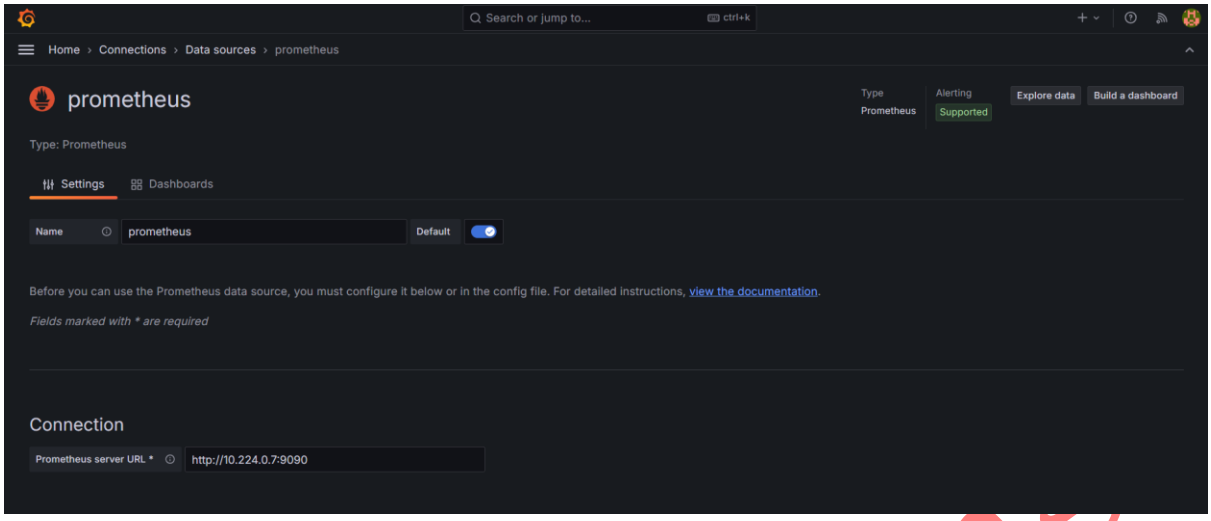

blogapp.singhritesh85.com/add
Twitter
Home
Add Post
Logout
Add Post
Content:
Dexter
Post
© 2024 Dexter. All rights reserved. | Privacy Policy | Terms of Service

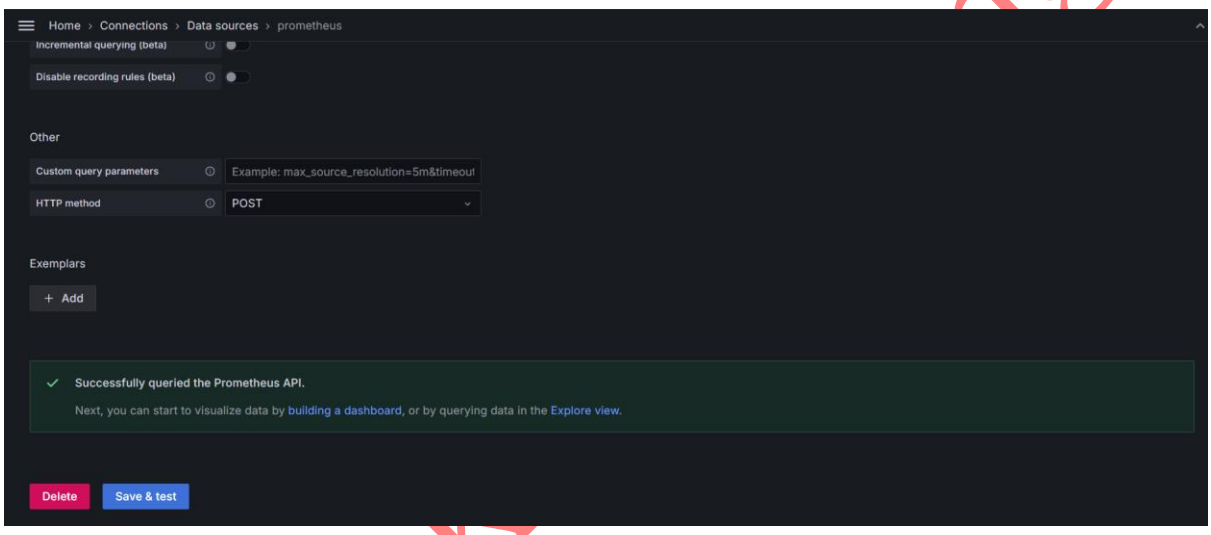




Monitor SonarQube using Prometheus and Grafana

For Monitoring, Prometheus acted as a Source as shown in the screenshot attached below.



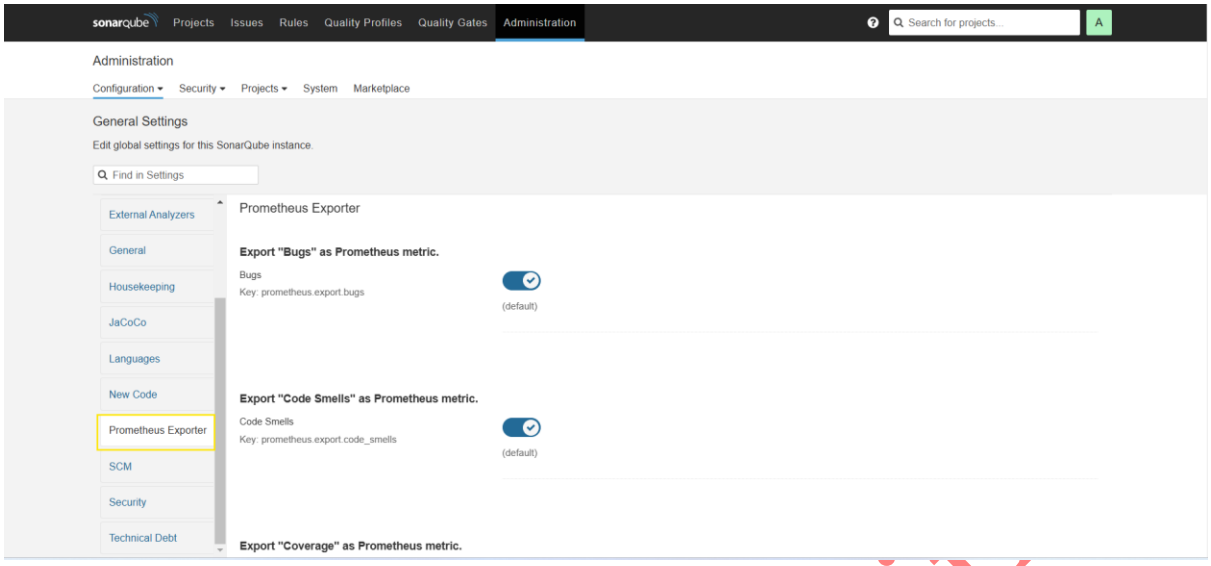


To monitor sonarqube using prometheus and grafana follow the steps as written below.

1. Download the sonarqube-prometheus-exporter at the path **/opt/sonarqube/extensions/plugins** as shown in the screenshot attached below.

```
[root@SonarQube-Server ~]# cd /opt/sonarqube/extensions/plugins/
[root@SonarQube-Server plugins]# ls
README.txt
[root@SonarQube-Server plugins]# wget https://github.com/dmeiners88/sonarqube-prometheus-exporter/releases/download/v1.0.0-SNAPSHOT-2018-07-04/sonar-prometheus-exporter-1.0.0-SNAPSHOT.jar
```

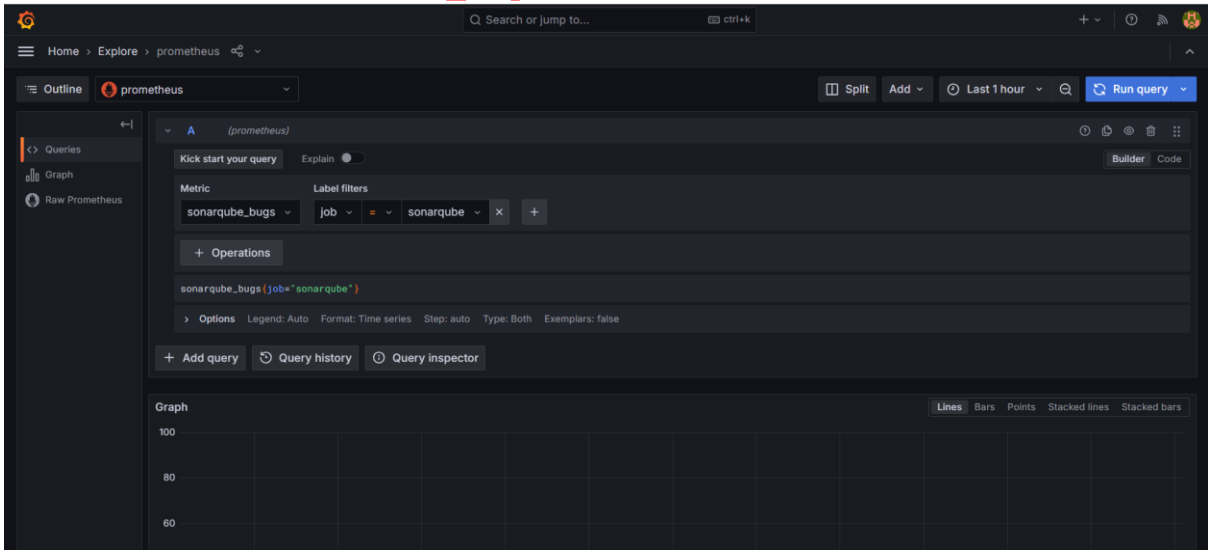
2. Now open SonarQube and go to Administration and in left side column you will see the prometheus exporter plugin as shown in screenshot attached below.

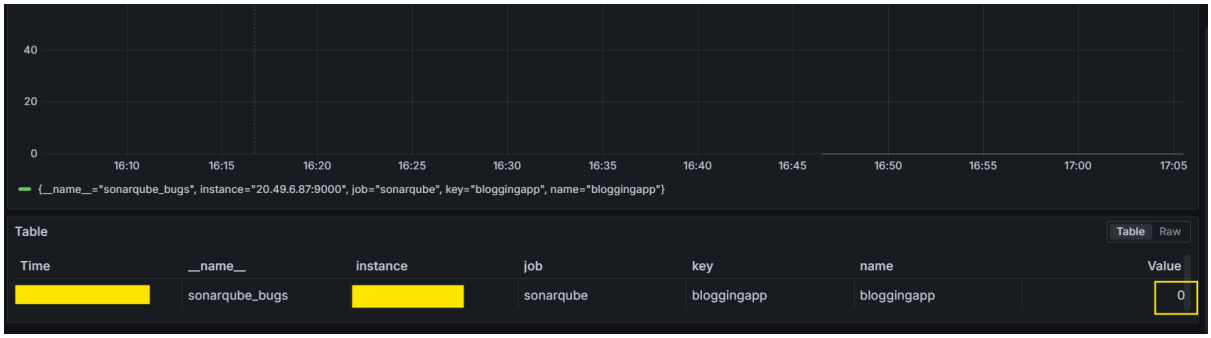


3. Restart the sonarqube service as shown in the screenshot attached below.

```
[root@SonarQube-Server plugins]# ls
README.txt  sonar-prometheus-exporter-1.0.0-SNAPSHOT.jar
[root@SonarQube-Server plugins]# su - sonar
[sonar@SonarQube-Server ~]$ sudo systemctl restart sonarqube
[sonar@SonarQube-Server ~]$ sudo systemctl status sonarqube
● sonarqube.service - SonarQube service
   Loaded: loaded (/etc/systemd/system/sonarqube.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2024-11-26 10:49:08 UTC; 6s ago
```

Now you can Explore the metrics as shown in the screenshot attached below.





Monitor All the Servers and AKS Cluster

To monitor all the servers and AKS Cluster using Prometheus and Grafana **Node Exporter** should be installed. It extracts the metrics from AKS Cluster and all the other Servers (where Node Exporter is installed) and then send them to Prometheus Server.

I had installed Node Exporter in AKS Cluster using the helm chart of Node Exporter as shown 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@devopsagent-vm ~]# kubectl get svc -n node-exporter
NAME                          TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)          AGE
my-prometheus-node-exporter   LoadBalancer   10.[redacted]  48.[redacted].177  9100:30885/TCP  [redacted]
```

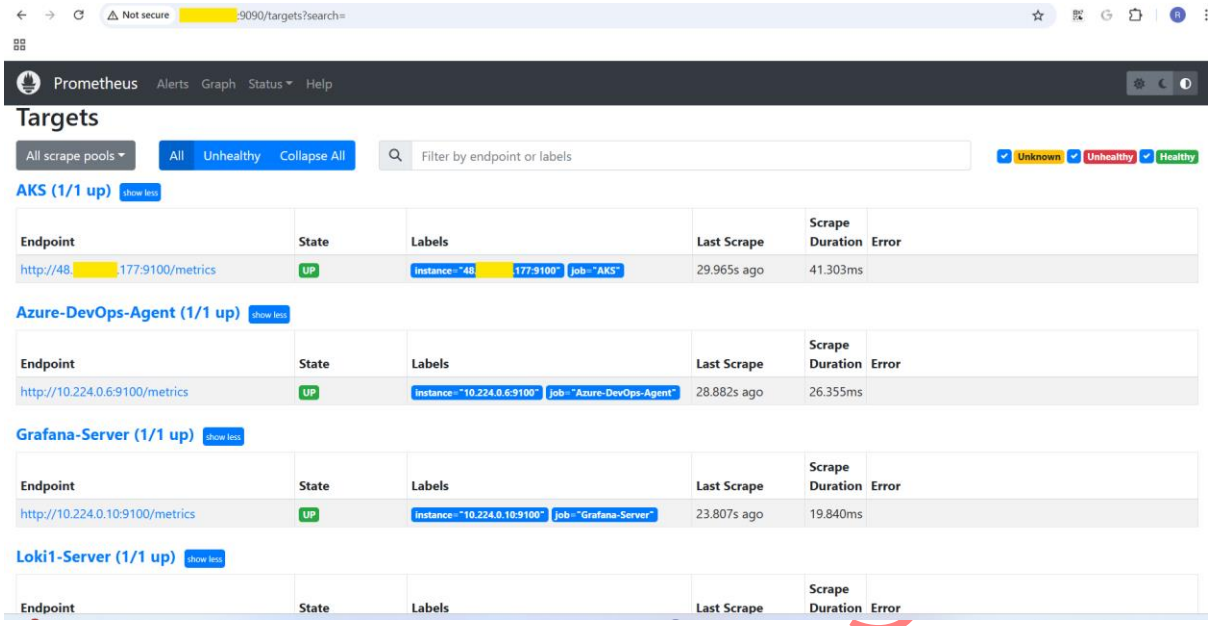
Configuration file for prometheus is as shown in the screenshot attached below.

```
    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ["localhost:9090"]
  - job_name: 'AKS'
    static_configs:
      - targets: ['48.[redacted].177:9100']
  - job_name: 'sonarqube'
    metrics_path: '/api/prometheus/metrics'
    static_configs:
      - targets: ['10.224.0.5:9000']
    basic_auth:
      username: '[redacted]'
      password: '[redacted]'
  - job_name: 'blackbox-exporter-server'
    static_configs:
      - targets: ['10.224.0.11:9100']
  - job_name: 'Azure-DevOps-Agent'
    static_configs:
      - targets: ['10.224.0.6:9100']
  - job_name: 'Grafana-Server'
    static_configs:
      - targets: ['10.224.0.10:9100']
  - job_name: 'Loki1-Server'
    static_configs:
      - targets: ['10.224.0.8:9100']
  - job_name: 'Loki2-Server'
    static_configs:
      - targets: ['10.224.0.4:9100']
  - job_name: 'Loki3-Server'
    static_configs:
      - targets: ['10.224.0.9:9100']
  - job_name: 'Prometheus-Server'
    static_configs:
      - targets: ['10.224.0.7:9100']
  - job_name: 'SonarQube-Server'
    static_configs:
      - targets: ['10.224.0.5:9100']
```

Restart prometheus service after any change in the configuration file as shown in the screenshot attached below.

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

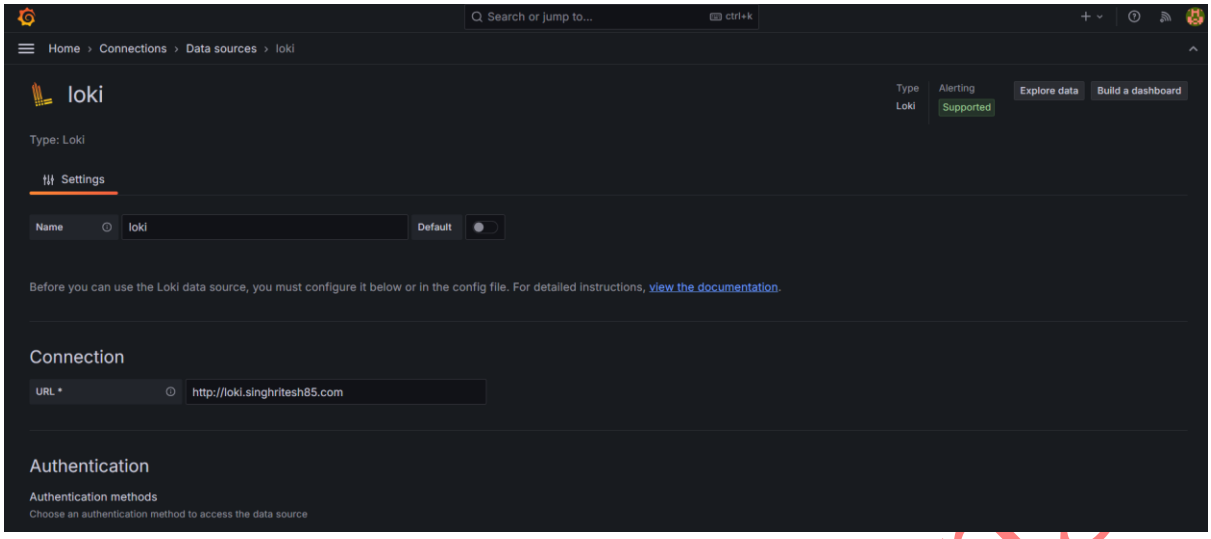
Below screenshot shows the Targets in Prometheus, make sure all the Targets should be in Up state.

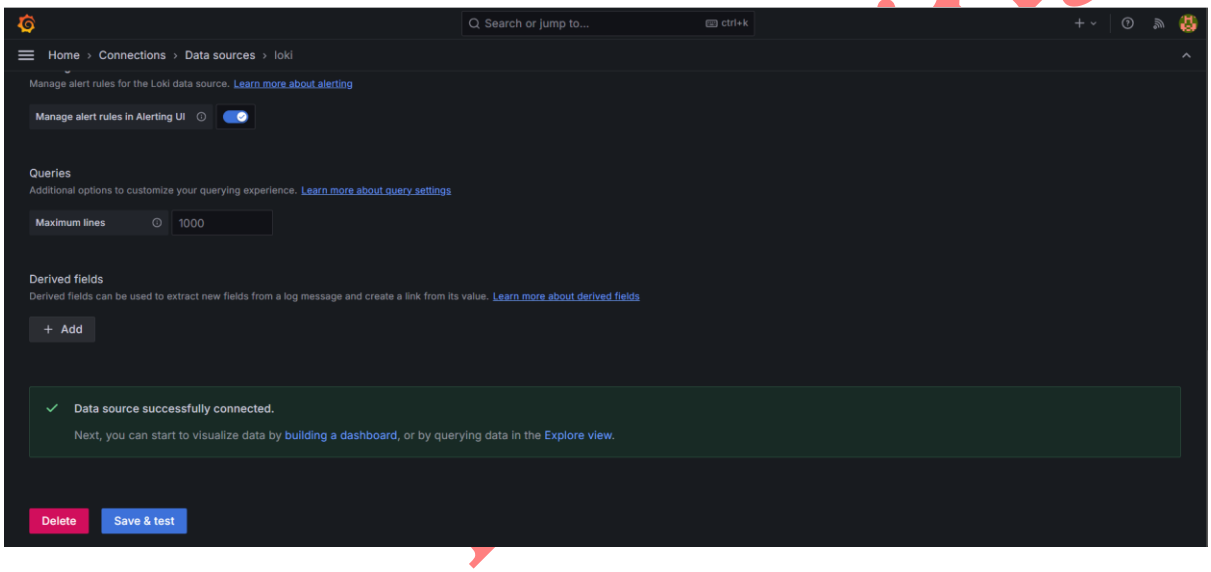


I had created the Grafana Dashboard for Node Exporter by importing the Grafana ID **1860** as shown in the screenshot attached below.

Log Aggregation using Loki, Promtail and Grafana

To aggregate Logs using Loki, Promtail and Grafana I had used Loki as a Source which is shown in the screenshot attached below.





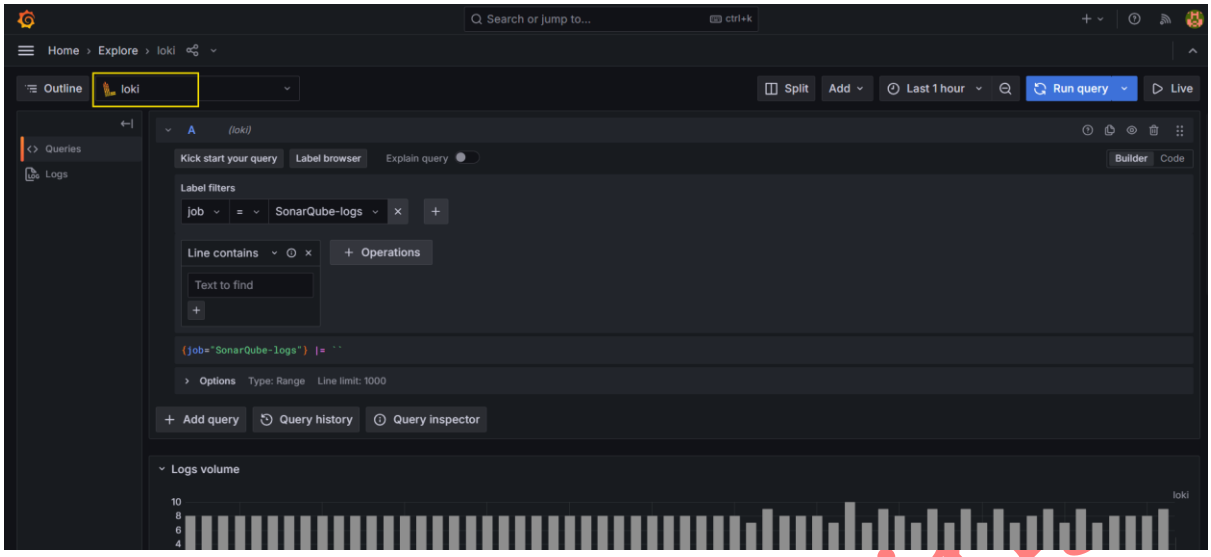
I had installed promtail on AKS Cluster using helm chart as shown in the screenshot attached below.

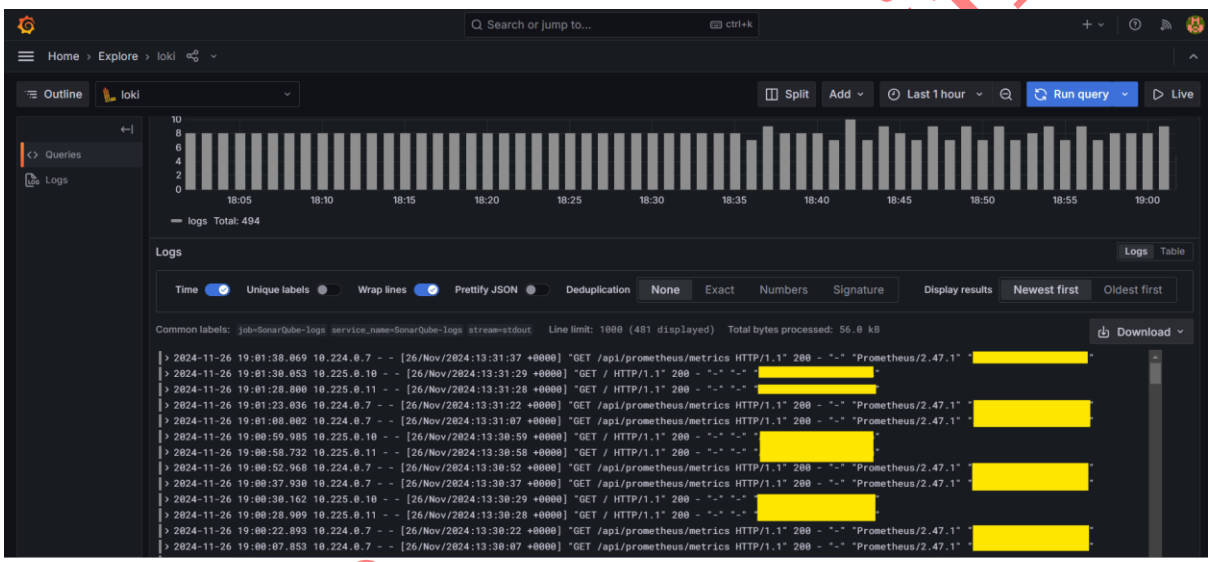
kubectl create ns promtail

helm upgrade --install promtail -f helm-chart-promtail/values.yaml ./helm-chart-promtail -n promtail

```
[root@devopsagent-vm ~]# helm upgrade --install promtail -f helm-chart-promtail/values.yaml ./helm-chart-promtail -n promtail
Release "promtail" does not exist. Installing it now.
```

Finally, you can explore the Logs using the filers as shown in the screenshot attached below.





Monitoring Application URL using blackbox exporter

Application URL **https://blogapp.singhritesh85.com** had been monitored using blackbox exporter. The configuration file for blackbox exporter had been shown in the screenshot attached below.

```
[root@blackboxexporter-vm ~]# cat /opt/blackbox_exporter_linux_amd64/monitor_website.yml
modules:
  http_2xx_example:
    prober: http
    timeout: 5s
    http:
      valid_http_versions: ["HTTP/1.1", "HTTP/2.0"]
      valid_status_codes: []  # Defaults to 2xx
      method: GET
      tls_config:
        insecure_skip_verify: true
```

After change in the blackbox exporter configuration file restarted the its service as shown in the screenshot attached below.

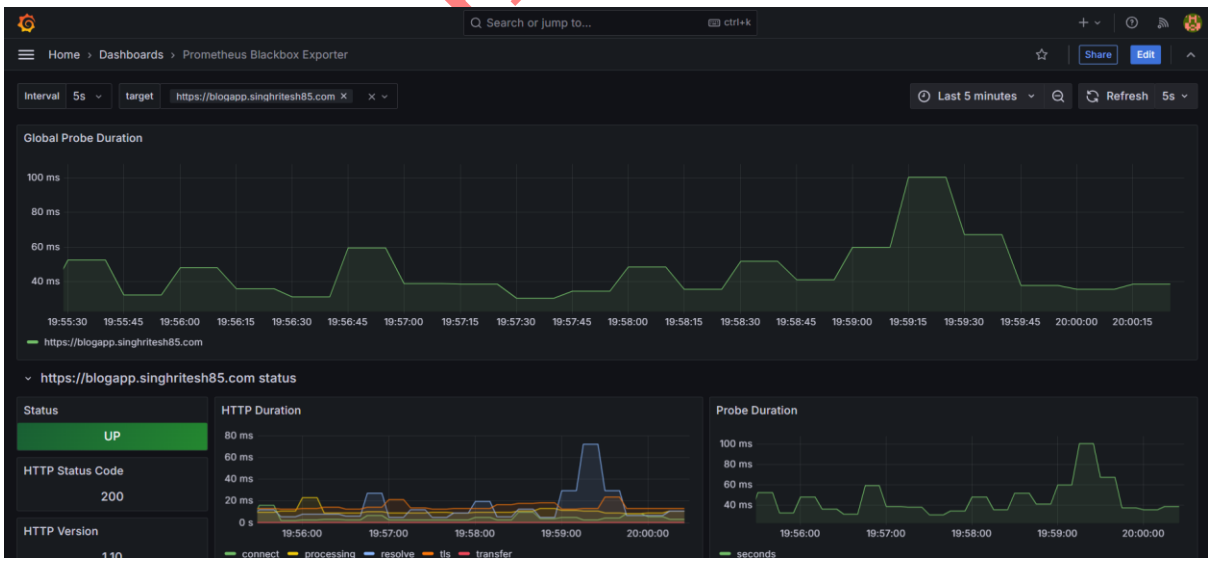
```
[root@blackboxexporter-vm ~]# systemctl restart blackbox_exporter.service
[root@blackboxexporter-vm ~]# systemctl status blackbox_exporter.service
● blackbox_exporter.service - Blackbox Exporter
   Loaded: loaded (/etc/systemd/system/blackbox_exporter.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue [redacted] s ago
```

Below changes had been done in the prometheus configuration file for blackbox exporter and restarted prometheus service as shown in the screenshot attached below.

```
  - job_name: 'blackbox'
    metrics_path: /probe
    params:
      module: [http_2xx_example]  # Look for a HTTP 200 response.
    static_configs:
      - targets:
          - https://blogapp.singhritesh85.com
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: 10.224.0.11:9115  # The blackbox exporter's real hostname:port.
```

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

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

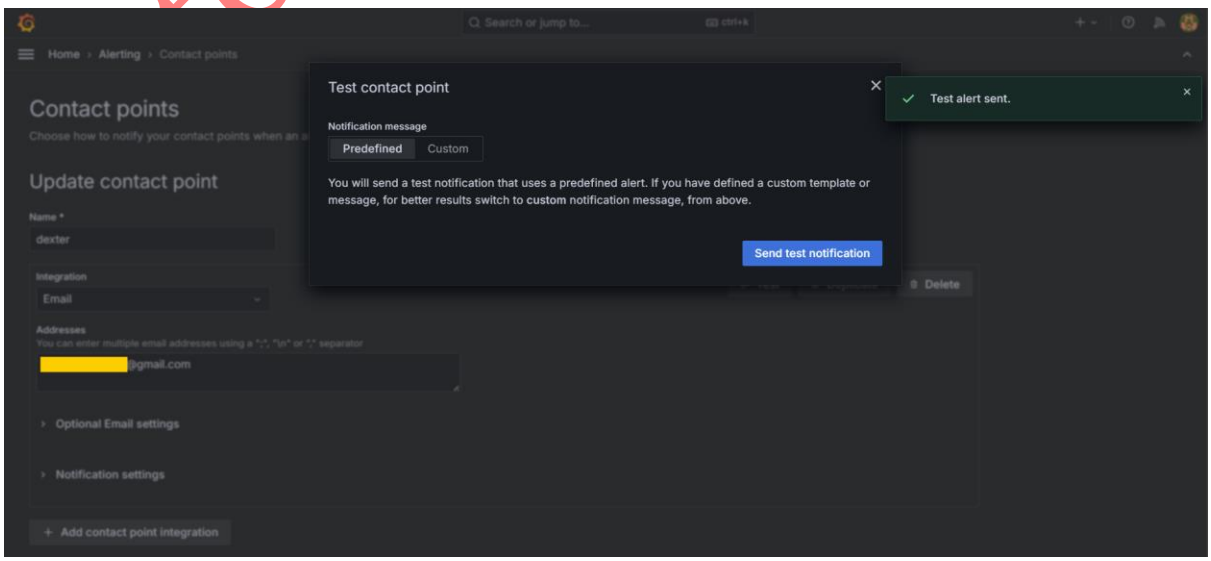
To configure Email Alerts in Grafana I changed the configuration for grafana as shown in the screenshot attached below and then restarted the service for grafana and checked its status as shown in the screenshot attached below.

```
[root@grafana-vm ~]# vim /etc/grafana/grafana.ini
```

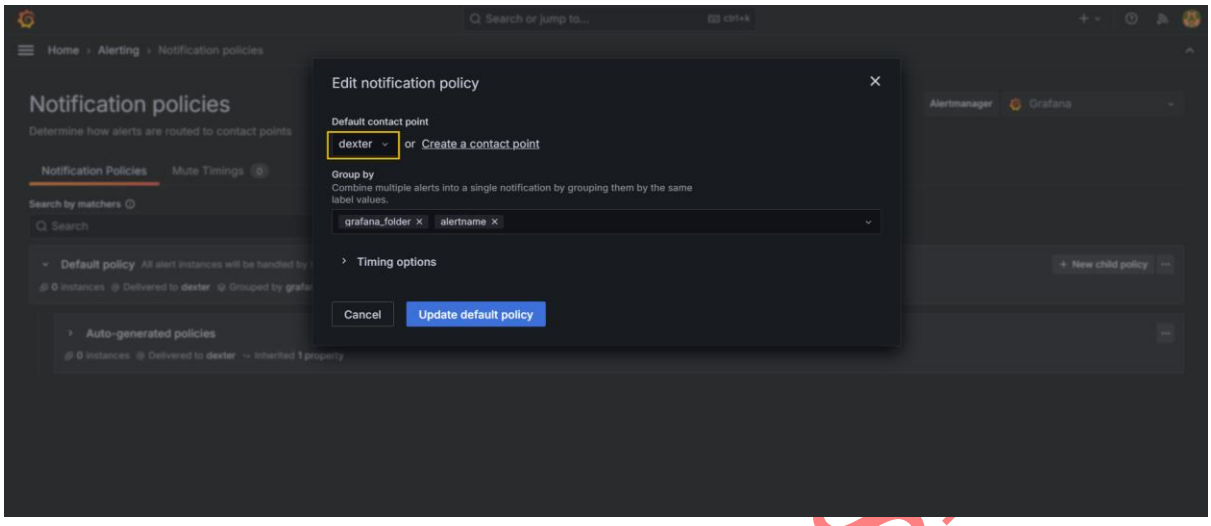
```
# Specifies whether Entra password auth can be used for the MSSQL data source
# Disabled by default, needs to be explicitly enabled
;azure_entra_password_credentials_enabled = false

#################################### Role-based Access Control ###########
[rbac]
;permission_cache = true

# Reset basic roles permissions on boot
# Warning left to true, basic roles permissions will be reset on every boot
#reset_basic_roles = false

# Validate permissions' action and scope on role creation and update
; permission_validation_enabled = true

#################################### 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
```

```
[root@grafana-vm ~]# systemctl restart grafana-server.service
[root@grafana-vm ~]# systemctl status grafana-server.service
● grafana-server.service - Grafana instance
   Loaded: loaded (/usr/lib/systemd/system/grafana-server.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon [redacted]s ago
```

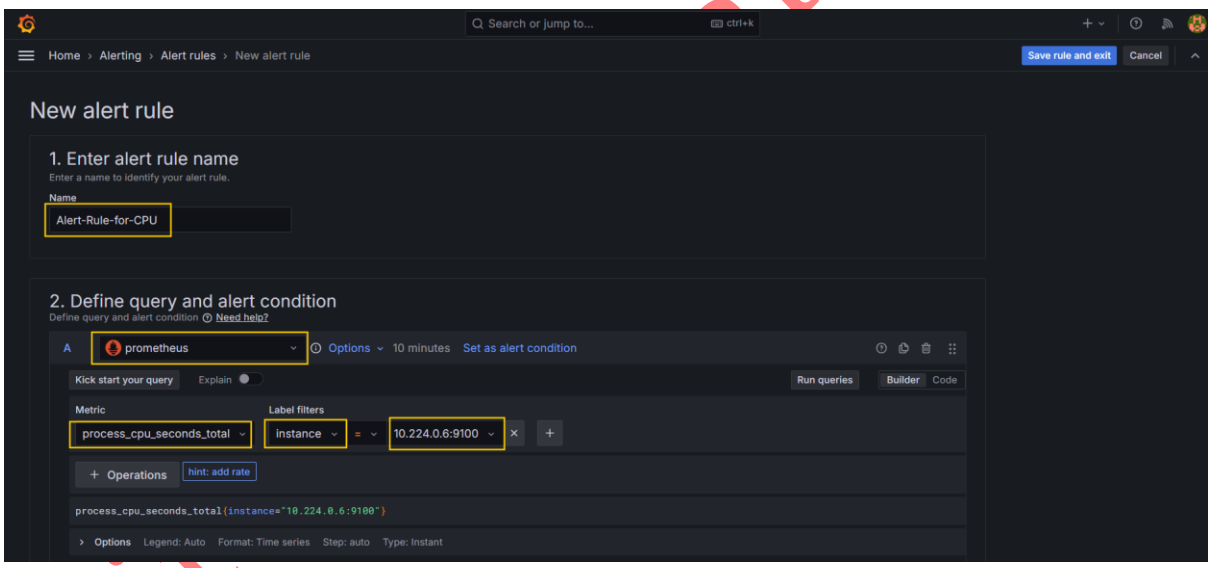
The **contact points** in Grafana Alerts had been created and tested as shown in the screenshot attached below.

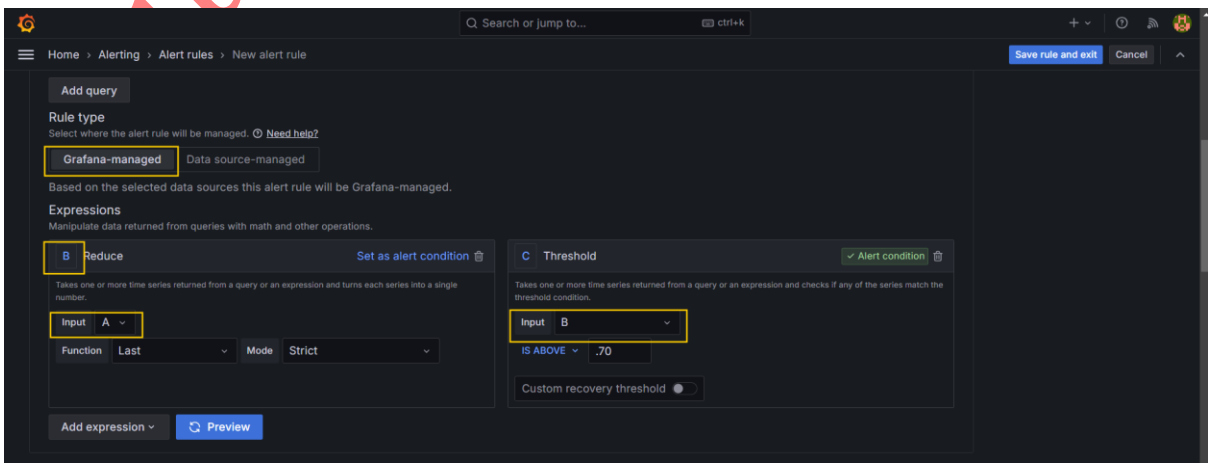


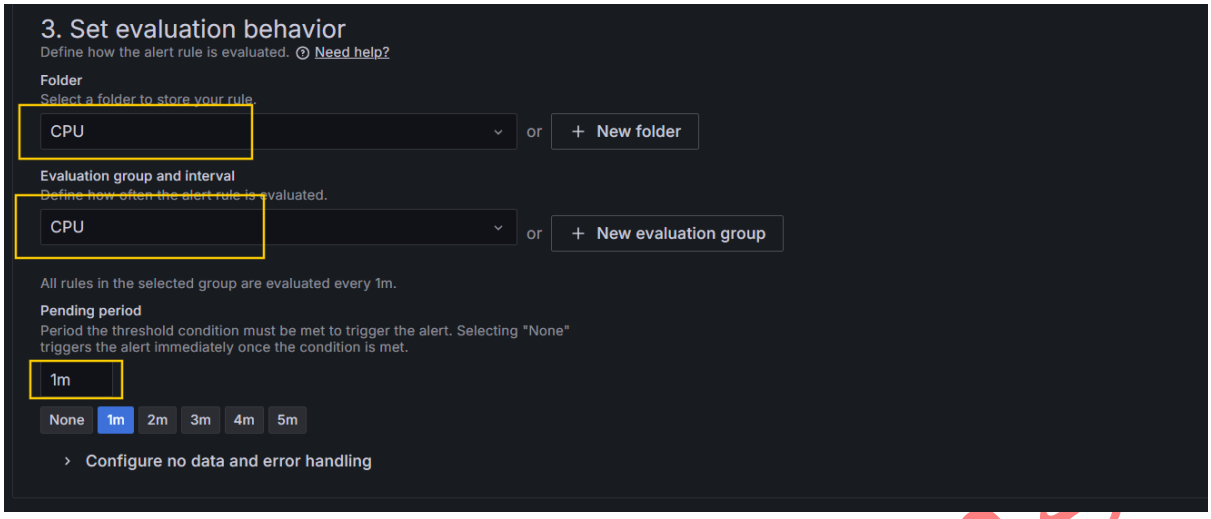
The default policy for the **Notification Policy** is as shown in the screenshot attached below.

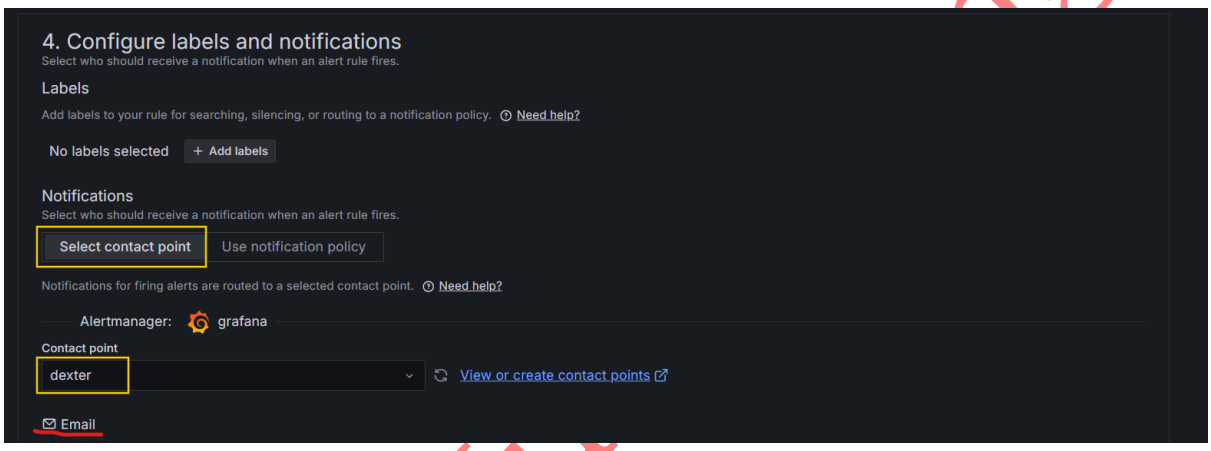


Finally create the **Alert Rule** in Grafana Alerts as shown in the screenshot attached below.

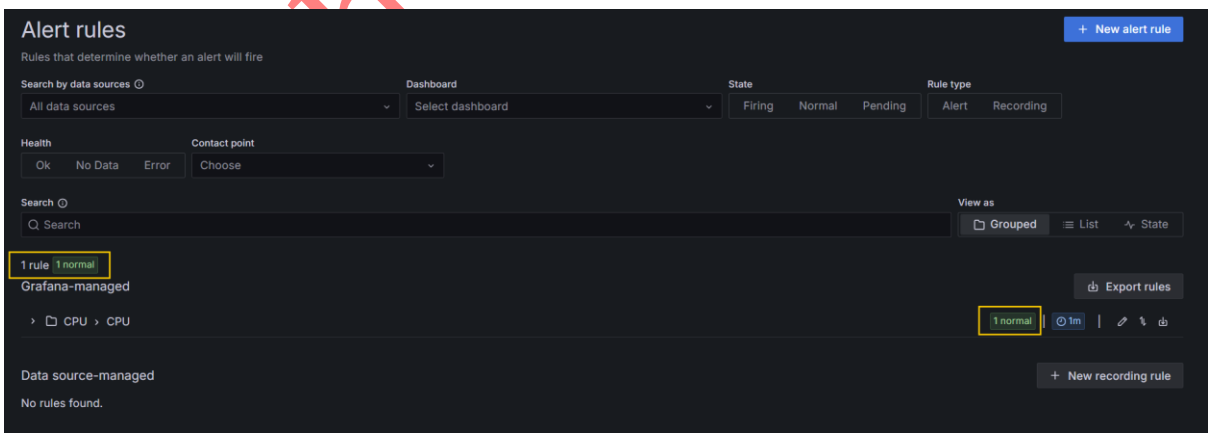






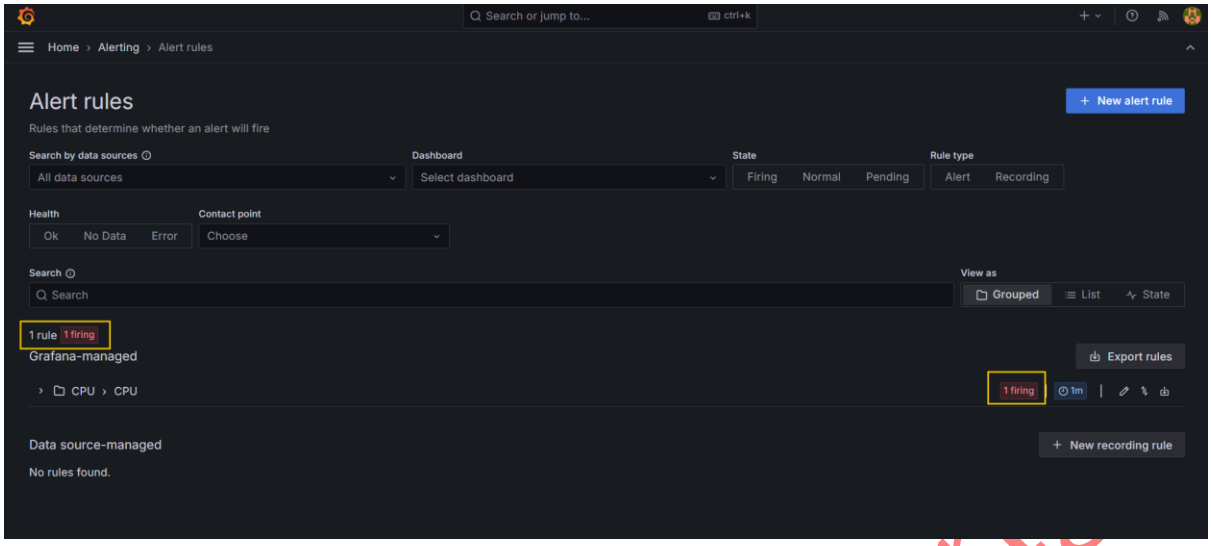


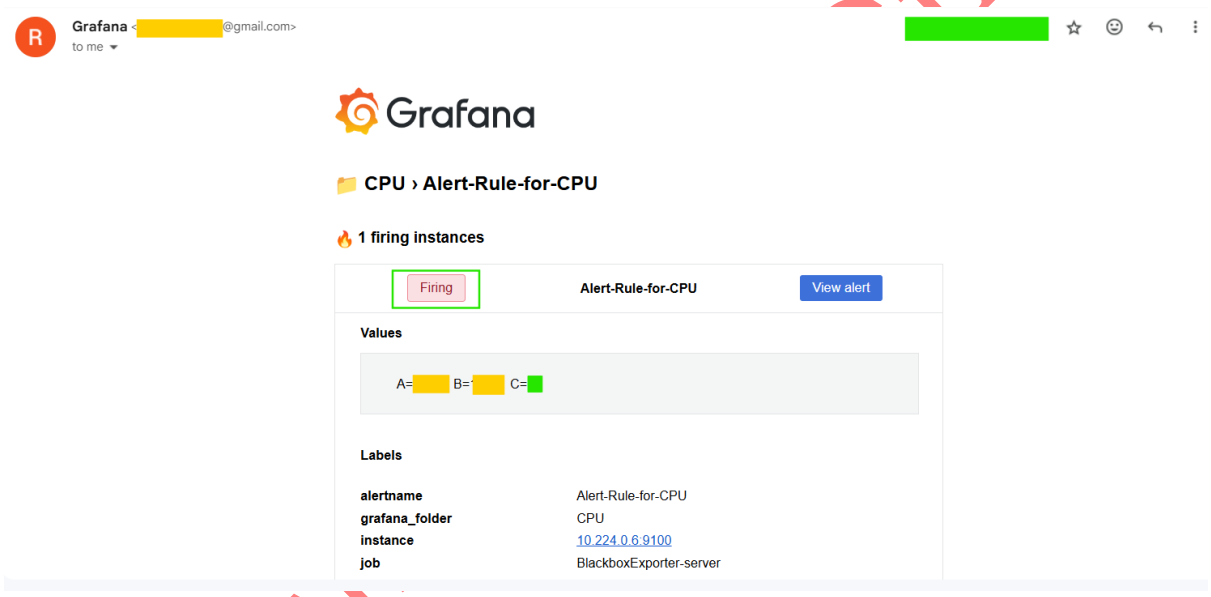
After creation of Alert Rule we can this Alert Rule in Normal Condition as shown in the screenshot attached below.



Whenever the Alert Rule crosses the Threshold, it will Trigger the Grafana Alert and will send the Email on Group Email Id of your team, so that the team will get notified on this issue.

After sometimes it crosses the threshold and this Alert will come into the Firing Condition as shown in the screenshot attached below and an email will be sent to the Group Email Id.





After the team received this Email on their Email Id, they will investigate its RCA (Root Cause Analysis) and will check the CPU usage using the command **htop**. Team will check whether any unnecessary crontab, any process is running which will utilize more CPU. If yes then edit the crontab and comment out that entry for crontab or kill that process. If necessary, team will check the Log file to investigate this issue or if needed upgrade the VM Size of Azure VM.