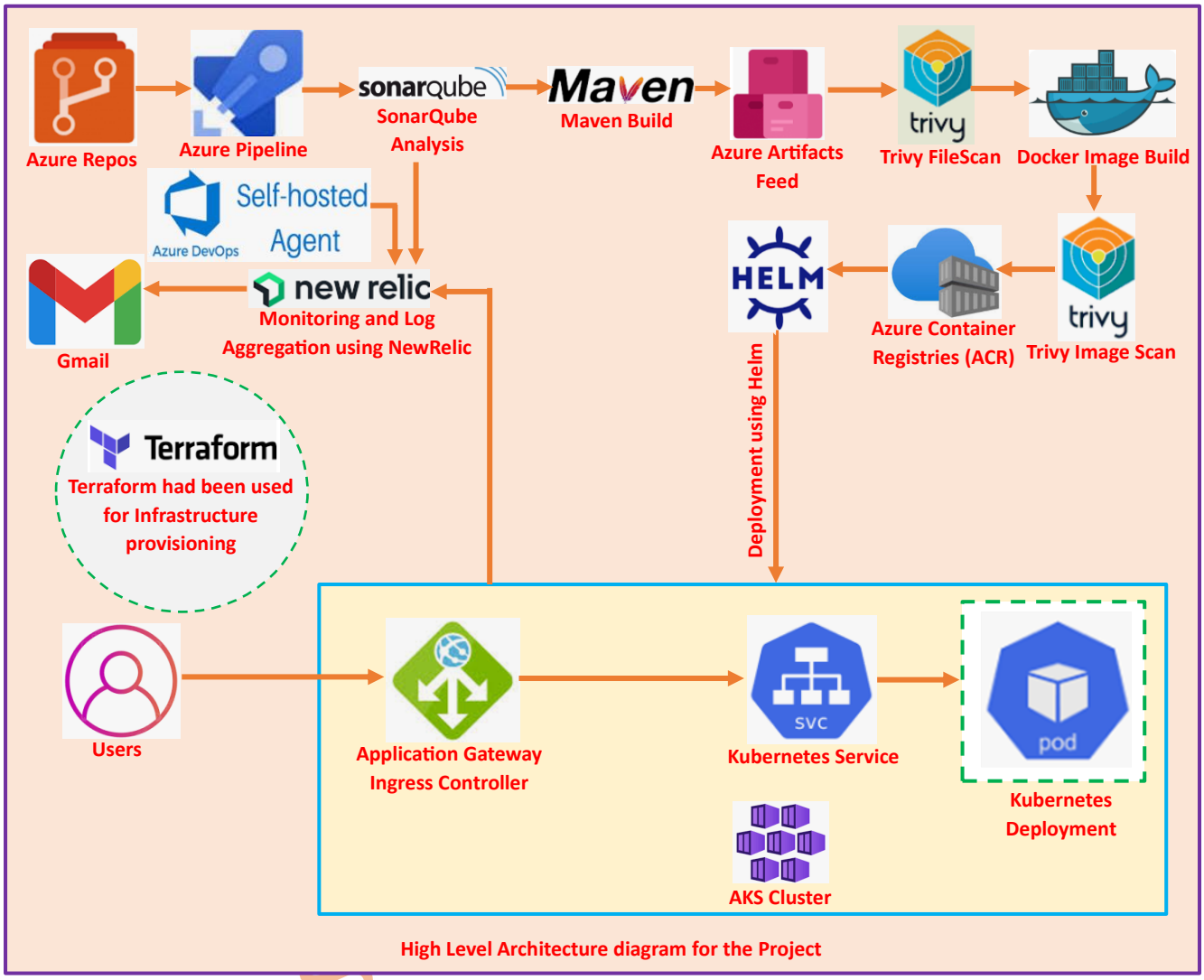
DevOps Project Blogging App Deployment Monitoring and Log Aggregation using NewRelic Azure



High Level Architecture diagram for the Project

This DevOps Project aims to create the infrastructure using Terraform and established the CICD set-up using Azure DevOps. For monitoring and Log Aggregation NewRelic had been used which was also shown in the screenshot attached above. The source code was present in the Azure Repos, which was deployed through the Azure DevOps Pipeline. SonarQube had been used for Code Analysis and Maven was used as the Build Tool, artifacts for this project were kept in the Azure Artifacts Feed. Trivy was used as the File Scan and Docker Image Scan which was also shown in the Architecture diagram shown above. The created Docker Image was kept in Azure Container Registries (ACR) which was deployed to Azure Kubernetes Services (AKS Cluster) using the Helm as shown in the Architecture diagram shown above.

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

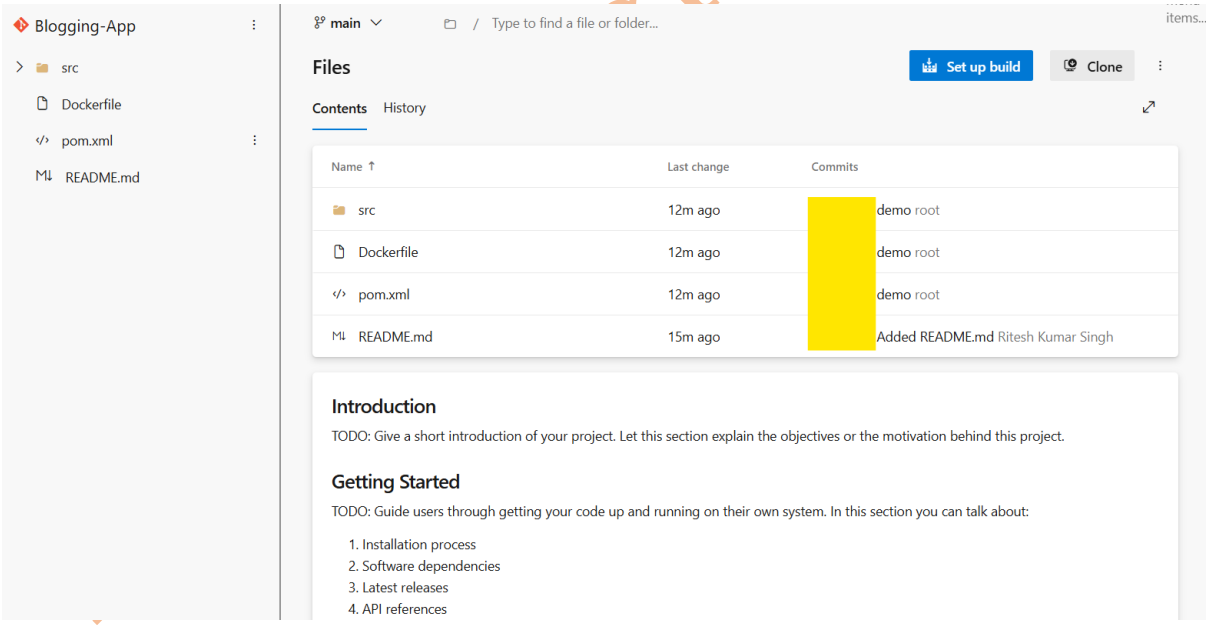
```
[root@devopsagent-vm ~]# cd /opt && mkdir myagent && cd myagent
[root@devopsagent-vm myagent]# wget https://vstsagentpackage.azureedge.net/agent/[redacted]/vsts-agent-linux-x64-[redacted].tar.gz
[root@devopsagent-vm myagent]# tar -xvf vsts-agent-linux-x64-[redacted] tar.gz
[root@devopsagent-vm myagent]# rm -f vsts-agent-linux-x64-[redacted].tar.gz
[root@devopsagent-vm myagent]# ./bin/installdependencies.sh
```

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

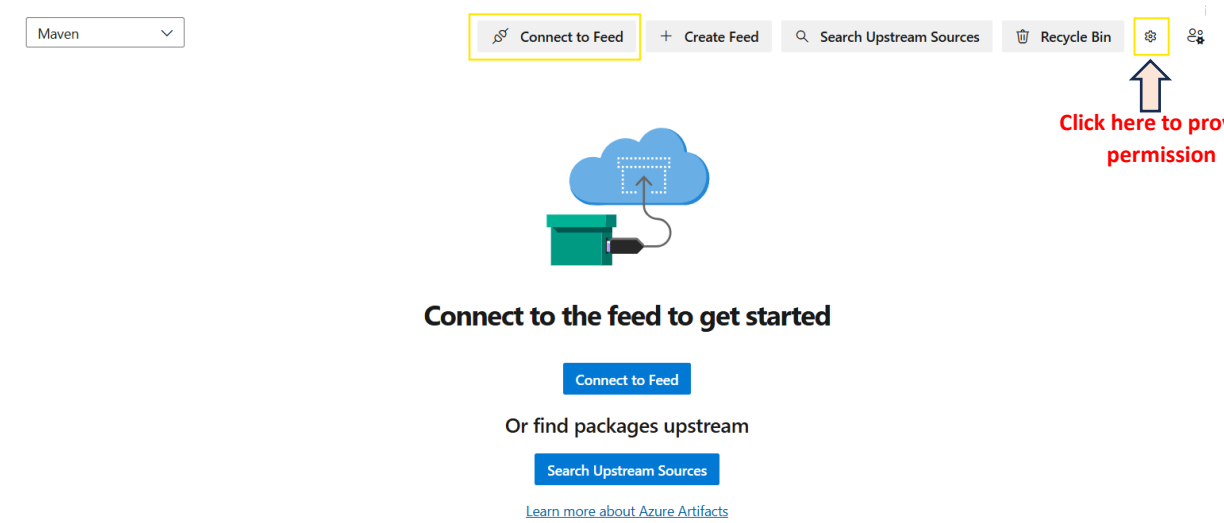
```
[demo@devopsagent-vm myagent]$ ./env.sh
[demo@devopsagent-vm myagent]$
[demo@devopsagent-vm myagent]$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/bin:/opt/sonar-scanner/bin:/opt/apache-maven/bin:/opt/node-v16.0.0/bin:/opt/dependency-check/bin:/usr/local
/bin
```

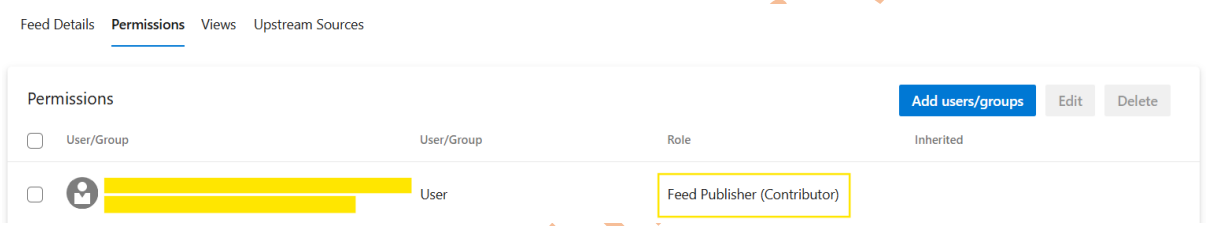
```
[demo@devopsagent-vm myagent]$ sudo ./svc.sh install
[demo@devopsagent-vm myagent]$ sudo ./svc.sh start
```

Below screenshot shows the source code which was present in the Azure Repos.

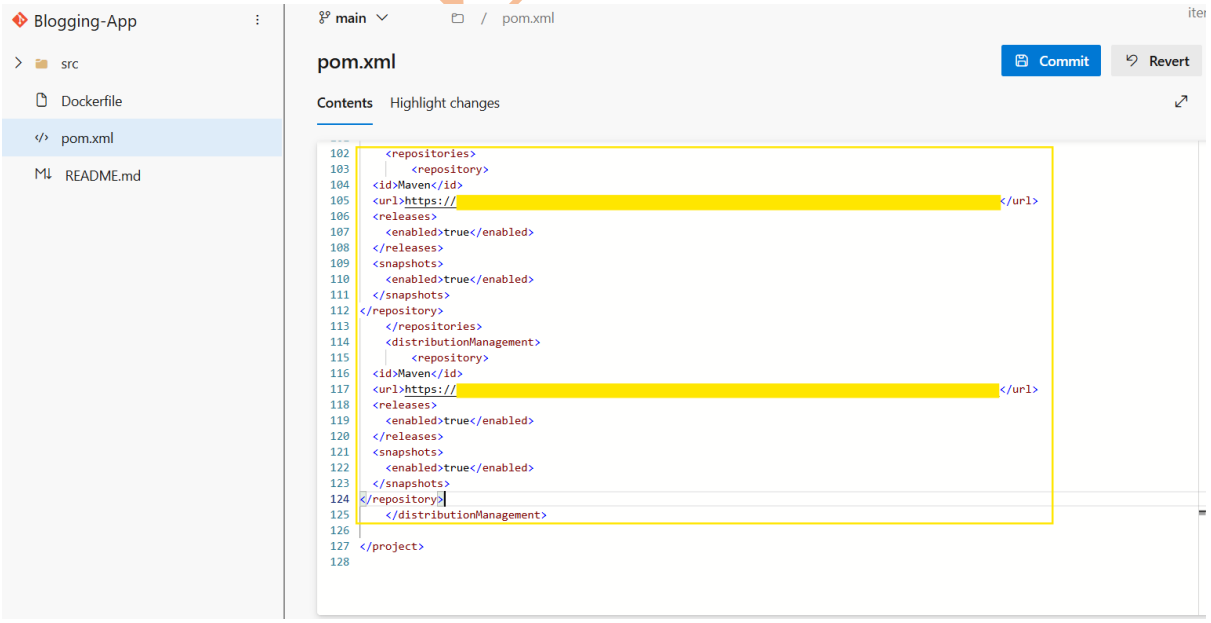


I had created Azure Artifacts Feed named as Maven and provided contributor access for the deployment user from Azure Artifacts Feed settings which was shown in the screenshot attached below.

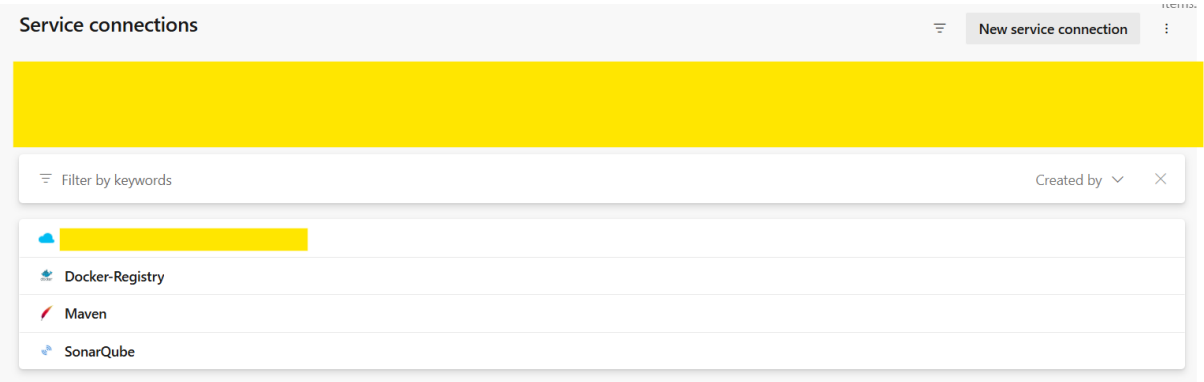




I had updated the pom.xml for Feed connection as shown in the screenshot attached below.



Service connection for Azure DevOps had been created for **SonarQube**, **Azure Artifacts Feed** and **Docker Registry** as shown in the screenshot attached below.



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

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

```
[root@ ~]# 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 Secrets in the namespace blogapp had been listed out using the command **kubectl get secrets -n blogapp** as shown in the screenshot attached below.

```
[root@ ~]# kubectl get secrets -n blogapp
NAME                TYPE                                  DATA   AGE
bloggingapp-auth    kubernetes.io/dockerconfigjson        1
ingress-secret      kubernetes.io/tls                     2
```

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 the deployment user demo, access was restricted to the namespace blogapp.

```
[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
[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
```

```
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@                    ~]# kubectl  apply -f secret.yaml
secret/mysecret created
[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 describe secret mysecret -n blogapp
Name:         mysecret
Namespace:    blogapp
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: demo
              kubernetes.io/service-account.uid:

Type:  kubernetes.io/service-account-token

Data
====
namespace:  7 bytes
token:      e
ca.crt:     1765 bytes
```

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 secret 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

====

ca.crt: 1765 bytes

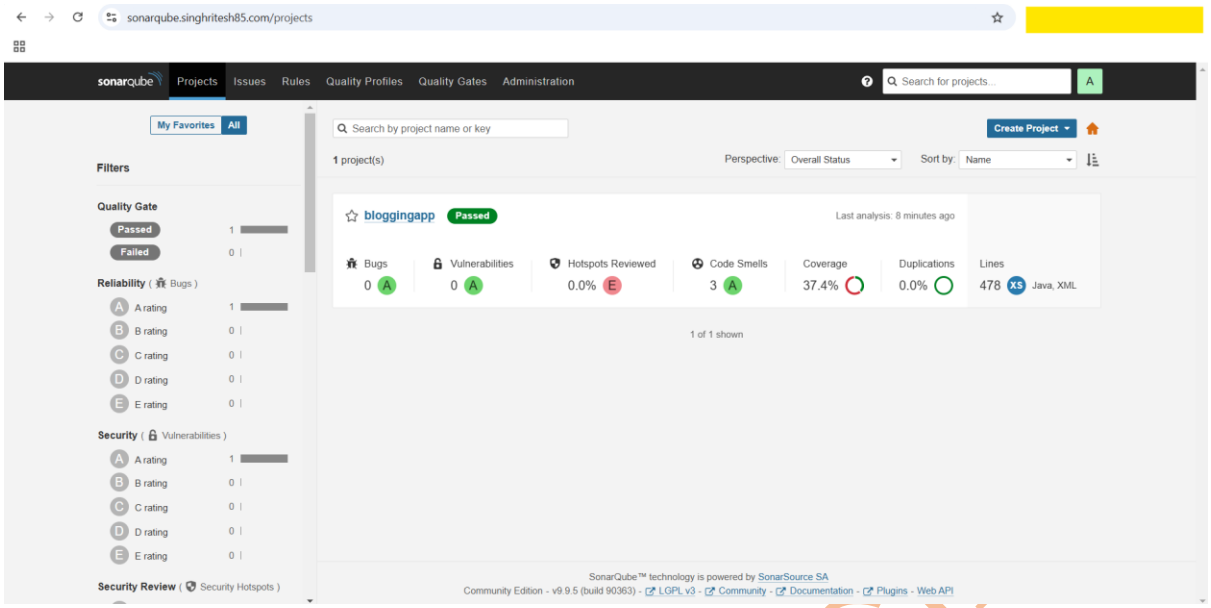
namespace: 7 bytes

token:
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

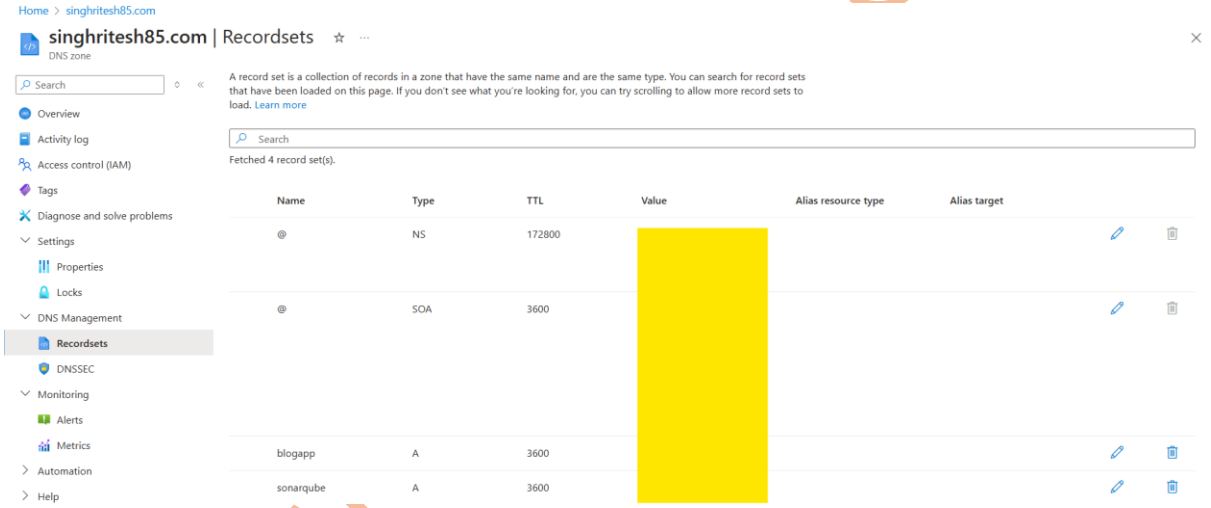
Below screenshot shows the kubeconfig file which was shared with the deployment user demo.

```
cat ~/.kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    server: https://blogapp-cluster-dns-XXXXXXXX.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: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```


The screenshot for SonarQube after running the Azure DevOps Pipeline is as shown in the screenshot attached below.



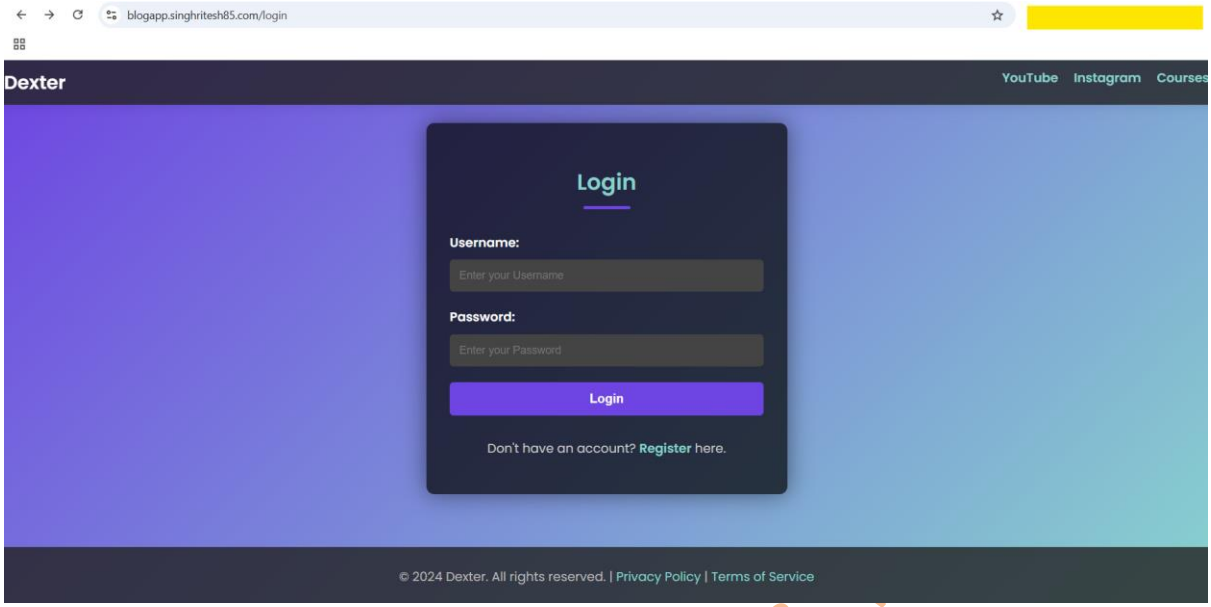
Below screenshot shows the entry for Azure DNS Zone for creation of Record Set.



After running Azure DevOps Pipeline Kubernetes Pods, Kubernetes Service and Kubernetes Deployment had been created as shown in the screenshot attached below.

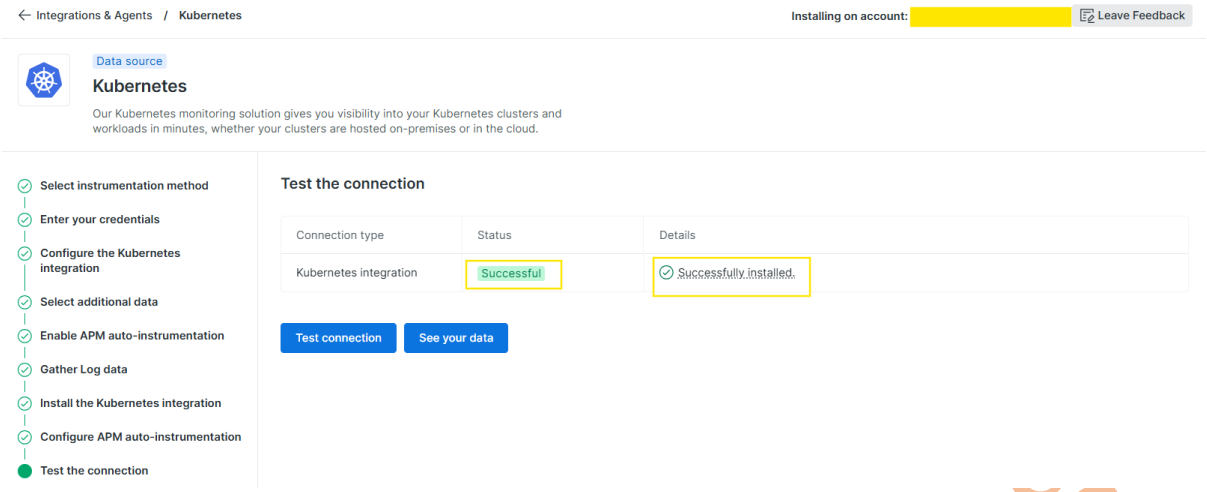
```
[demo@devopsagent-vm ~]$ kubectl get all -n blogapp
NAME                                READY   STATUS    RESTARTS   AGE
pod/blogapp-folo-[redacted]         1/1     Running   0          53m
pod/blogapp-folo-[redacted]         1/1     Running   0          53m

NAME                    TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
service/blogapp-folo    ClusterIP   10.[redacted]   <none>        80/TCP    121m

NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/blogapp-folo    2/2     2            2           121m

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/blogapp-folo-[redacted]    0         0         0       66m
replicaset.apps/blogapp-folo-[redacted]    2         2         2       53m
[demo@devopsagent-vm ~]$ kubectl get ing -n blogapp
NAME              CLASS                        HOSTS                       ADDRESS          PORTS     AGE
blogapp-ingress   azure-application-gateway    blogapp.singhritesh85.com   135.[redacted]   80, 443   5h30m
[demo@devopsagent-vm ~]$ kubectl get get nodes
error: the server doesn't have a resource type "get"
```

Finally, I access the Blogging Application using the newly created URL as shown in the screenshot attached below.

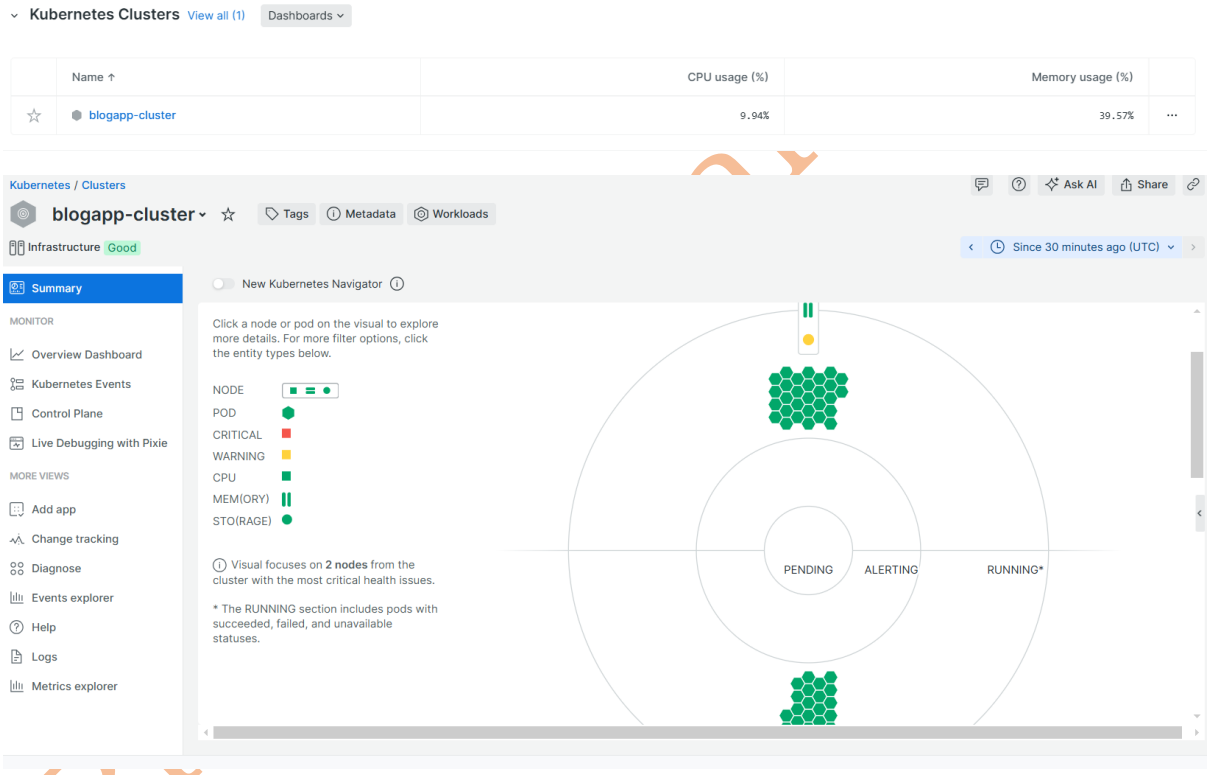


For Monitoring AKS Cluster using NewRelic I installed the NewRelic Agent as shown in the screenshot attached below.

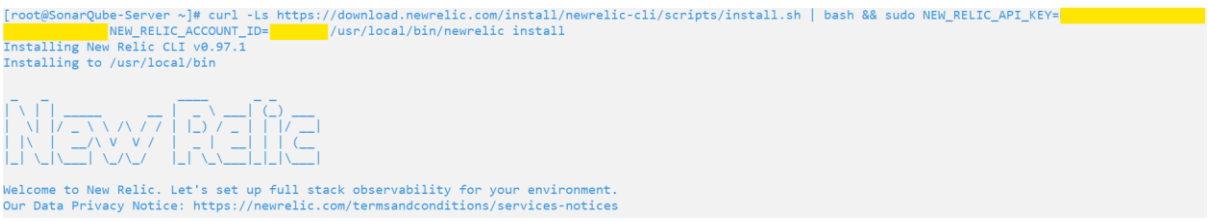
```
[root@                ~]# KSM_IMAGE_VERSION="v2.10.0" && helm repo add newrelic https://helm-charts.newrelic.com && helm repo update && kubectl create namespac
e newrelic ; helm upgrade --install newrelic-bundle newrelic/nri-bundle --set global.licenseKey=                                    --set global.cluster=
blogapp-cluster --namespace=newrelic --set newrelic-infrastructure.privileged=true --set global.lowDataMode=true --set kube-state-metrics.image.tag=${KSM_IMAG
E_VERSION} --set kube-state-metrics.enabled=true --set kubeEvents.enabled=true --set newrelic-prometheus-agent.enabled=true --set newrelic-prometheus-agent.lo
wDataMode=true --set newrelic-prometheus-agent.config.kubernetes.integrations_filter.enabled=false --set k8s-agents-operator.enabled=true --set logging.enable
d=true --set newrelic-logging.lowDataMode=true
"newrelic" has been added to your repositories
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "newrelic" chart repository
Update Complete. ⎈Happy Helming!⎈
namespace/newrelic created
Release "newrelic-bundle" does not exist. Installing it now.
NAME: newrelic-bundle
LAST DEPLOYED: Mon Nov
NAMESPACE: newrelic
STATUS: deployed
REVISION: 1
TEST SUITE: None
[root@                ~]# kubectl get pods -n blogapp
NAME                          READY   STATUS    RESTARTS   AGE
blogapp-folo-                 1/1     Running   0
blogapp-folo-                 1/1     Running   0
[root@                ~]# kubectl get pods -n newrelic
NAME                                                  READY   STATUS    RESTARTS   AGE
newrelic-bundle-k8s-agents-operator-                  2/2     Running   0
newrelic-bundle-kube-state-metrics-                   1/1     Running   0
newrelic-bundle-newrelic-logging-                     1/1     Running   0
newrelic-bundle-newrelic-logging-                     1/1     Running   0
newrelic-bundle-newrelic-prometheus-agent-0           1/1     Running   0
newrelic-bundle-nri-kube-events-                      2/2     Running   0
newrelic-bundle-nri-metadata-injection-               1/1     Running   0
newrelic-bundle-nrk8s-ksm-                            2/2     Running   0
newrelic-bundle-nrk8s-kubelet-                        2/2     Running   0
newrelic-bundle-nrk8s-kubelet-                        2/2     Running   0
[root@                ~]#
```



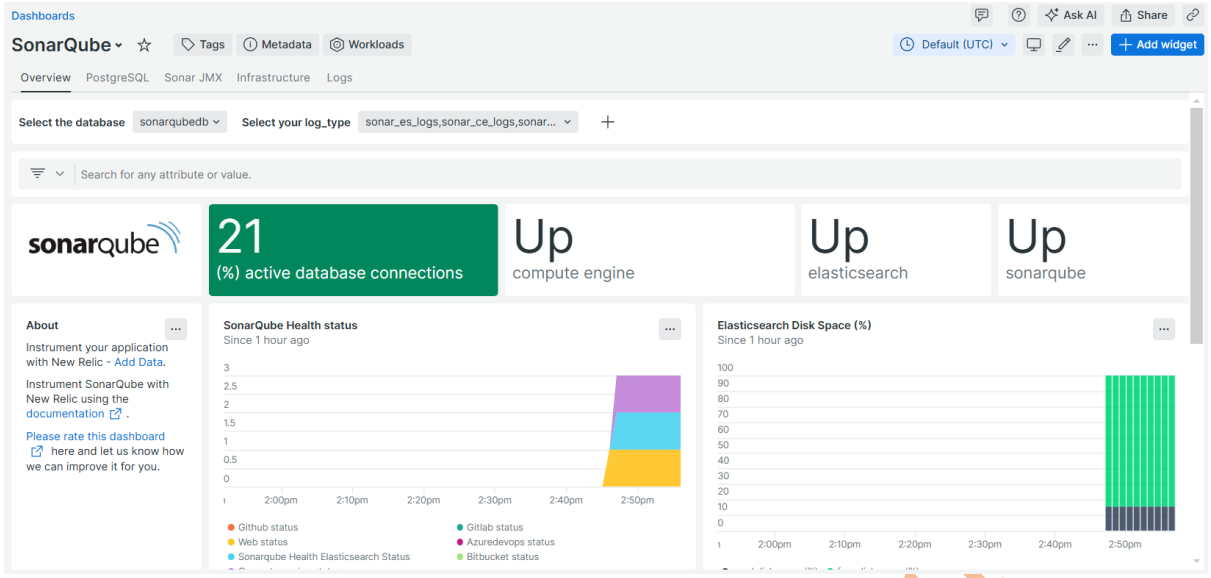
After successful installation of NewRelic Agent on AKS Cluster we saw the AKS Cluster with the name of blogapp-cluster in NewRelic Console as shown in the screenshot attached below.



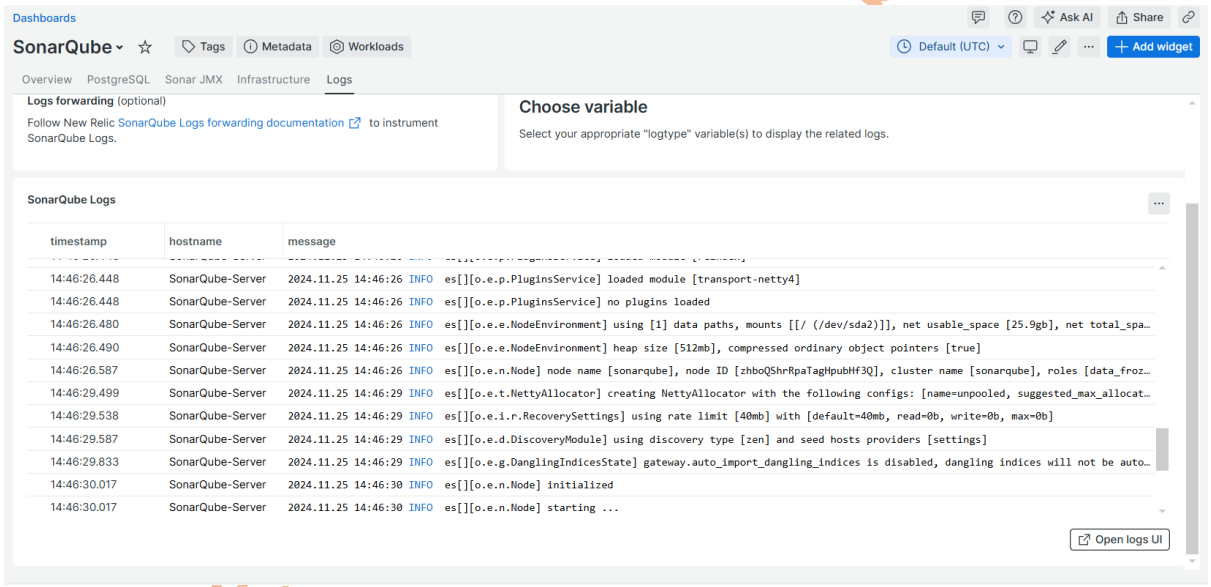
For Monitoring SonarQube using NewRelic I installed the NewRelic Agent for SonarQube on SonarQube Server as shown in the screenshot attached below.



After successful installation of NewRelic Agent on SonarQube Server I was able to see the sonarqube dashboard for monitoring in NewRelic as shown in the screenshot attached below.



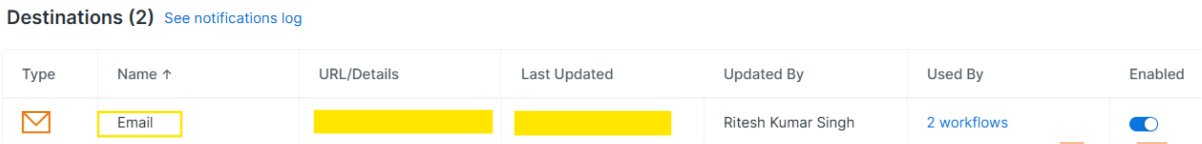
As shown in the screenshot attached below I was able to see the SonarQube logs in the NewRelic console.



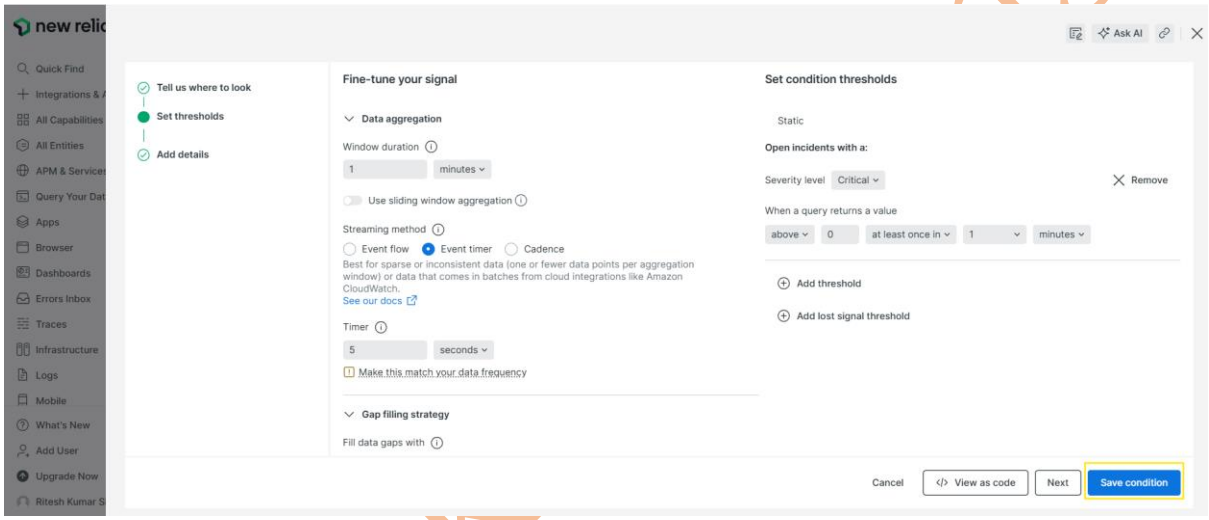
Alerts in NewRelic

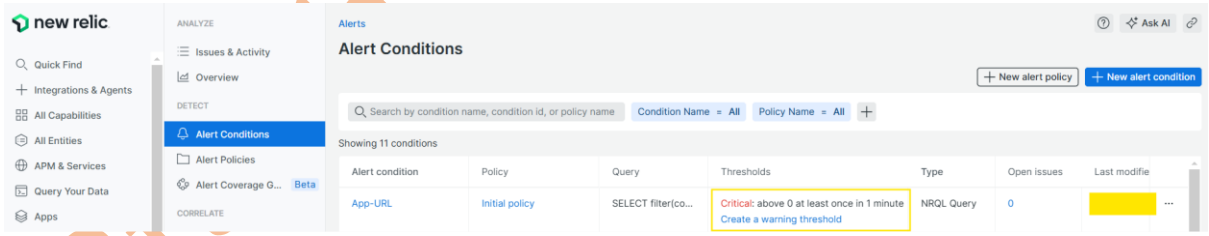
For Alerts in NewRelic we need to discuss about **Destination**, **Alert Condition**, **Alert Policy**, and **Workflows**.

I had created an Email destination with the Group Email ID of the concerned team as shown in the screenshot attached below.

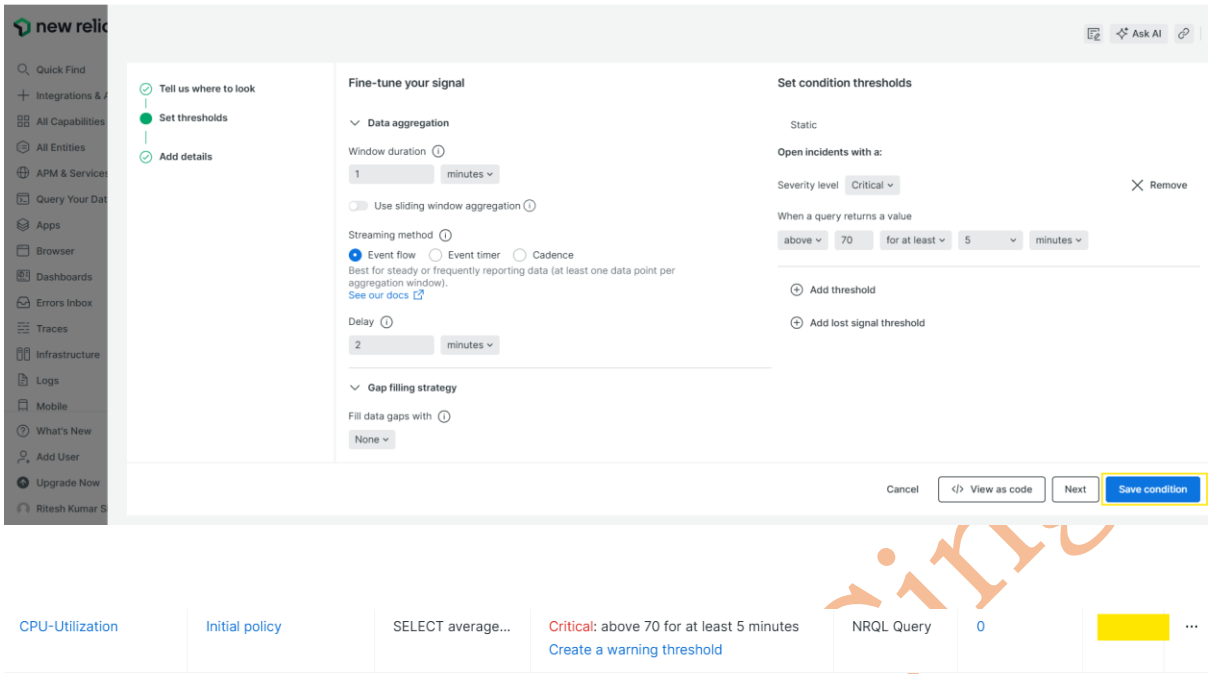


An alert condition had been created for Synthetic Monitoring as shown in the screenshot attached below.



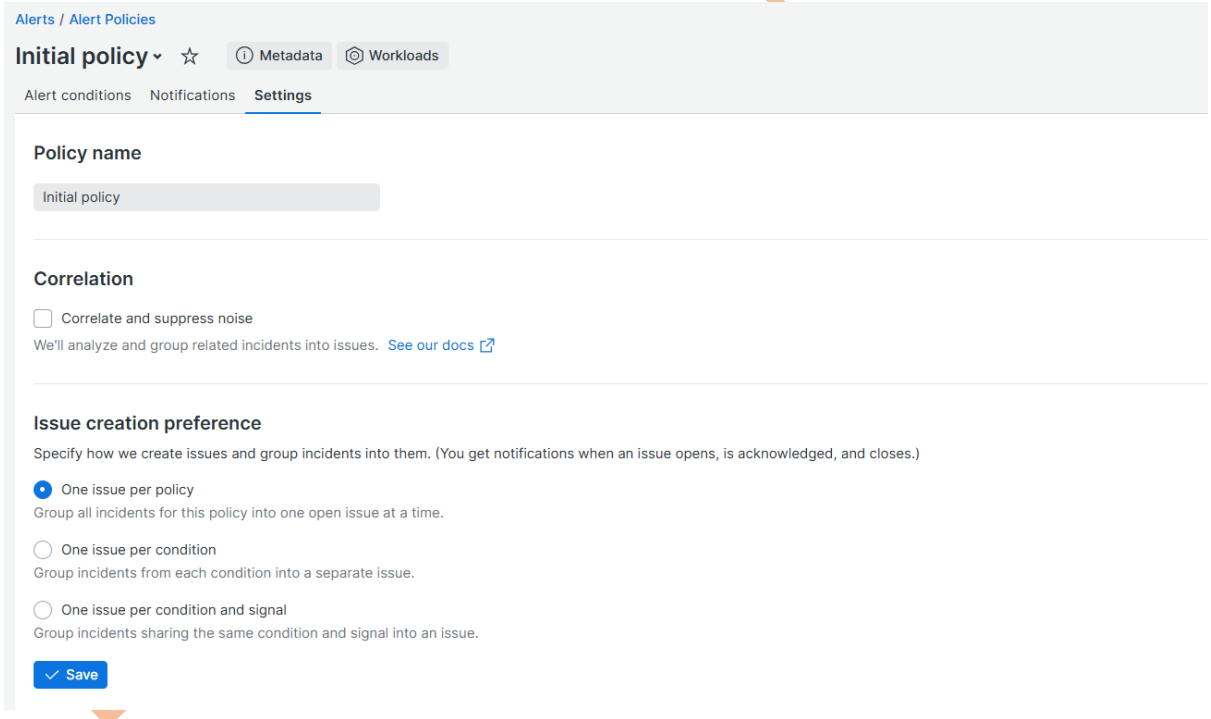


An Alert condition for CPU Utilization had been created as shown in the screenshot attached below.

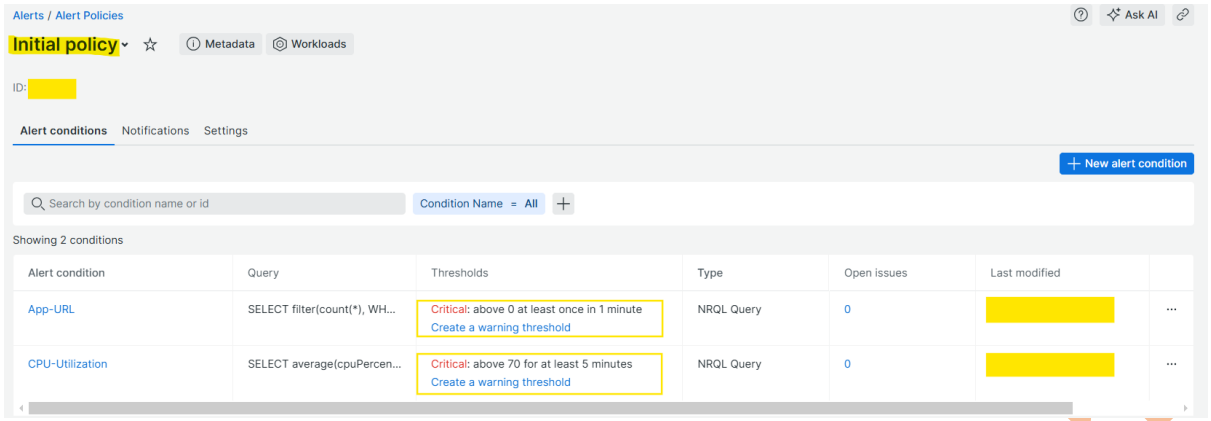


CPU-Utilization	Initial policy	SELECT average...	Critical: above 70 for at least 5 minutes Create a warning threshold	NRQL Query	0	...

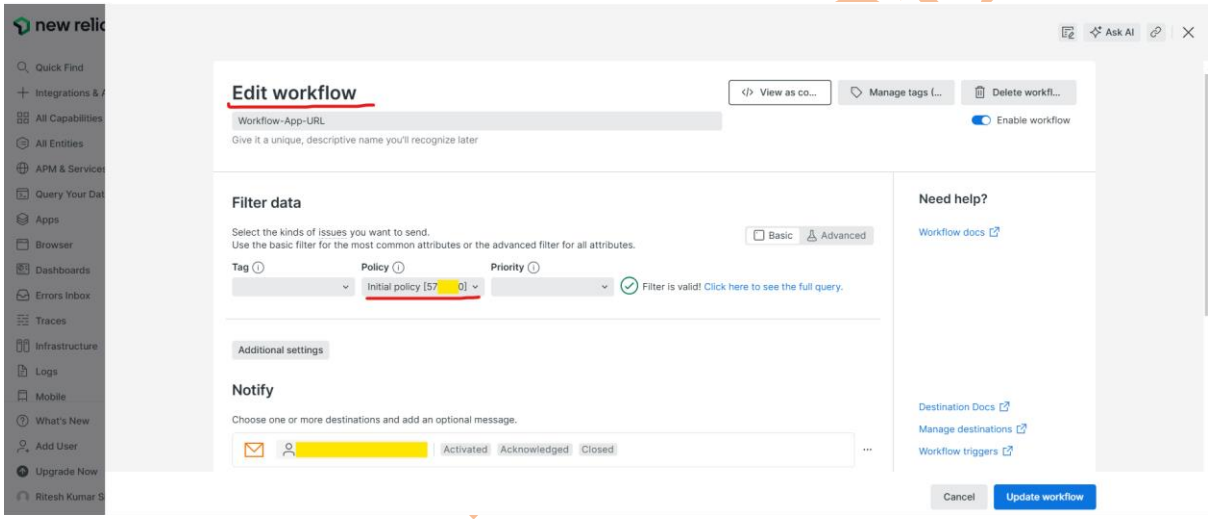
An Alert Policy had been created as shown in the screenshot attached below.



With this Alert Policy two Alert Conditions had been associated as shown in the screenshot attached below.

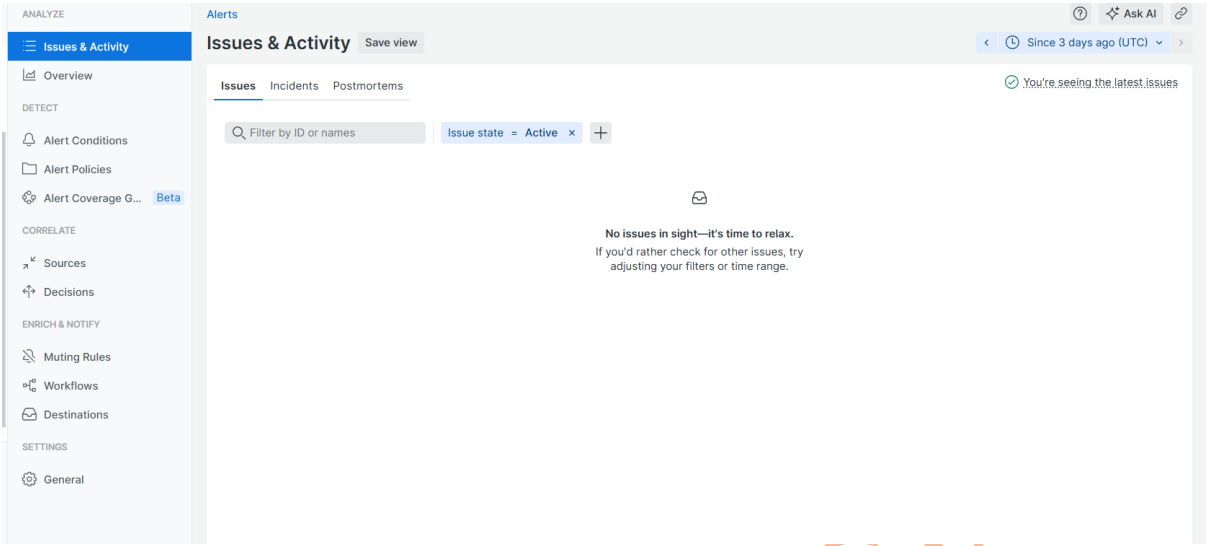


The Workflow had been shown in the screenshot attached below which showed with this Alert Workflow an Alert Policy, and an Email Destination had been attached.

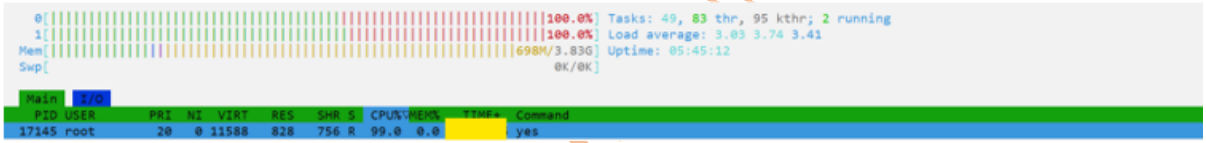


At present there is no Alert, but for the demonstration purpose I had created Alert by increasing the CPU Utilization more than 70% and I had deleted the Application URL https://blogapp.singhritesh85.com entry from the Azure DNS Zone Record set for which I had created Synthetic Monitoring to monitor this Application URL.

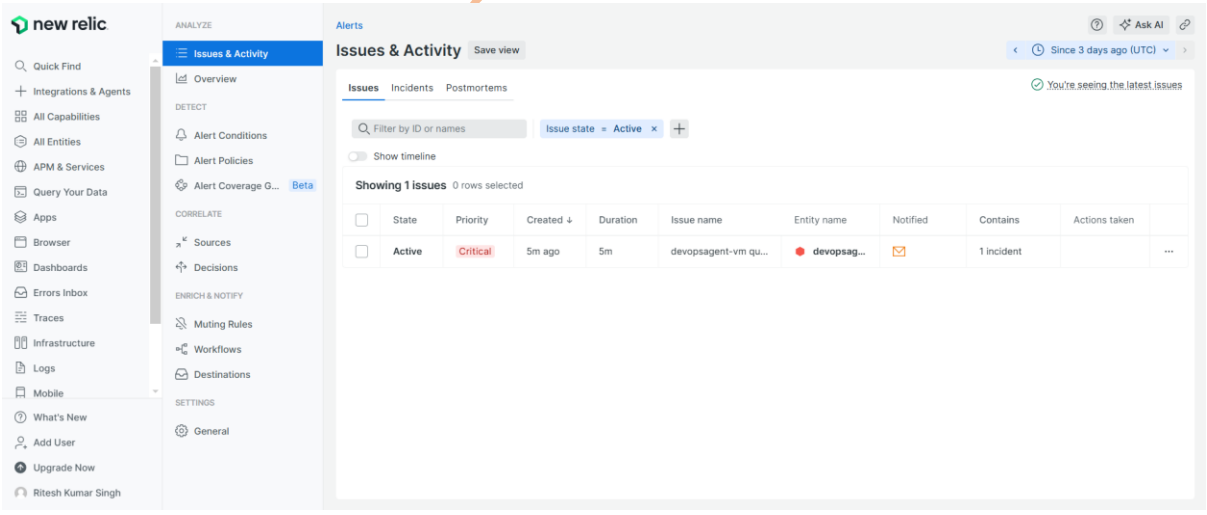
Below screenshot showed no Active alerts now.



For demonstration purpose I increased the CPU Utilization with the help of the command **yes > /dev/null &** and checked the same using the **htop** command. The screenshot for the same is as shown below.

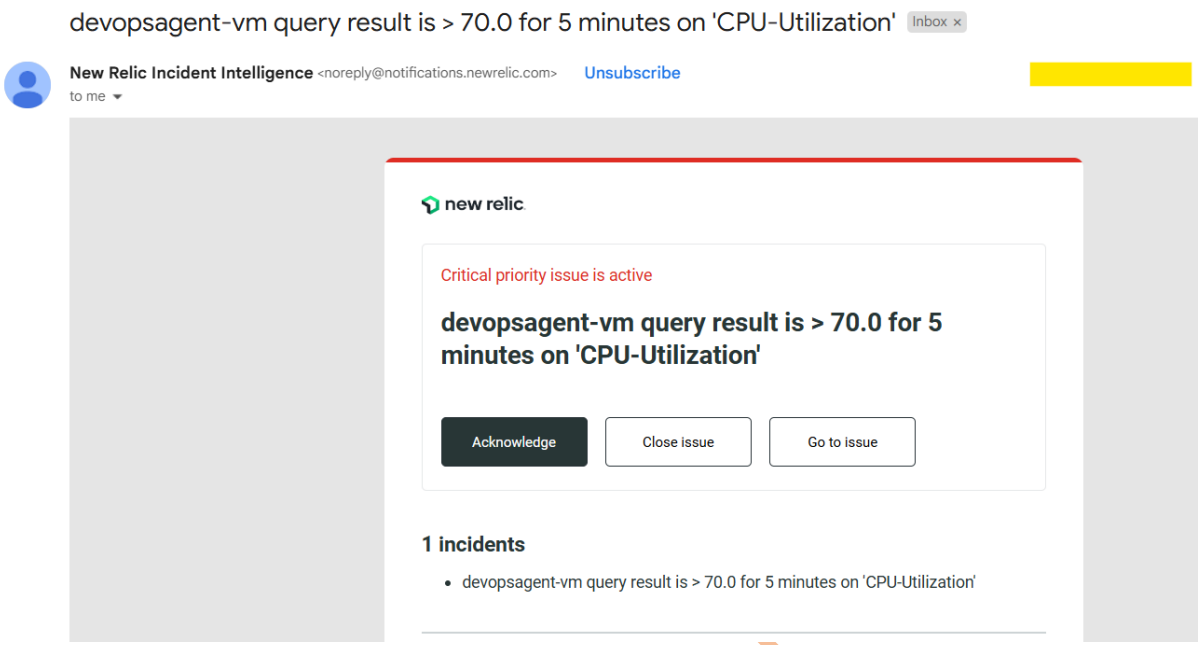


Now the Alert is in Active condition as shown in the screenshot attached below.

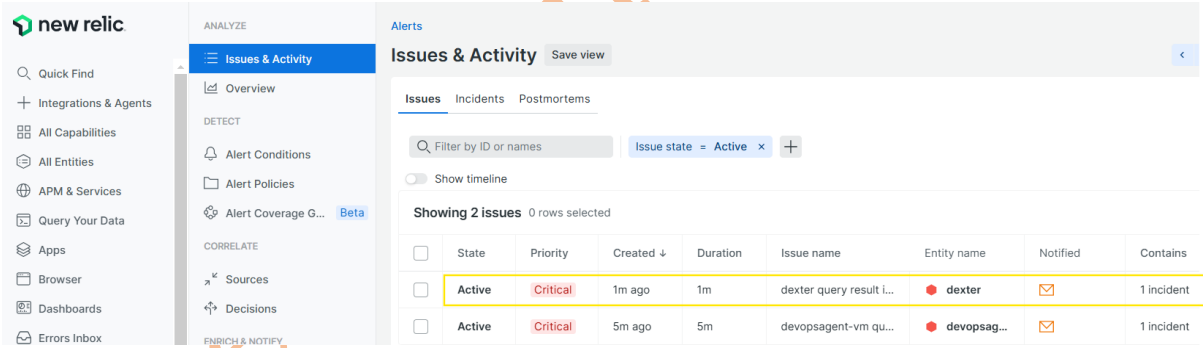


An Email had been triggered to the Group Email ID as shown in the screenshot attached below. After the team received this Email on their Email Id, they will investigate its RCA (Root Cause Analysis) and will check the CPU usage using the command **htop**. Team will check whether any unnecessary crontab, 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.



The Alert for Synthetic Monitoring is also in Active condition as shown in the screenshot attached below.



After the concerned team will get notification on their group Email Id as shown in the screenshot attached below, they will Acknowledge the issue and do RCA (Root Cause Analysis). As a part of RCA, they are supposed to check the entry in Azure DNS zone for record set if it is not present then they will make an entry for the same and this issue will be resolved. It is very important that concerned team should resolve the issue as per the SLA (between the organisation and its client). For demonstration purpose I had deleted the record set form Azure DNS Zone but, in your organization, if any one deleted an Azure Resource and you want to find out the person who deleted it then you need to check the Azure Monitor > Activity Log. Then filter out the Subscription into which the Azure Resource was existed and Timespan. You will be able to see the person who had initiated that Event.

.