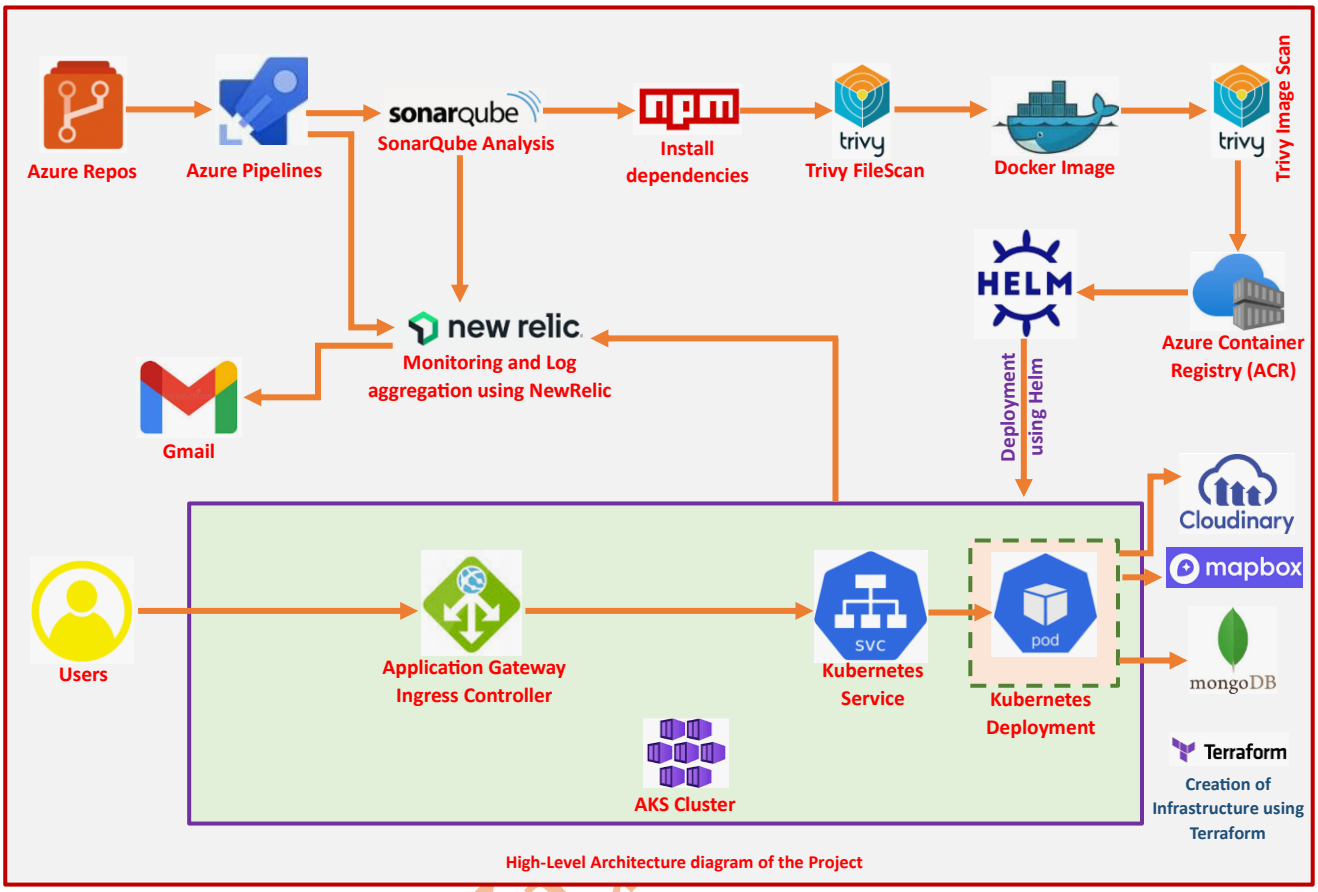
DevOps Project YelpCamp Azure



High-Level Architecture diagram of the Project

This DevOps Project aims to create the infrastructure of AKS Cluster, Azure Container Registry (ACR), Azure DevOps Agent and SonarQube Server with the help of Terraform. Creation of CI/CD pipeline using Azure DevOps for the purpose of Deployment of the Application. For Monitoring and Log Aggregation NewRelic had been used which is shown in the high-level architecture diagram above. Alerts had been created in NewRelic and will send the Email on the Group Email Id if the Alert condition crosses the threshold. In this project Mapbox, Cloudinary and MongoDB had been used. The credentials for Mapbox, Cloudinary and MongoDB had been encrypted using base64 and provided through the kubernetes secrets using helm chart present in the GitHub Repository https://github.com/singhritesh85/helm-chart-yelp-camp.git. Credentials used in this project had been changed later.

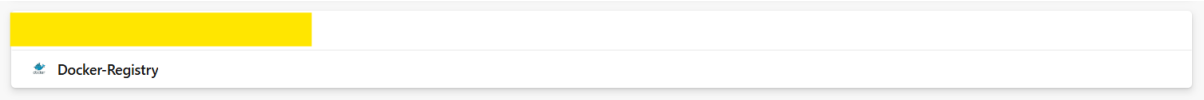
Using Cloudinary I had uploaded the images to the website. Mapbox had been used to locate a location using Maps and MongoDB is the database used for this project.

Agent for NewRelic had been installed in Jenkins Master, Jenkins Slave, SonarQube Server and EKS Cluster. In EKS cluster this Agent was installed using the helm chart which is explained at the later stage in this project.

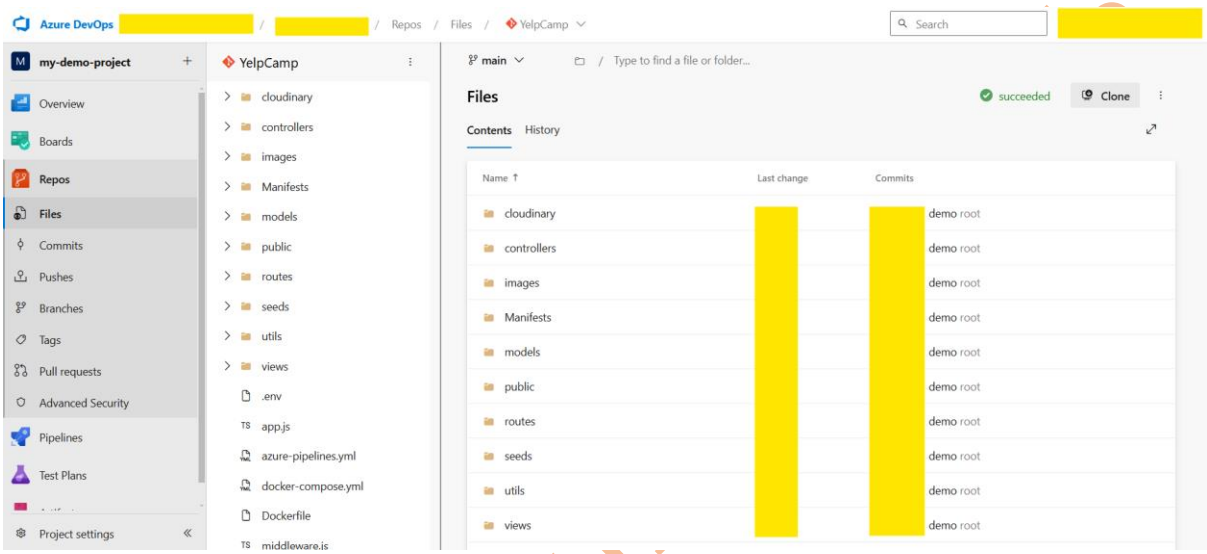
To monitor Jenkins Job using NewRelic you need to install OpenTelemetry Plugin in Jenkins and do the configuration from Jenkins Systems and for monitoring SonarQube the procedure to install the NewRelic Agent has been discussed later in this project.

The Source code for this project is present in the GitHub Repository https://github.com/singhritesh85/YelpCamp.git**.**

Azure Container Registry (ACR) had been Integrated with Azure DevOps Pipeline using the Service Connection as shown in the screenshot attached below.



The Source Code for this project was present in the Azure Repos as shown in the screenshot attached below.



I had provided restricted access to the deployment user **demo** limited to the namespace **yelp-camp**. That means the deployment user demo has all the access in the namespace yelp-camp but did not have overall access for the entire Kubernetes Cluster (AKS Cluster).

To provide the Access to the deployment user demo I had used Kubernetes service account, Kubernetes Role, and Kubernetes RoleBinding. Created Kubernetes Secret and used the Token for kubernetes kubeconfig file which I had shared with the deployment user demo as shown in the screenshot attached below.

```
[root@[redacted]~]# 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@[redacted] ~]# cat sa-role-rolebinding.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: demo
  namespace: yelp-camp
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: user-role
  namespace: yelp-camp
rules:
  - apiGroups: ["*"]
    resources: ["*"]
    verbs: ["*"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: user-rolebinding
  namespace: yelp-camp
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: user-role
subjects:
- namespace: yelp-camp
  kind: ServiceAccount
  name: demo
```

cat sa-role-rolebinding.yaml

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: demo
  namespace: yelp-camp
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: user-role
  namespace: yelp-camp
rules:
  - apiGroups: ["*"]
    resources: ["*"]
    verbs: ["*"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: user-rolebinding
  namespace: yelp-camp
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: user-role
subjects:
- namespace: yelp-camp
  kind: ServiceAccount
  name: demo
```

cat secret.yaml

apiVersion: v1

kind: Secret

type: kubernetes.io/service-account-token

metadata:

name: mysecret

namespace: yelp-camp

annotations:

kubernetes.io/service-account.name: demo

kubectl describe secret mysecret -n yelp-camp

Name: mysecret

Namespace: yelp-camp

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

====

namespace: 9 bytes

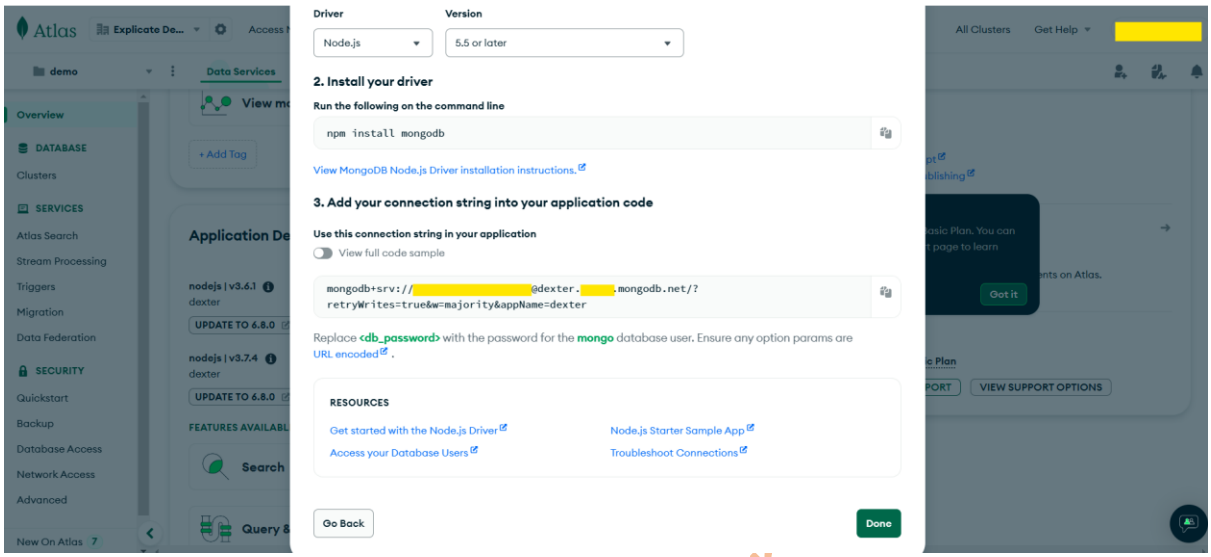
token:

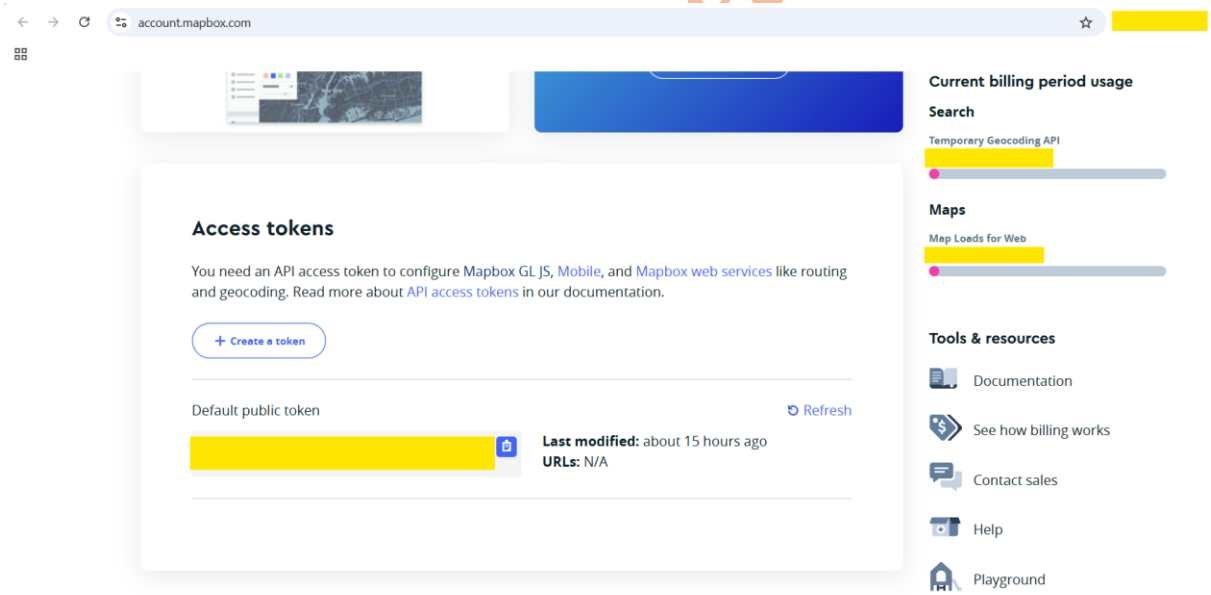
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

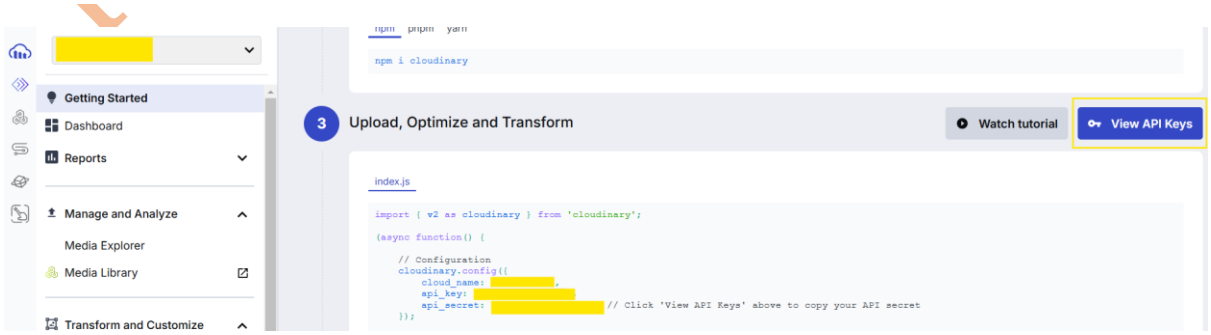
```
[demo@devopsagent-vm ~]$ cat ~/.kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: L[redacted]TV
F[redacted]dJ
S[redacted]VV
V[redacted]Ix
Z[redacted]aC
8[redacted]I2
U[redacted]Un
R[redacted]BJ
d[redacted]Rz
d[redacted]9v
d[redacted]Qz
N[redacted]NT
S[redacted]OG
R[redacted]9J
c[redacted]QV
RFLS0tLS0K
    server: https://yelpcamp-cluster-dns-p[redacted]y.e[redacted]e.privatelink.eastus2.azmk8s.io:443
  name: yelpcamp-cluster
contexts:
- context:
    cluster: yelpcamp-cluster
    user: demo
  name: dexter
current-context: dexter
kind: Config
preferences: {}
users:
- name: demo
  user:
    token: e[redacted]Du
Z[redacted]cn
Z[redacted]Rj
O[redacted]H-
9[redacted]OE
5[redacted]qe
j[redacted]a_
g[redacted]JY
[demo@devopsagent-vm ~]$ kubectl get nodes
Error from server (Forbidden): nodes is forbidden: User "system:serviceaccount:yelp-camp:demo" cannot list resource "nodes" in API group "" at the cluster sco
pe
[demo@devopsagent-vm ~]$ kubectl get pods -n yelp-camp
No resources found in yelp-camp namespace.
[demo@devopsagent-vm ~]$
```

```
cat  ~/.kube/config

apiVersion: v1

clusters:

- cluster:

    certificate-authority-data:
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

    server: https://yelpcamp-cluster-dns-
p3nfyuxy.eXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXe.privatelink.eastus2.azmk8s.io:443

  name: yelpcamp-cluster

contexts:

- context:

    cluster: yelpcamp-cluster

    user: demo

  name: dexter

current-context: dexter

kind: Config

preferences: {}

users:

- name: demo

  user:

    token:
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Below screenshot shows the credentials for MongoDB, Mapbox and Cloudinary. These credentials I had provided in .env file. I had encrypted these credentials using base64 and provided in the helm chart. I kept this helm chart in the GitHub Repo https://github.com/singhritesh85/helm-chart-yelp-camp.git.







The azure-pipelines.yaml file to create the Azure DevOps Pipeline is as shown below.

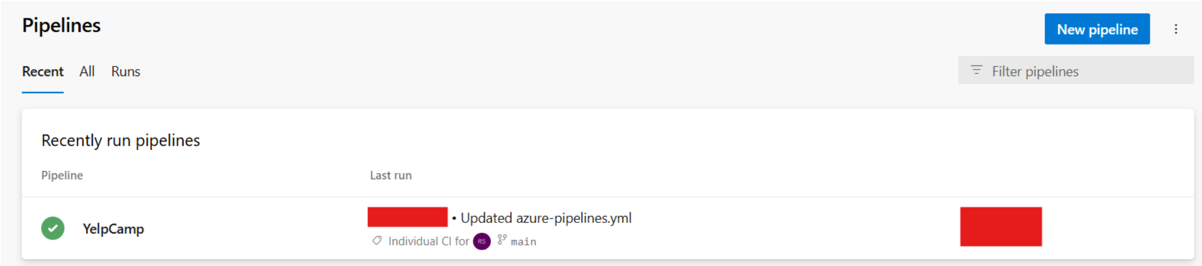
azure-pipelines.yaml

```
trigger:
- main

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

variables:
  imagePullSecret: 'yelpcamp-auth'

stages:
- stage: Build
  displayName: Build
  jobs:
  - job: Build
    displayName: Build
    workspace:
      clean: all
    steps:
    - task: CmdLine@2
      inputs:
       script: 'sonar-scanner -Dsonar.projectKey=yelpcamp -Dsonar.projectName=yelpcamp -
Dsonar.qualitygate.wait=true -Dsonar.host.url=http://XX.XX.X.XXX:9000 -
Dsonar.login=squ_aXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXd'
- stage: "InstallDependencies"
  displayName: InstallDependencies
  dependsOn: Build
  jobs:
  - job: "InstallDependencies"
    displayName: InstallDependencies
    steps:
```

```
    - checkout: none
    - task: Npm@1
      inputs:
        command: 'install'
        workingDir: '.'
    - task: CmdLine@2
      inputs:
        script: 'trivy fs . > /home/demo/trivy-filescan.txt'
- stage: DockerImageBuild
  displayName: DockerImageBuild
  dependsOn: "InstallDependencies"
  jobs:
  - job: DockerImageBuild
    displayName: DockerImageBuild
    steps:
    - checkout: none
    - task: CmdLine@2
      inputs:
        script: |
          docker system prune -f --all
          docker build -t yelpcampcontainer24registry.azurecr.io/samplewebapp:$(Build.BuildId) .
          trivy image --exit-code 0 --severity MEDIUM,HIGH
yelpcampcontainer24registry.azurecr.io/samplewebapp:$(Build.BuildId)
          #trivy image --exit-code 1 --severity CRITICAL
yelpcampcontainer24registry.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: 'yelpcamp-rg'
        kubernetesCluster: 'yelpcamp-cluster'
        namespace: 'yelp-camp'
        command: 'upgrade'
        chartType: 'FilePath'
        chartPath: '/home/demo/helm-chart-yelp-camp/folo'
        releaseName: 'yelp-camp'
        overrideValues: 'imagePullSecrets[0].name=yelpcamp-
auth,image.repository=yelpcampcontainer24registry.azurecr.io/samplewebapp,image.tag=$(Build.Bu
ildId),replicaCount=1,service.type=ClusterIP,service.port=80'
```

I ran the Azure DevOps Pipeline and its screenshot after the execution is shown in the screenshot attached below.



After successful execution of the Azure DevOps Pipeline created Kubernetes Pod, Service, Deployment and Kubernetes Ingress as shown in the screenshot attached below.

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

NAME                     TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/yelp-camp-folo   ClusterIP   [redacted]   <none>        80/TCP    [redacted]

NAME                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/yelp-camp-folo   1/1     1            1           [redacted]

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/yelp-camp-folo-[redacted]  1         1         1       [redacted]
[demo@devopsagent-vm ~]$ kubectl get ing -n yelp-camp
NAME               CLASS                       HOSTS                       ADDRESS      PORTS     AGE
yelpcamp-ingress   azure-application-gateway   yelpcamp.singhritesh85.com  [redacted]   80, 443   [redacted]
```

I had created Kubernetes Secrets before execution of the Azure DevOps Pipeline as shown in the screenshot attached below.

kubectl create secret docker-registry yelpcamp-auth --docker-server=https://yelpcampcontainer24registry.azurecr.io --docker-username=yelpcampcontainer24registry --docker-password=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX **-n yelp-camp**

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

Checked the Kubernetes Secrets using the command as written below.

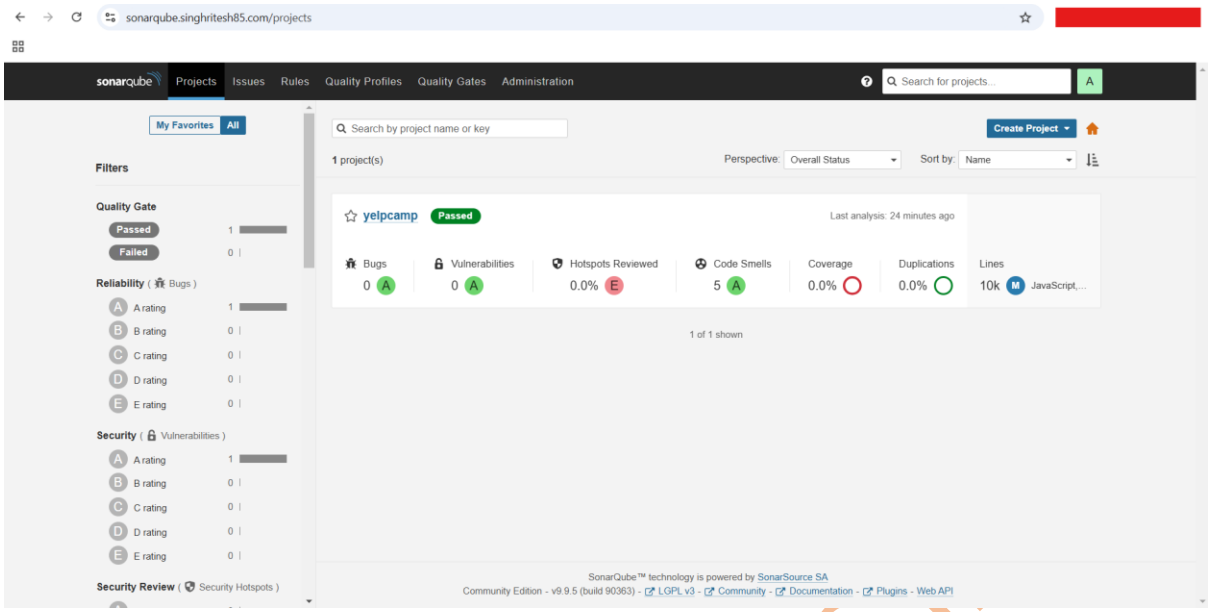
Kubectl get secrets -n yelp-camp

Kubernetes Ingress Rule is as shown in the screenshot attached below.

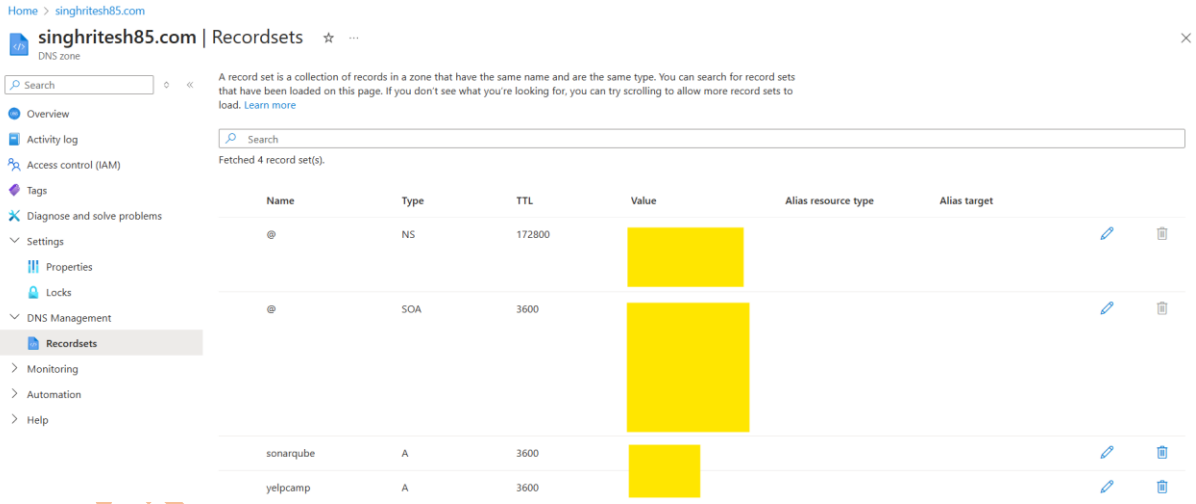
cat ingress-rule.yaml

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

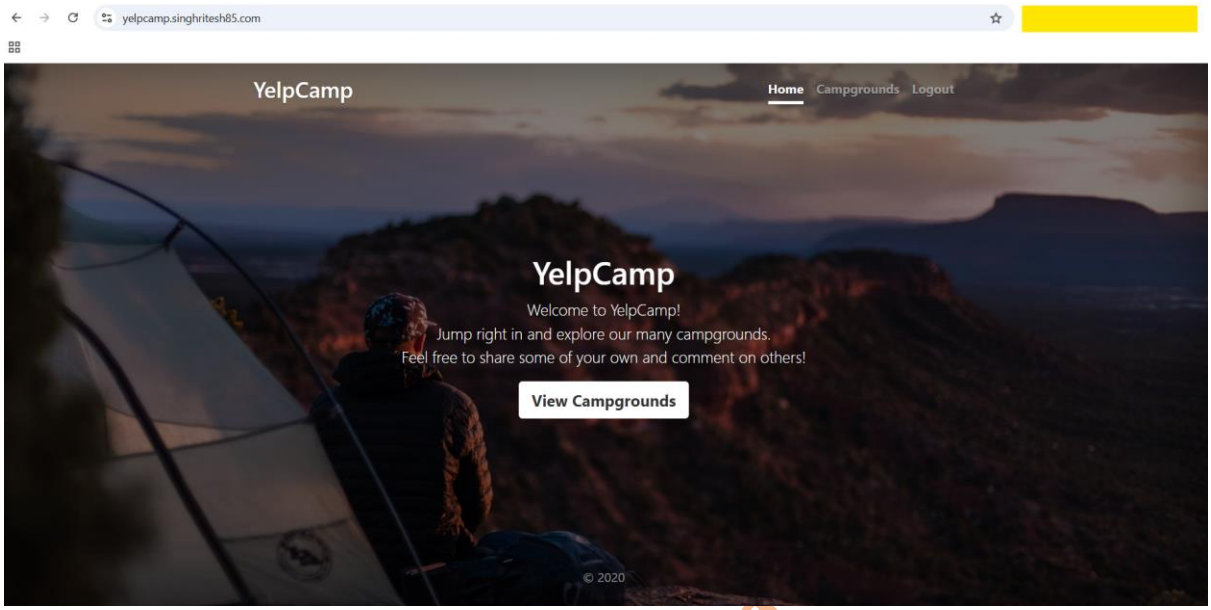
Below screenshot shows the SonarQube project after execution of Azure DevOps Pipeline.

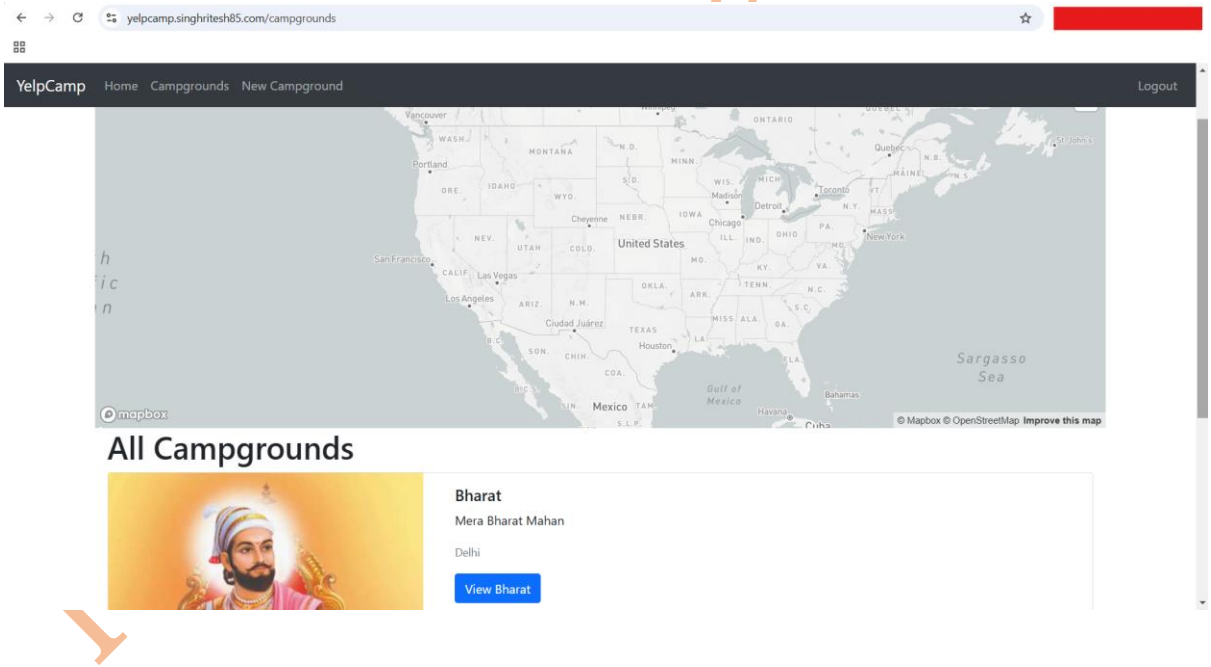


I did the entry for Public IP Address and host of Kubernetes Ingress (shown in the screenshot attached above) in Azure DNS Zone to create the Record Set as shown in the screenshot attached below.

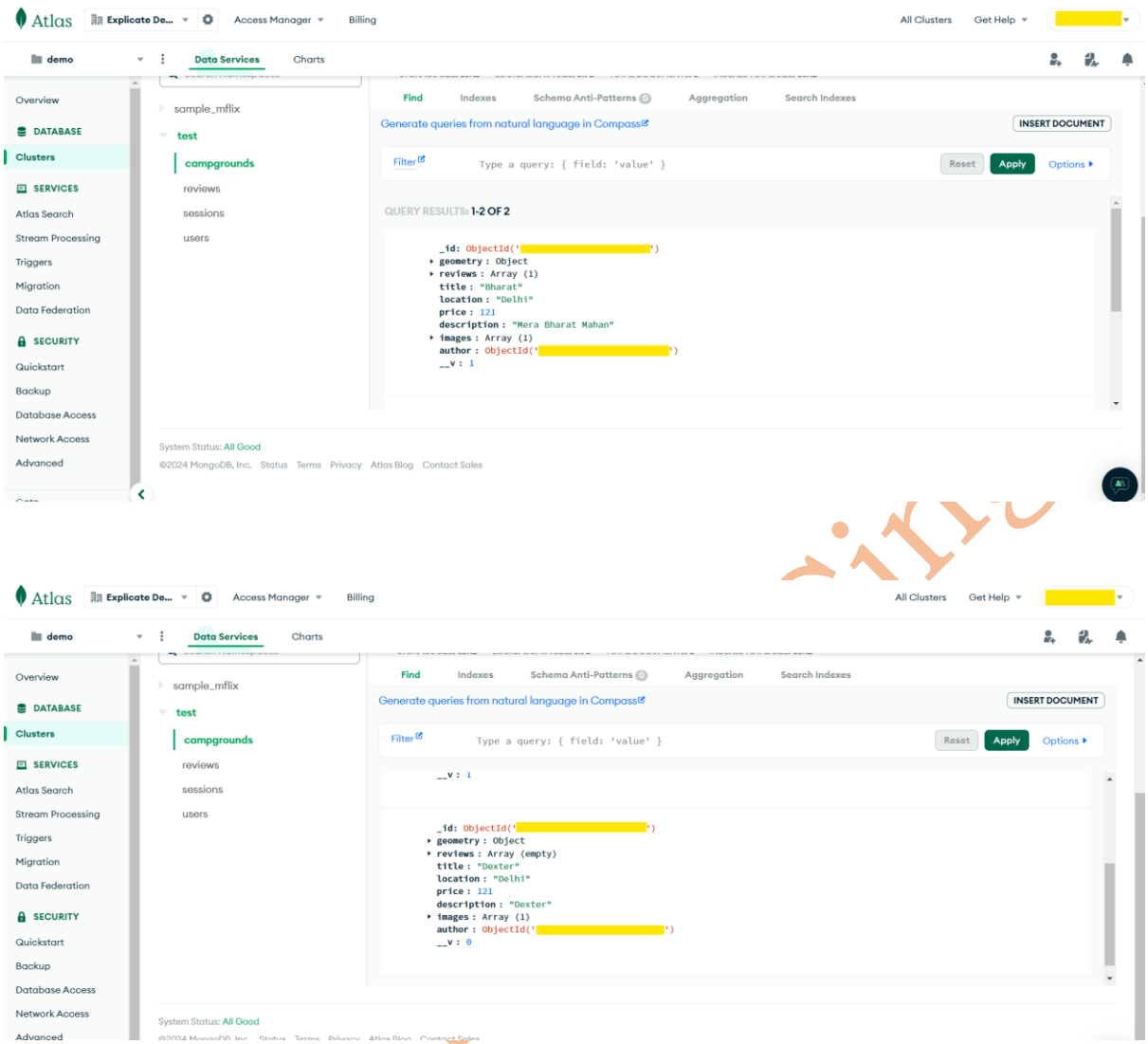


Finally, you can access the Yelp Camp Application using the newly created URL as shown in the screenshot attached below. I had also created campgrounds shown in the screenshot attached below.

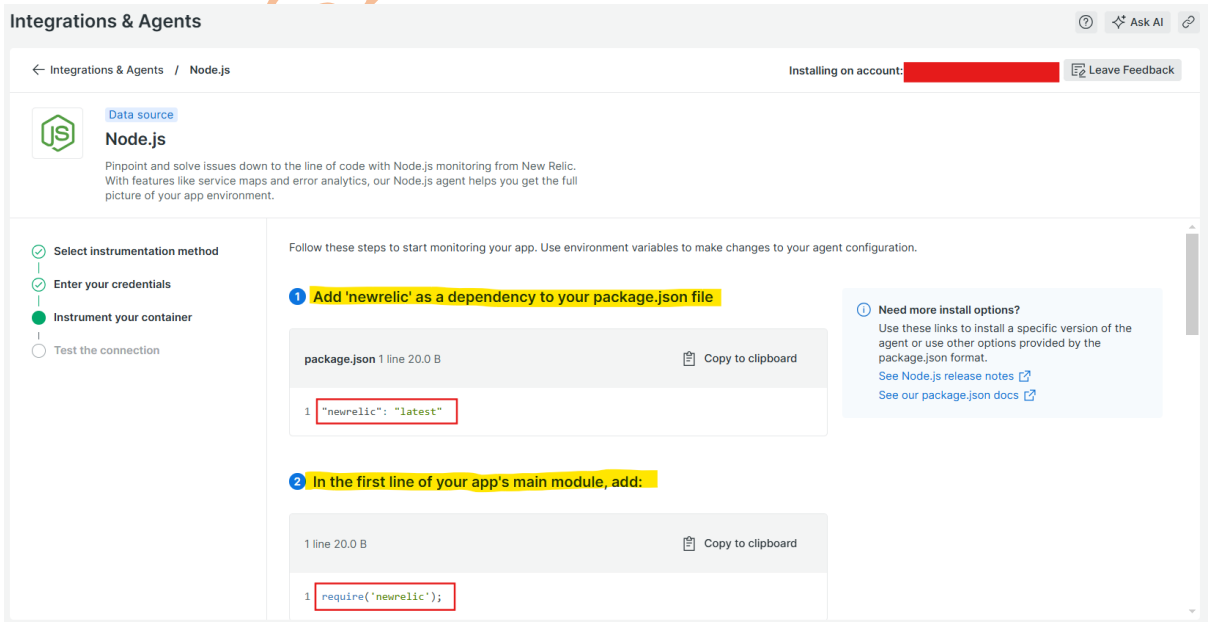


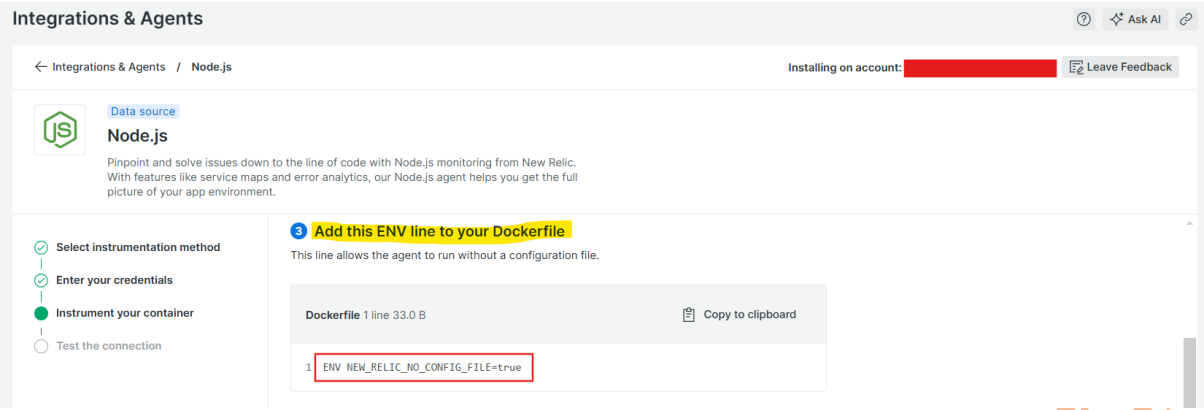


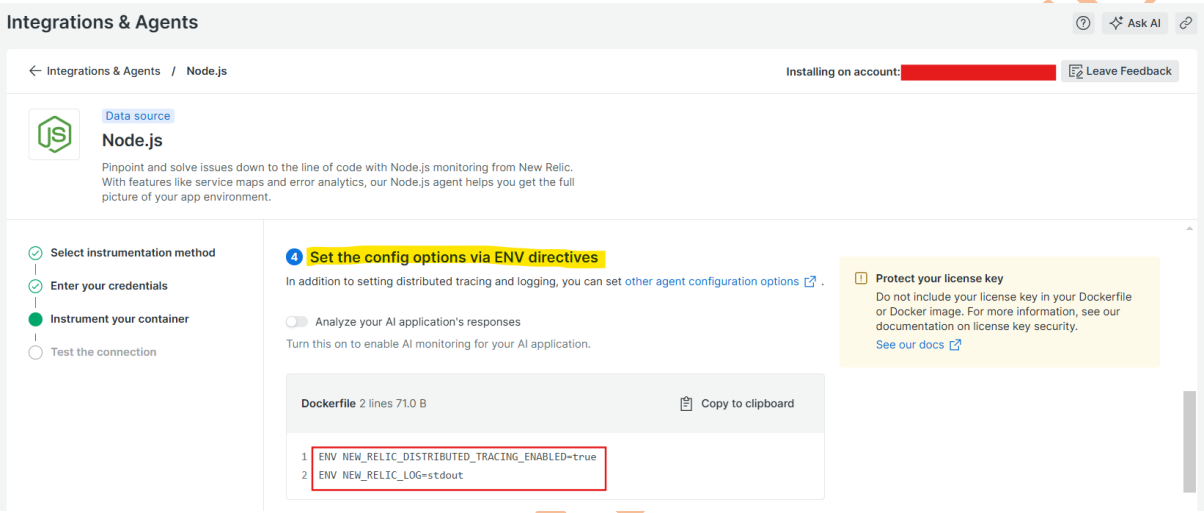
I had created two campgrounds which was verified with the MongoDB entry as shown in the screenshot attached below.



To monitor performance of Yelp-Camp Application I used NewRelic. For **Application Performance Monitoring (APM)**, follow the procedure as written below.



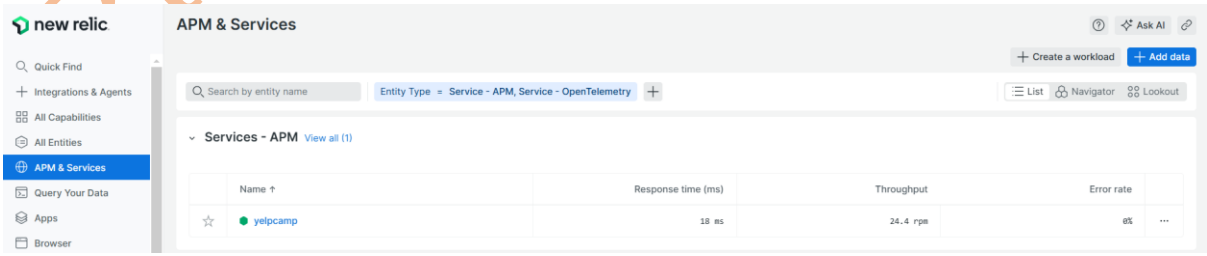


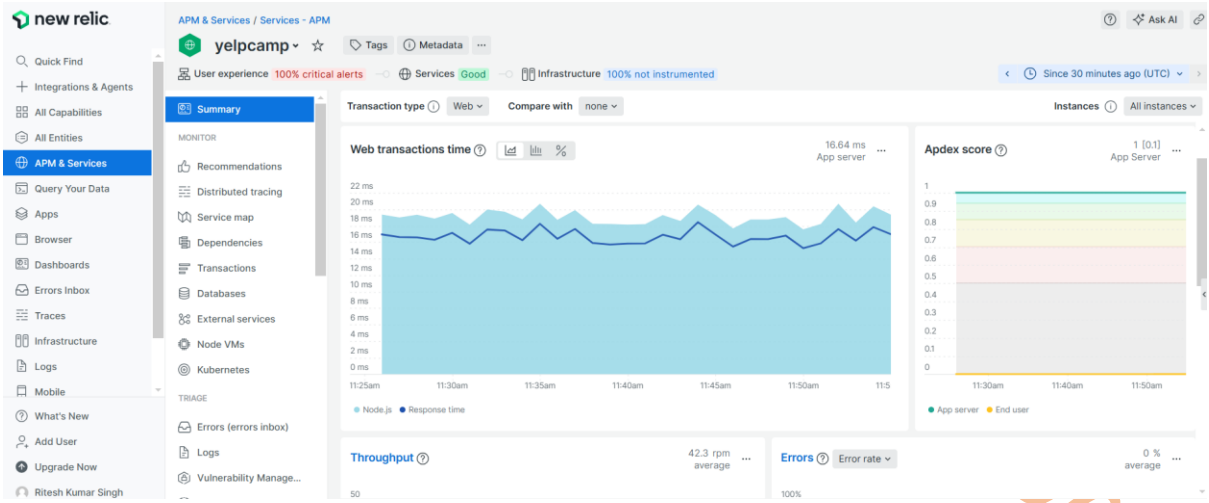


Do the below line entry in your Dockerfile. Finally, ran the Azure DevOps Pipeline.

```
ENV NEW_RELIC_NO_CONFIG_FILE=true
ENV NEW_RELIC_DISTRIBUTED_TRACING_ENABLED=true
ENV NEW_RELIC_LOG=stdout
ENV NEW_RELIC_LICENSE_KEY=c[redacted]L
ENV NEW_RELIC_APP_NAME="yelpcamp"
```

Below screenshot shows the Application Performance Monitoring (APM) for Yelp-Camp Application.





Web transaction time is time taken by web application to respond to a request and send back the response.

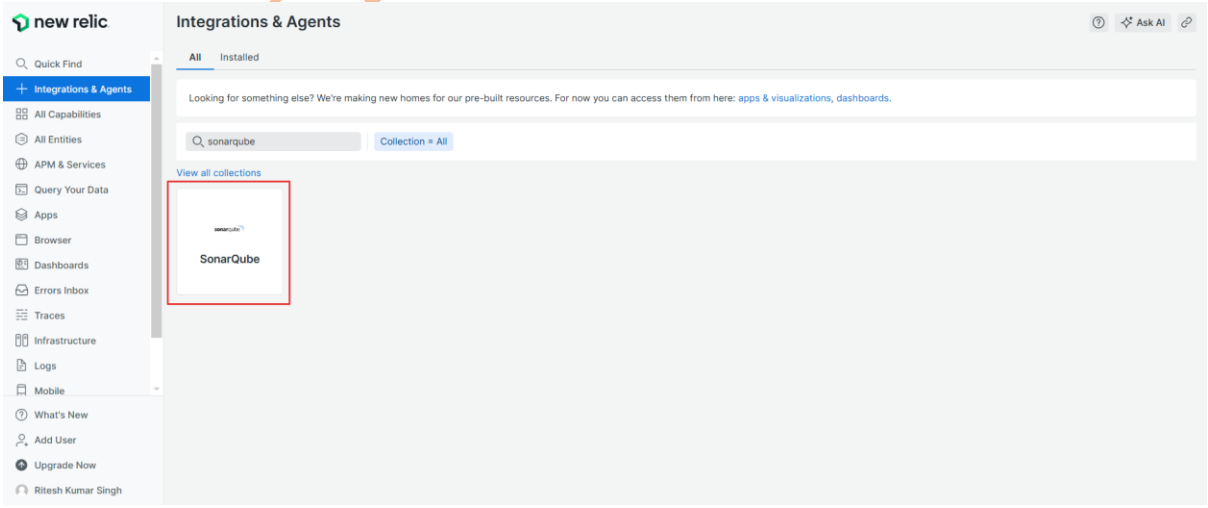
$$\text{Appdex Score} = \frac{\text{Satisfied Request} + (\text{Tolerated Request} / 2)}{\text{Total Number of Request}}$$

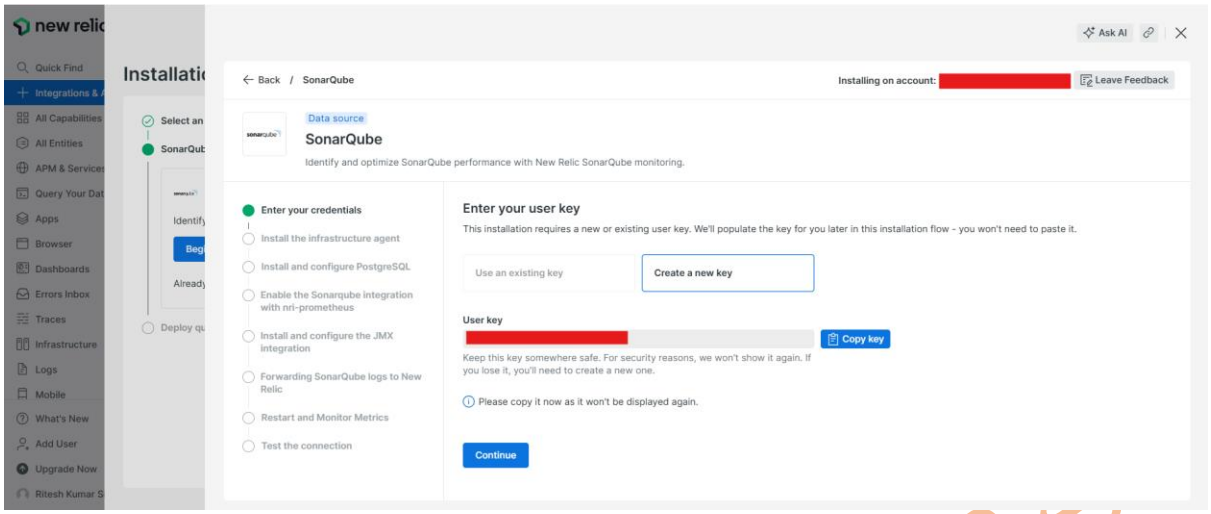
Total Request = Satisfied Request + Tolerated Request + Frustrated Request

It is calculated to measure the user's satisfaction with the web application and its value varies between 0 to 1. 0 is the worst score and 1 is the best score.

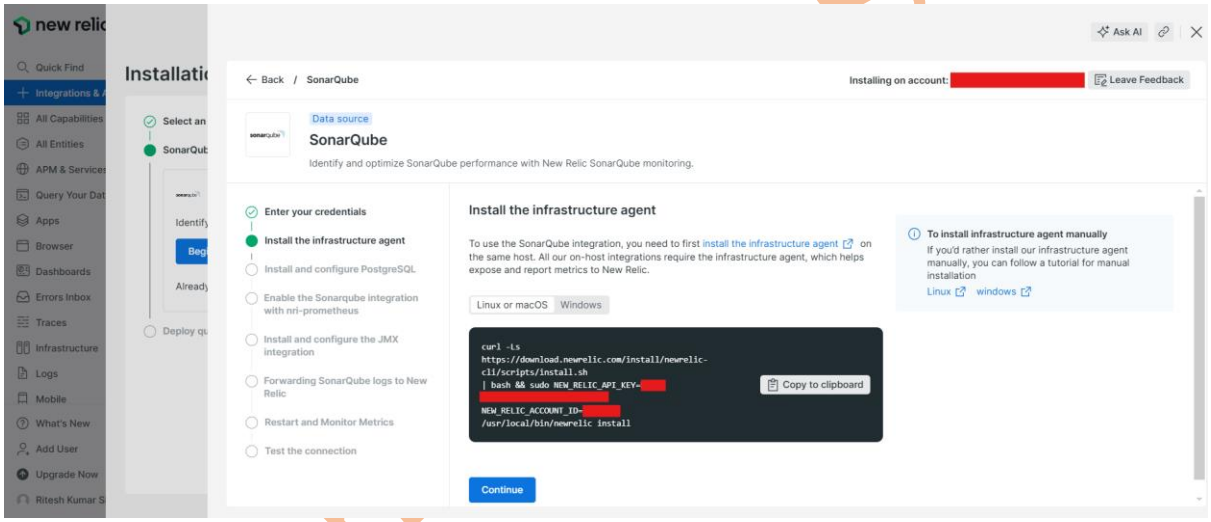
Monitoring SonarQube with NewRelic

To Monitor SonarQube with NewRelic install the Agent of SonarQube for NewRelic as shown in the screenshot attached below.





Install the NewRelic Agent on SonarQube Server as shown in the screenshot attached below.



I ran the above command on SonarQube Server as shown in the screenshot attached below.

Provide the necessary details as shown in the screenshot attached below and create a file at the path /etc/newrelic-infra/integrations.d/nri-prometheus-config.yml

cat /etc/newrelic-infra/integrations.d/nri-prometheus-config.yml

```
integrations:
  - name: nri-prometheus
    config:
      # When standalone is set to false nri-prometheus requires an infrastructure agent to work and send data. Defaults to true
      standalone: false
      # When running with infrastructure agent emitters will have to include infra-sdk
      emitters: infra-sdk
      # The name of your cluster. It's important to match other New Relic products to relate the data.
      cluster_name: "sonarqube"
      targets:
        - description: Sonarqube metrics list
          urls: ["http://aXmXn:XXXXXXXX@X.XXX.XXX.XXX:9000/api/monitoring/metrics"]
      #     tls_config:
      #     ca_file_path: "/etc/etcd/etcd-client-ca.crt"
      #     cert_file_path: "/etc/etcd/etcd-client.crt"
      #      key_file_path: "/etc/etcd/etcd-client.key"
      # Whether the integration should run in verbose mode or not. Defaults to false
      verbose: false
      # Whether the integration should run in audit mode or not. Defaults to false.
      # Audit mode logs the uncompressed data sent to New Relic. Use this to log all data sent.
      # It does not include verbose mode. This can lead to a high log volume, use with care
      audit: false
      # The HTTP client timeout when fetching data from endpoints. Defaults to 30s.
      # scrape_timeout: "30s"
      # Length in time to distribute the scraping from the endpoints
      scrape_duration: "5s"
      # Number of worker threads used for scraping targets.
      # For large clusters with many (&gt;400) endpoints, slowly increase until scrape
      # time falls between the desired `scrape_duration`.
      # Increasing this value too much will result in huge memory consumption if too
      # many metrics are being scraped.
      # Default: 4
      # worker_threads: 4
      # Whether the integration should skip TLS verification or not. Defaults to false
      insecure_skip_verify: true
    timeout: 10s
```

```
[root@ip-172-31-18-254 opt]# cat /etc/newrelic-infra/integrations.d/nri-prometheus-config.yml
integrations:
  - name: nri-prometheus
    config:
      # When standalone is set to false nri-prometheus requires an infrastructure agent to work and send data. Defaults to true
      standalone: false
      # When running with infrastructure agent emitters will have to include infra-sdk
      emitters: infra-sdk
      # The name of your cluster. It's important to match other New Relic products to relate the data.
      cluster_name: "sonarqube"
      targets:
        - description: Sonarqube metrics list
          urls: ["http://[redacted]:9000/api/monitoring/metrics"]
      #     tls_config:
      #     ca_file_path: "/etc/etcd/etcd-client-ca.crt"
      #     cert_file_path: "/etc/etcd/etcd-client.crt"
      #      key_file_path: "/etc/etcd/etcd-client.key"
      # Whether the integration should run in verbose mode or not. Defaults to false
      verbose: false
      # Whether the integration should run in audit mode or not. Defaults to false.
      # Audit mode logs the uncompressed data sent to New Relic. Use this to log all data sent.
      # It does not include verbose mode. This can lead to a high log volume, use with care
      audit: false
      # The HTTP client timeout when fetching data from endpoints. Defaults to 30s.
      # scrape_timeout: "30s"
      # Length in time to distribute the scraping from the endpoints
      scrape_duration: "5s"
      # Number of worker threads used for scraping targets.
      # For large clusters with many (&gt;400) endpoints, slowly increase until scrape
      # time falls between the desired `scrape_duration`.
      # Increasing this value too much will result in huge memory consumption if too
      # many metrics are being scraped.
      # Default: 4
      # worker_threads: 4
      # Whether the integration should skip TLS verification or not. Defaults to false
      insecure_skip_verify: true
    timeout: 10s
```

To configure logging edit the file as shown in the screenshot attached below present at the path

/etc/newrelic-infra/logging.d/logging.yml

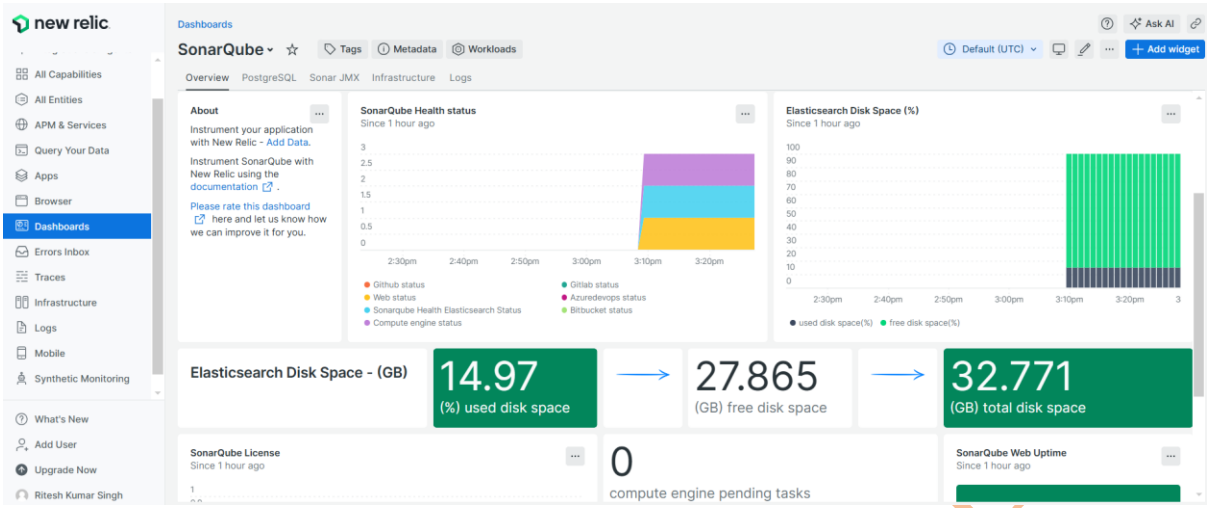
```
logs:
  - name: cloud-init.log
    file: /var/log/cloud-init.log
    attributes:
      logtype: linux_cloud-init
  - name: messages
    file: /var/log/messages
    attributes:
      logtype: linux_messages
  - name: secure
    file: /var/log/secure
    attributes:
      logtype: linux_secure
  - name: yum.log
    file: /var/log/yum.log
    attributes:
      logtype: linux_yum
  - name: newrelic-cli.log
    file: /root/.newrelic/newrelic-cli.log
    attributes:
      newrelic-cli: true
      logtype: newrelic-cli
  - name: sonar_logs
    file: /opt/sonarqube/logs/sonar.log
    attributes:
      logtype: sonar_logs
  - name: ce_logs
    file: /opt/sonarqube/logs/ce.log
    attributes:
      logtype: sonar_ce_logs
  - name: es_logs
    file: /opt/sonarqube/logs/es.log
    attributes:
      logtype: sonar_es_logs
  - name: web_logs
    file: /opt/sonarqube/logs/web.log
    attributes:
      logtype: sonar_web_logs
```

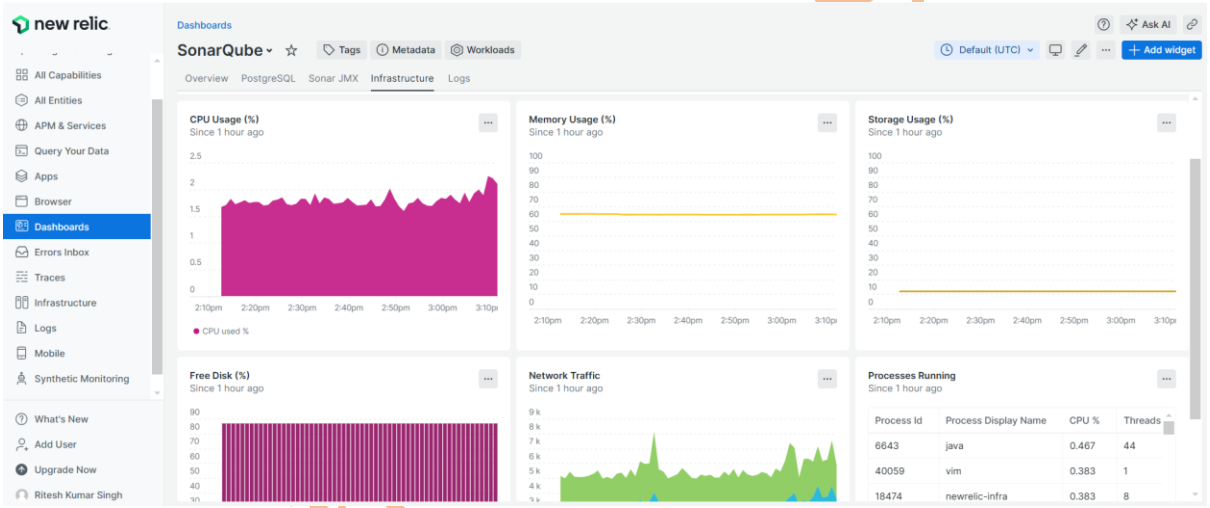
```
[root@[redacted]pt]# cat /etc/newrelic-infra/logging.d/logging.yml
logs:
  - name: cloud-init.log
    file: /var/log/cloud-init.log
    attributes:
      logtype: linux_cloud-init
  - name: messages
    file: /var/log/messages
    attributes:
      logtype: linux_messages
  - name: secure
    file: /var/log/secure
    attributes:
      logtype: linux_secure
  - name: yum.log
    file: /var/log/yum.log
    attributes:
      logtype: linux_yum
  - name: newrelic-cli.log
    file: /root/.newrelic/newrelic-cli.log
    attributes:
      newrelic-cli: true
      logtype: newrelic-cli
  - name: sonar_logs
    file: /opt/sonarqube/logs/sonar.log
    attributes:
      logtype: sonar_logs
  - name: ce_logs
    file: /opt/sonarqube/logs/ce.log
    attributes:
      logtype: sonar_ce_logs
  - name: es_logs
    file: /opt/sonarqube/logs/es.log
    attributes:
      logtype: sonar_es_logs
  - name: web_logs
    file: /opt/sonarqube/logs/web.log
    attributes:
      logtype: sonar_web_logs
```

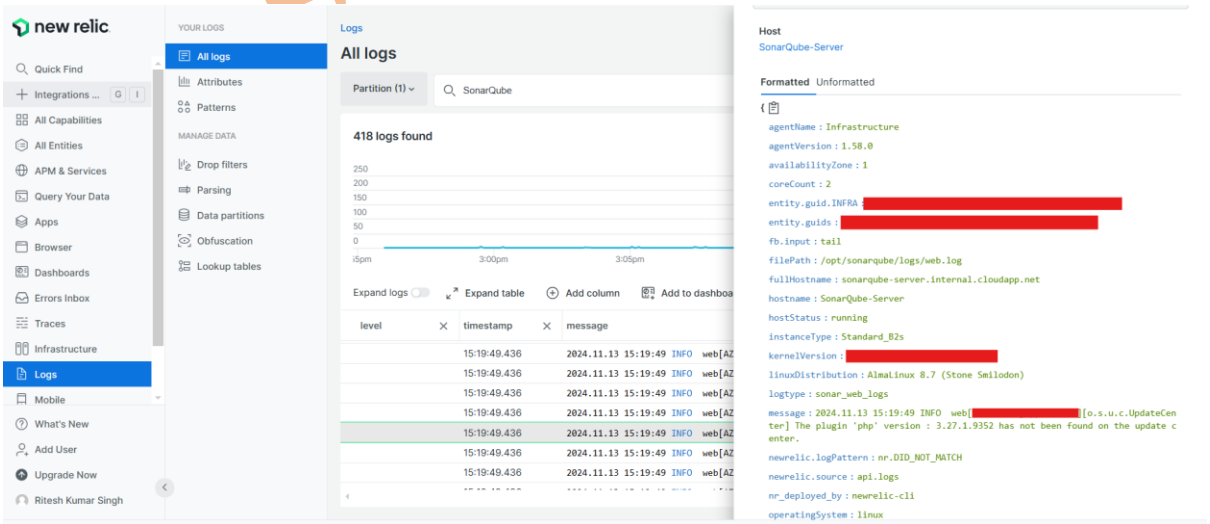
After the above-mentioned configuration restarted the NewRelic service and checked its status as shown in the screenshot attached below.

```
[root@[redacted] opt]# sudo systemctl restart newrelic-infra
[root@[redacted] opt]# sudo systemctl status newrelic-infra
● newrelic-infra.service - New Relic Infrastructure Agent
   Loaded: loaded (/etc/systemd/system/newrelic-infra.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon [redacted] ago
```

Finally, you can start monitor SonarQube as shown in the screenshot attached below.

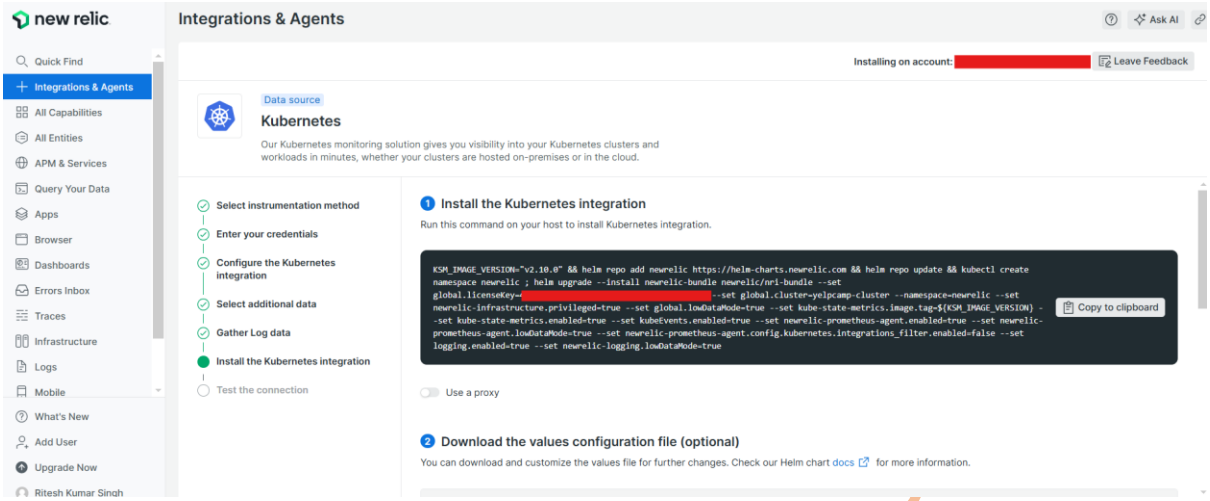






Monitoring AKS Cluster using NewRelic

To monitor AKS Cluster using NewRelic you need to install the NewRelic Agent of AKS Cluster using Helm as shown in the screenshot attached below.



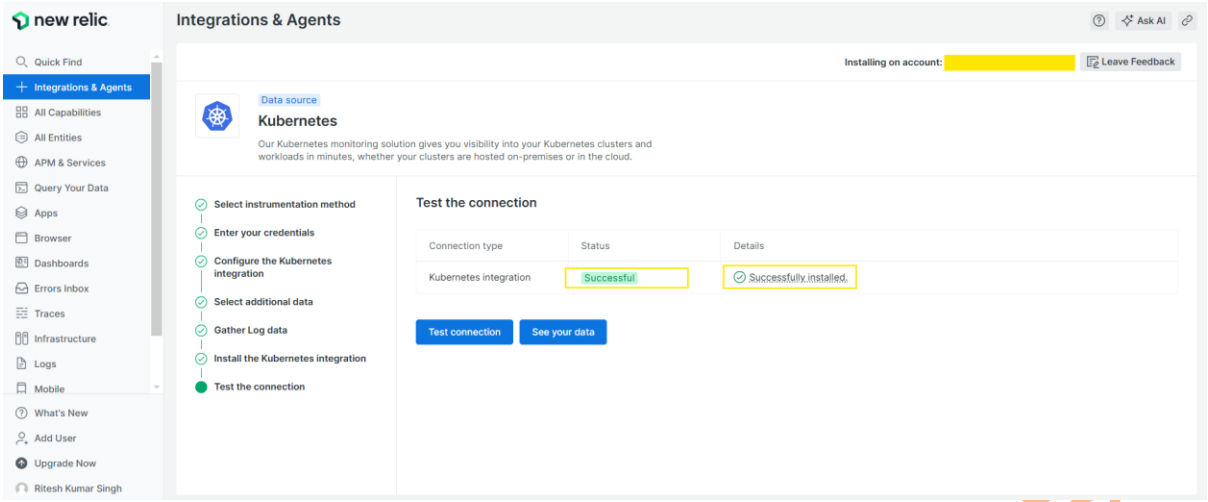
I ran the above command form a Linux machine where I kept the kubeconfig file and kubectl and helm was installed.

```
[root@ ~]# KSM_IMAGE_VERSION="v2.10.0" && helm repo add newrelic https://helm-charts.newrelic.com && helm repo update && kubectl create namespace newrelic ; helm upgrade --install newrelic-bundle newrelic/nri-bundle --set global.licenseKey= --set global.cluster=yelpcamp-cluster --namespace=newrelic --set newrelic-infrastructure.privileged=true --set global.lowDataMode=true --set kube-state-metrics.image.tag=${KSM_IMAGE_VERSION} --set kube-state-metrics.enabled=true --set kubeEvents.enabled=true --set newrelic-prometheus-agent.enabled=true --set newrelic-prometheus-agent.lowDataMode=true --set newrelic-prometheus-agent.config.kubernetes.integrations_filter.enabled=false --set logging.enabled=true --set newrelic-logging.lowDataMode=true
```

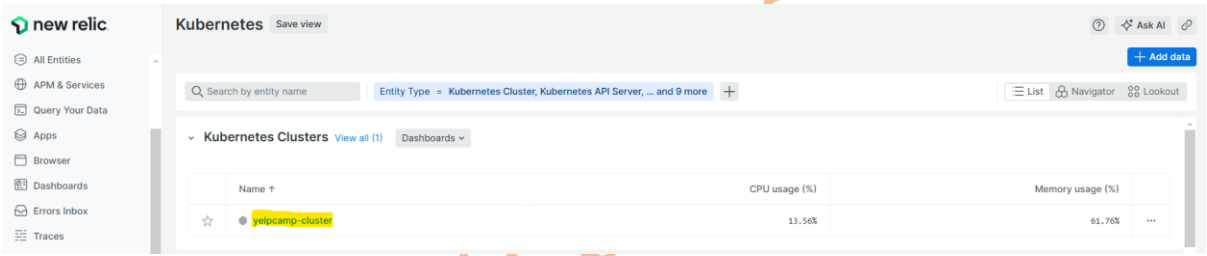
I checked the pods in the namespace **newrelic** after running the above command and found that all the pods are in the **Running** state as shown in the screenshot attached below.

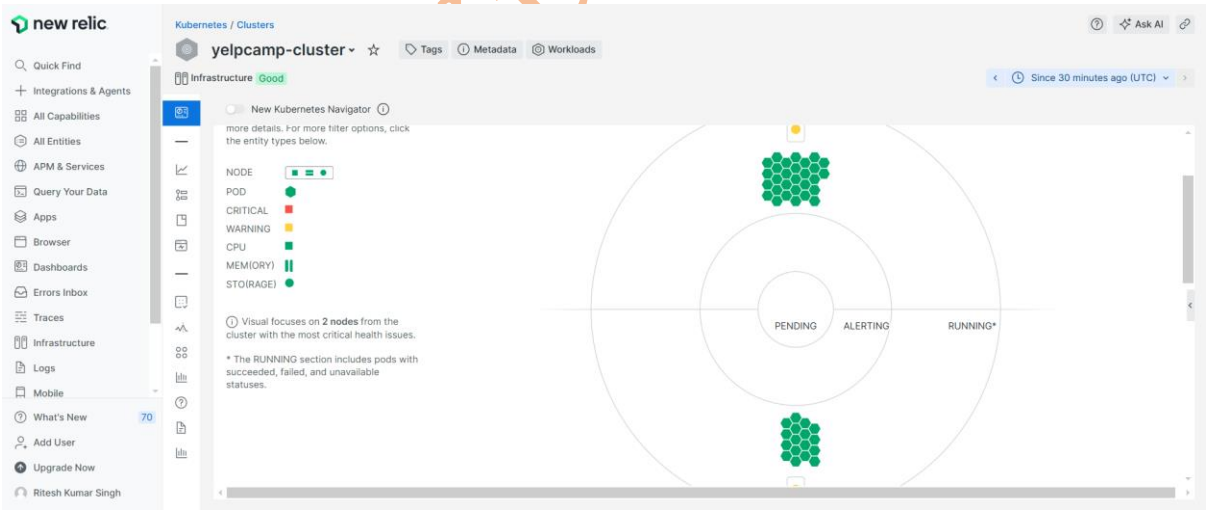
```
[root@ ~]# kubectl get pods -n newrelic
NAME                                                  READY   STATUS    RESTARTS   AGE
newrelic-bundle-kube-state-metrics-                   1/1     Running   0          61s
newrelic-bundle-newrelic-logging-                     1/1     Running   0          61s
newrelic-bundle-newrelic-logging-                     1/1     Running   0          61s
newrelic-bundle-newrelic-prometheus-agent-0           1/1     Running   0          61s
newrelic-bundle-nri-kube-events-                      2/2     Running   0          61s
newrelic-bundle-nri-metadata-injection-               1/1     Running   0          61s
newrelic-bundle-nrk8s-                                2/2     Running   0          61s
newrelic-bundle-nrk8s-kubelet-                        2/2     Running   0          61s
newrelic-bundle-nrk8s-kubelet-                        2/2     Running   0          61s
[root@ ~]#
```

Finally, I tested the connection for Kubernetes Integration and found it was Successful as shown in the screenshot attached below.



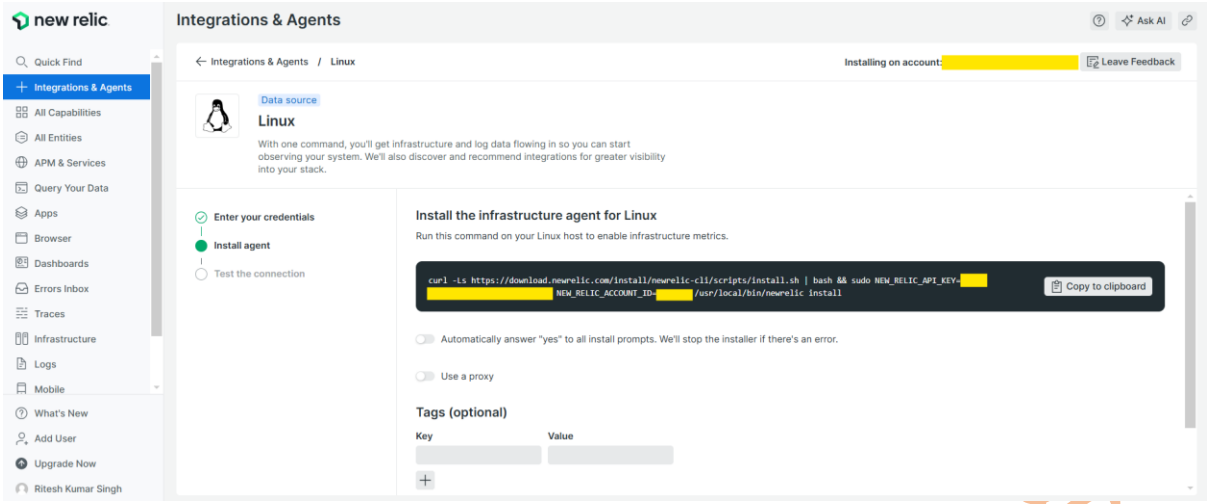
Finally, I checked the NewRelic console for Kubernetes and found my AKS Cluster name there as shown in the screenshot attached below. You can monitor the details of kubernetes cluster and can create the Alerts depending on your need.



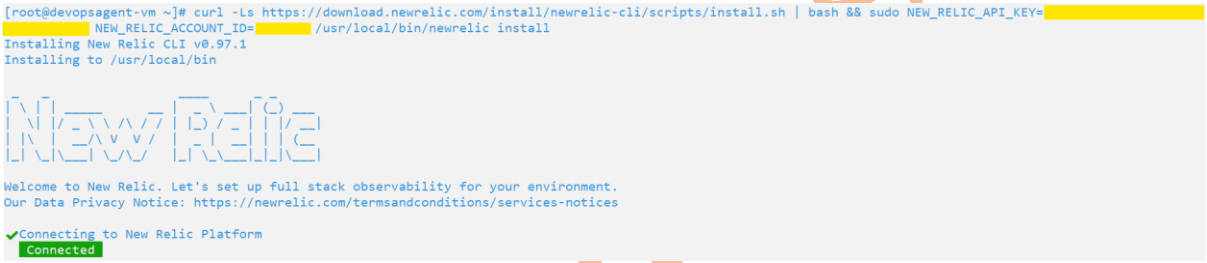


Monitoring Azure DevOps Agent using NewRelic

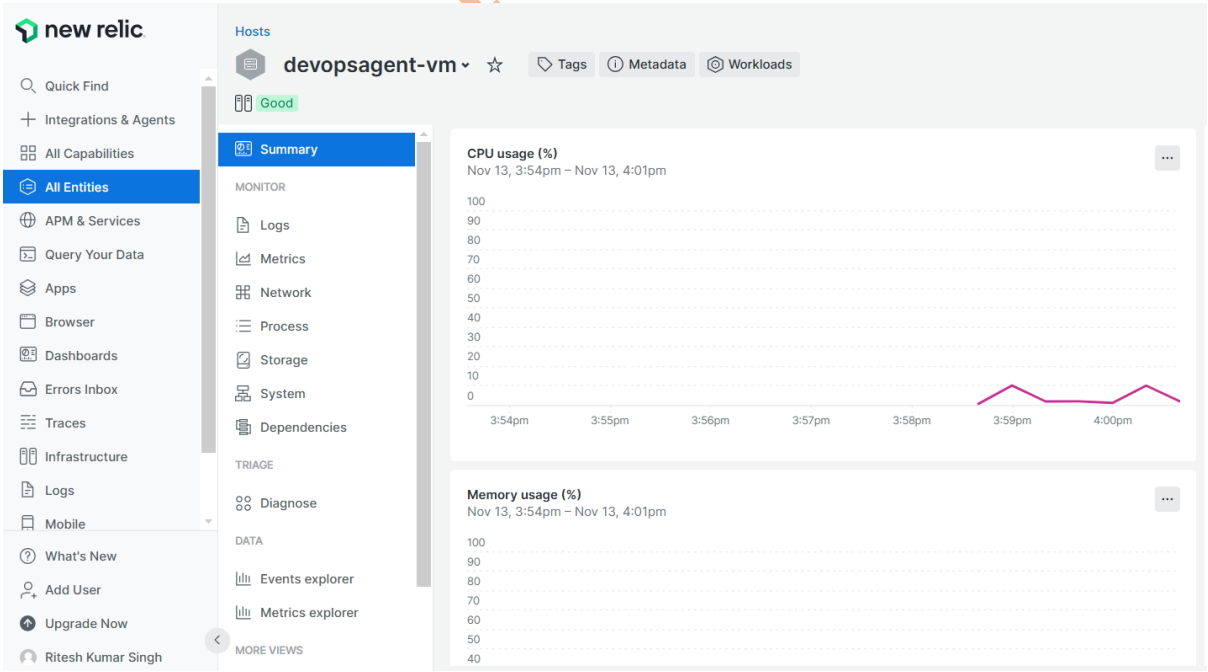
To monitor Azure DevOps Agent using NewRelic I need to install the NewRelic Agent on Azure DevOps Agent Server. To install the NewRelic Agent I ran the below command on Azure DevOps Agent Server as shown in the screenshot attached below.

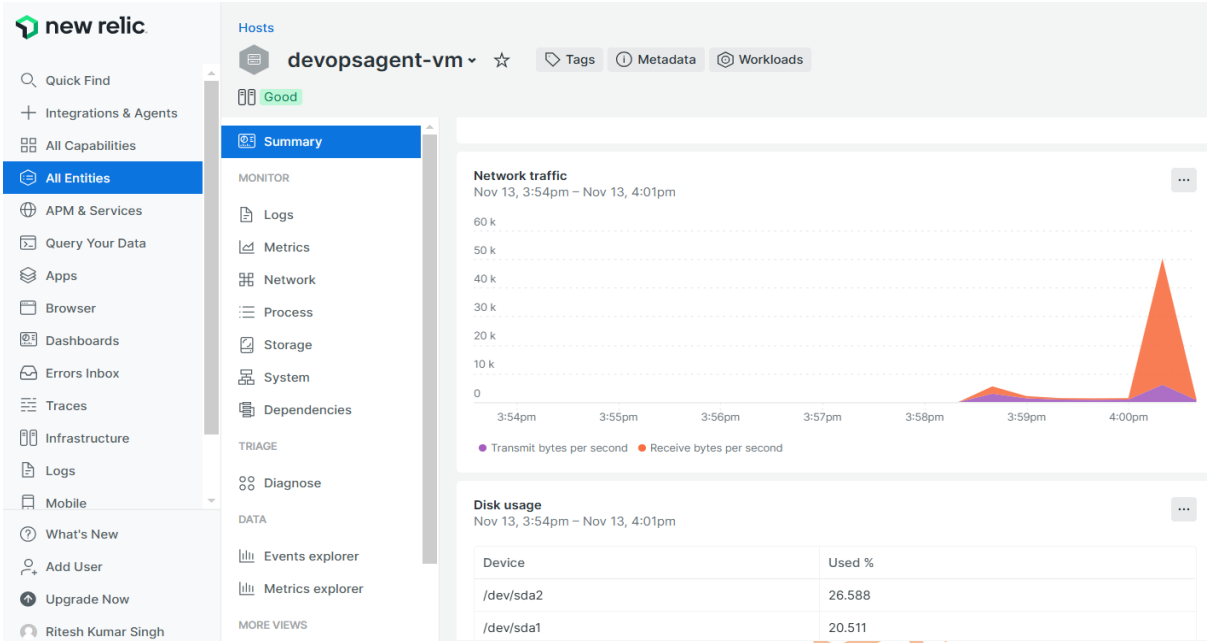


I ran the above command on Azure DevOps Agent Server as shown in the screenshot attached below.



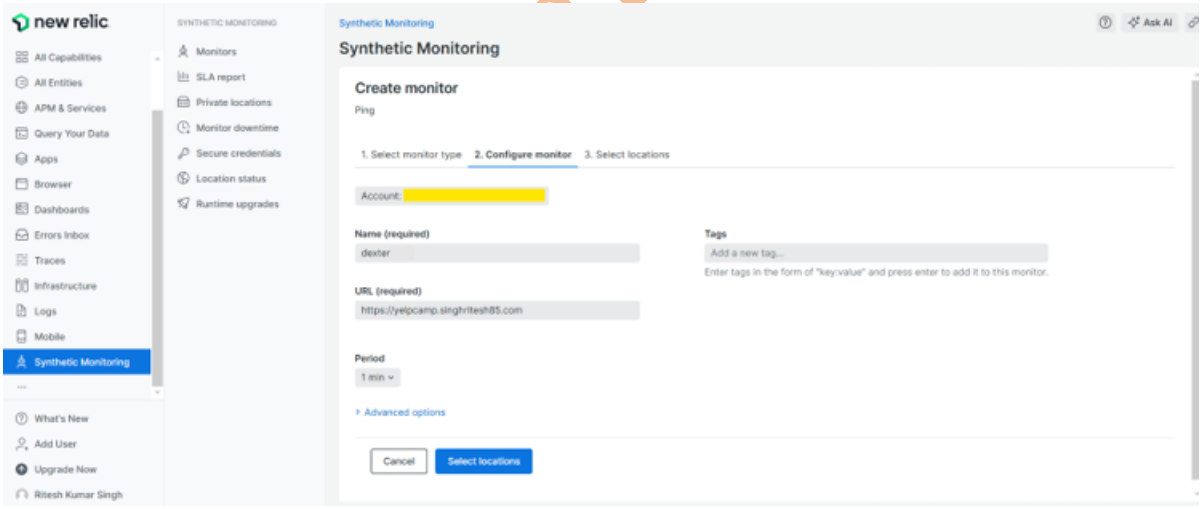
Finally, I checked Infrastructure > Hosts and corresponding name of the Azure DevOps Agent Server and started monitoring the Azure DevOps Agent Server as shown in the screenshot attached below.

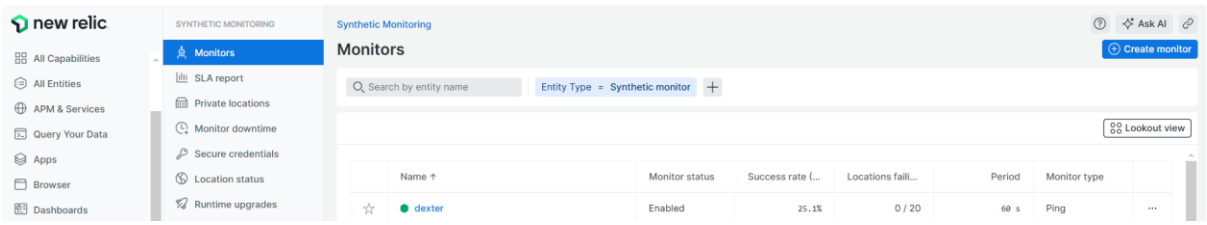


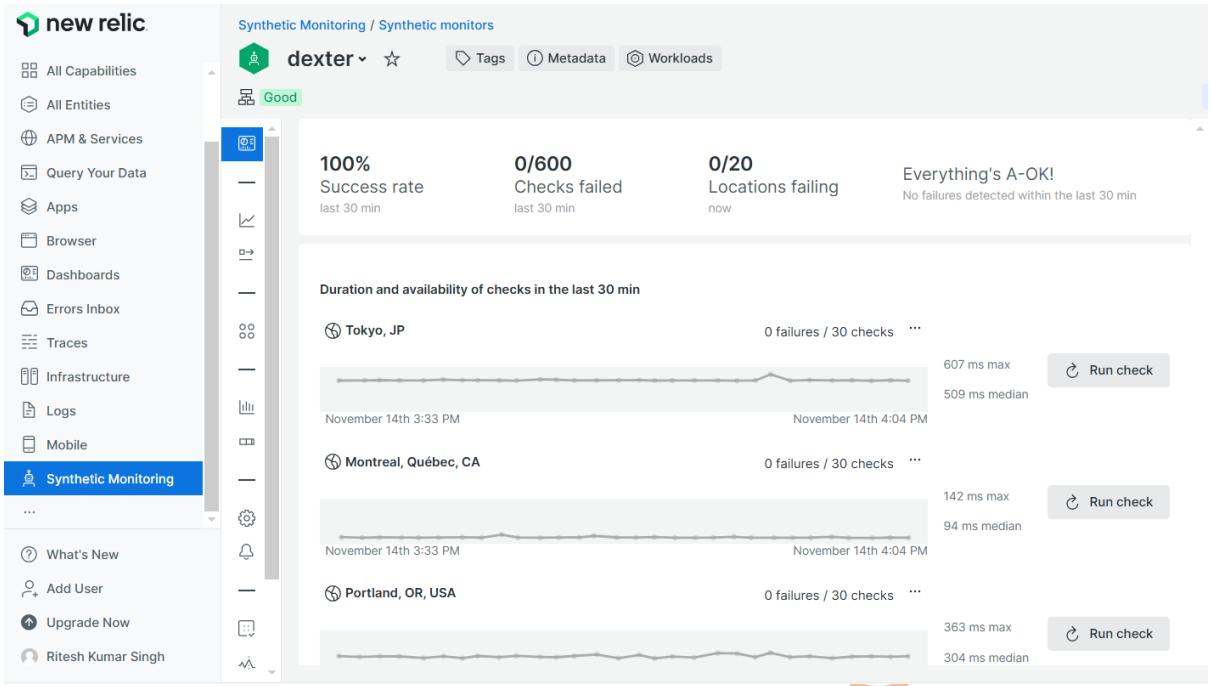


Synthetic Monitoring to Monitor the Application URL

I am performing synthetic monitoring on Application URL https://yelpcamp.singhritesh85.com. On a periodic Interval of every 1 min, it will perform the Synthetic Monitoring from All the public Locations. The below screenshot shows the result of Synthetic Monitoring.



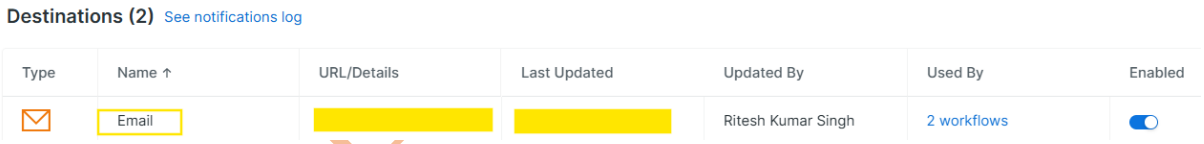




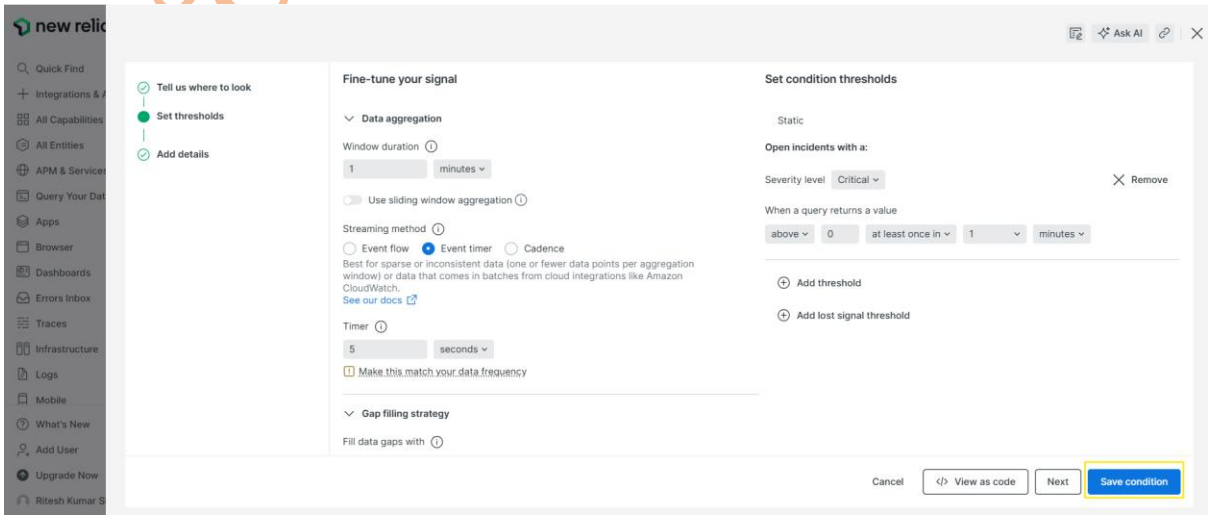
Alerts in NewRelic

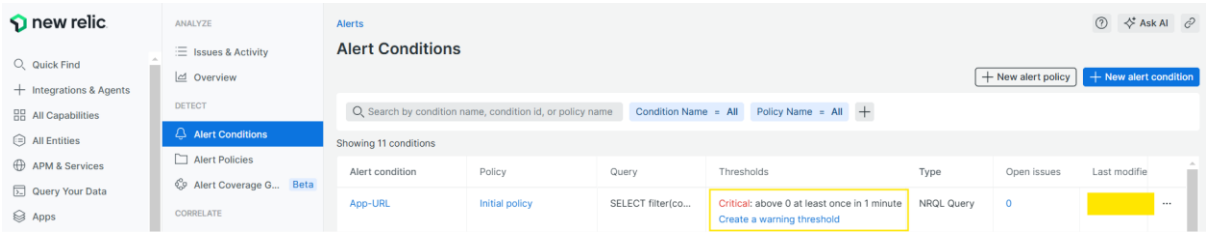
For Alerts in NewRelic we need to discuss about **Destination**, **Alert Condition**, **Alert Policy**, and **Workflows**.

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

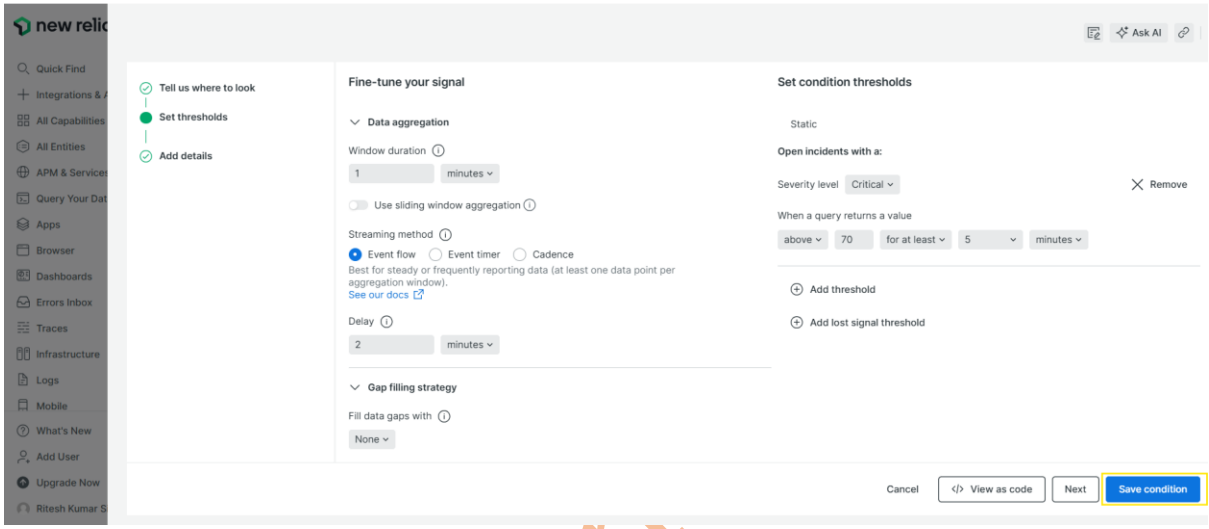


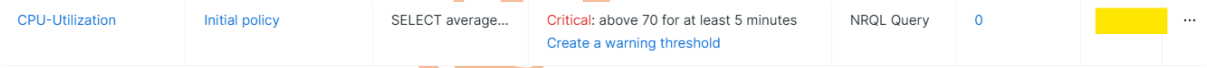
An alert condition had been created for Synthetic Monitoring as shown in the screenshot attached below.



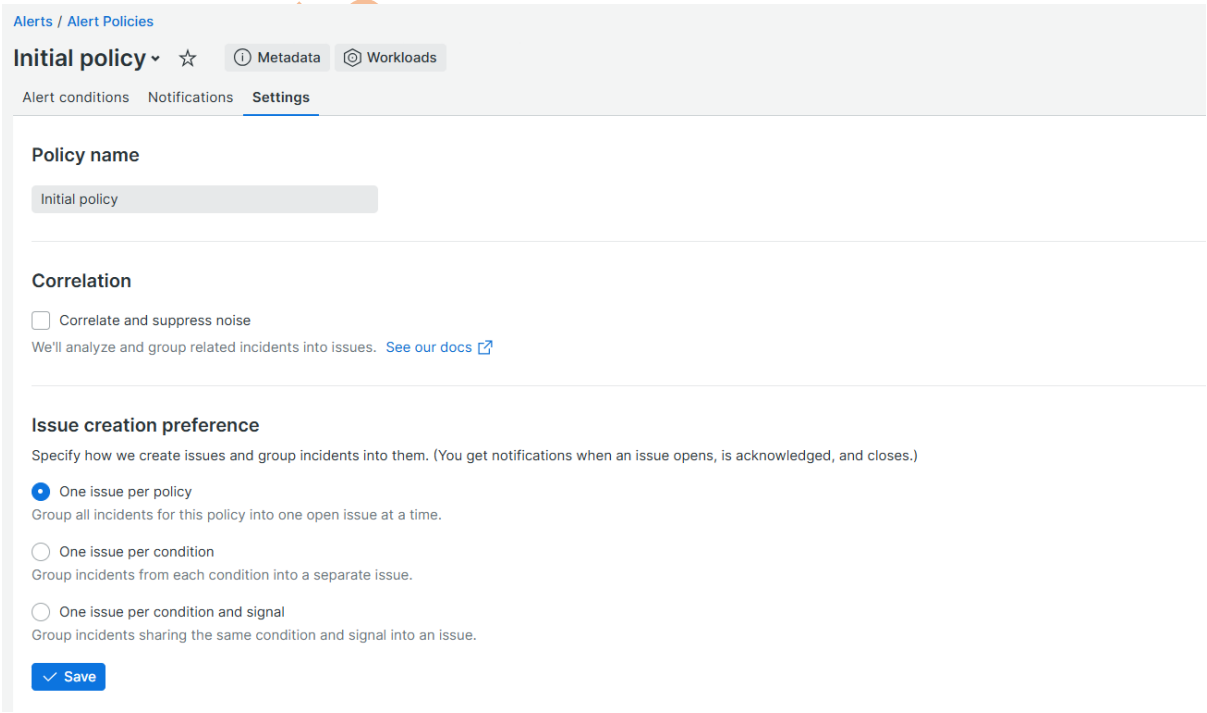


An Alert condition for CPU Utilization had been created as shown in the screenshot attached below.

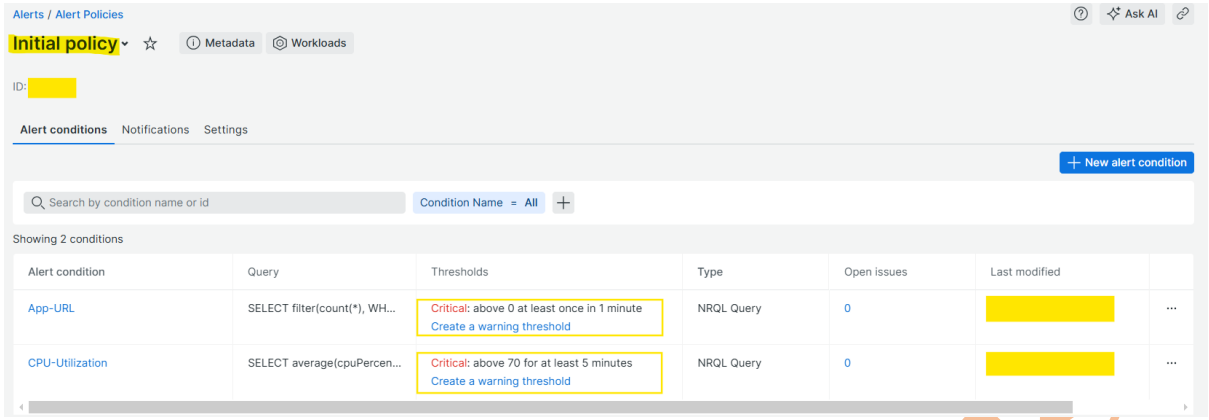




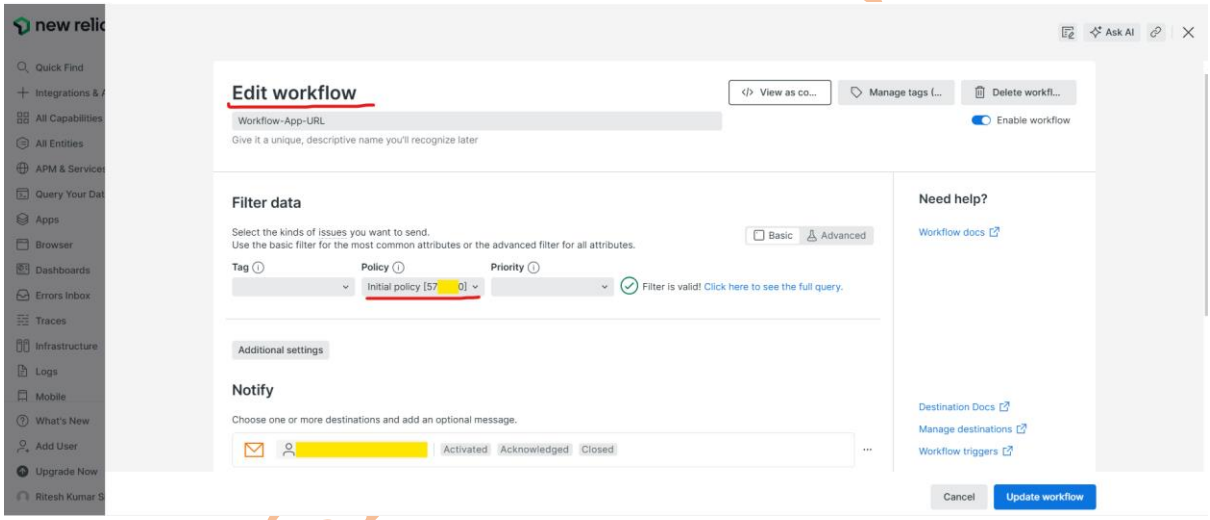
An Alert Policy had been created as shown in the screenshot attached below.



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

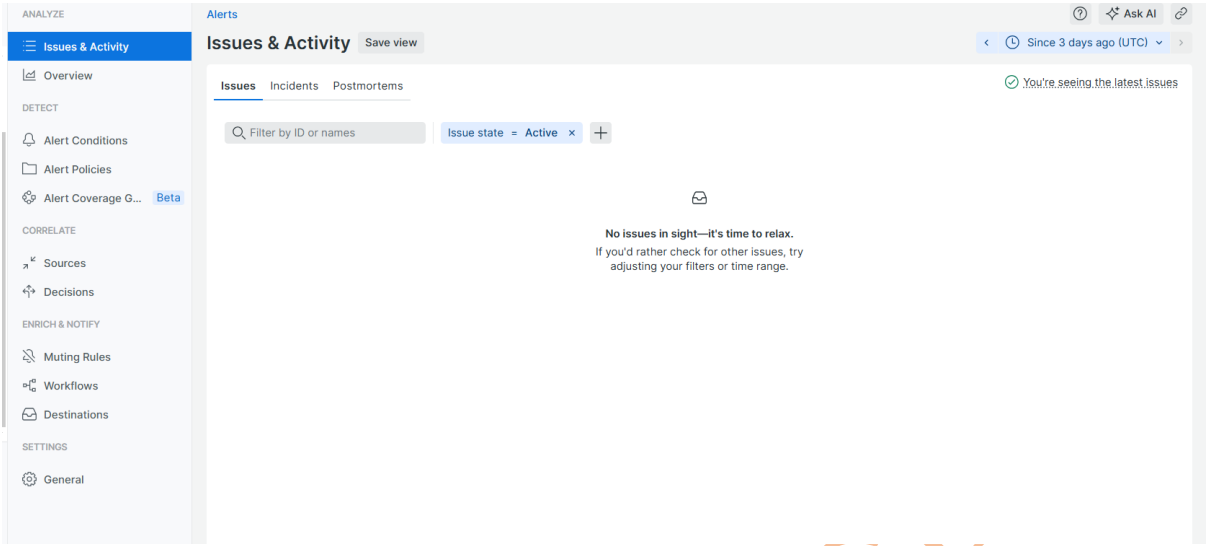


The Workflow had been shown in the screenshot attached below which showed with this Alert Workflow an Alert Policy, and an Email Destination had been attached.

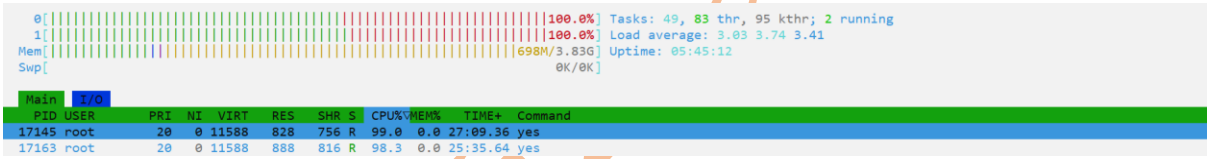


At present there is no Alert, but for the demonstration purpose I had created I had created Alert by increasing the CPU Utilization more than 70% and I had deleted the Application URL https://yelpcamp.singhritesh85.com entry from the Azure DNS Zone Record set for which I had created Synthetic Monitoring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

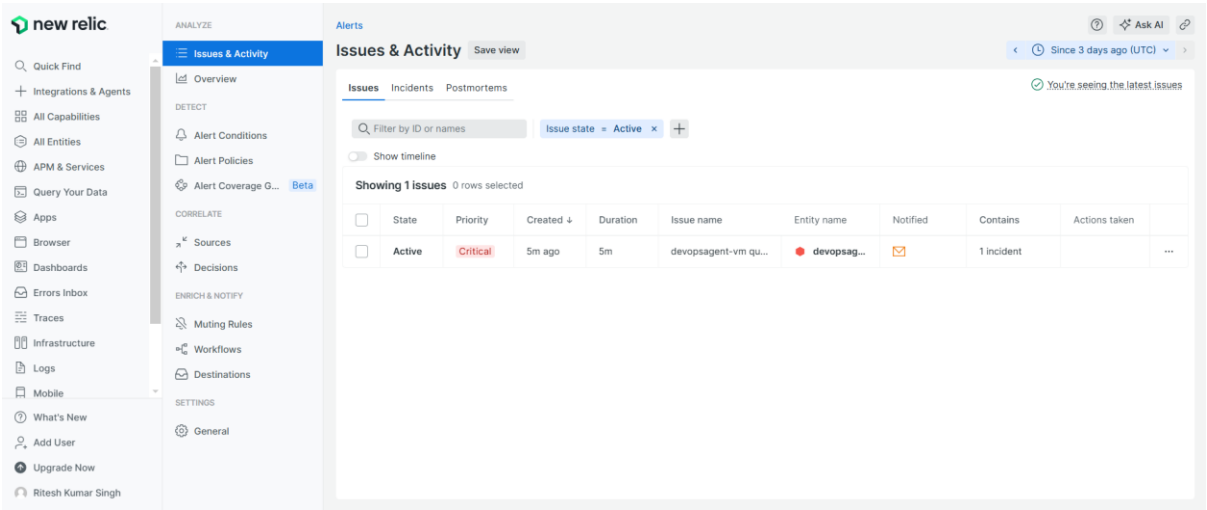
Below screenshot showed no Active alerts now.



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.

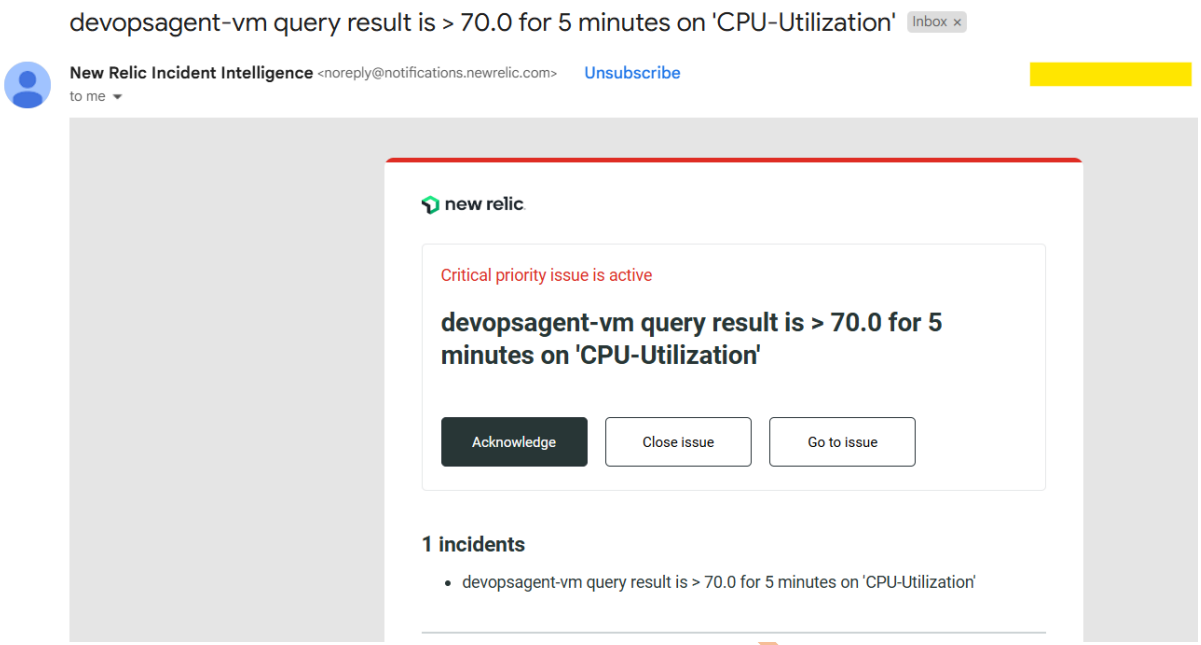


Now the Alert is in Active condition as shown in the screenshot attached below.

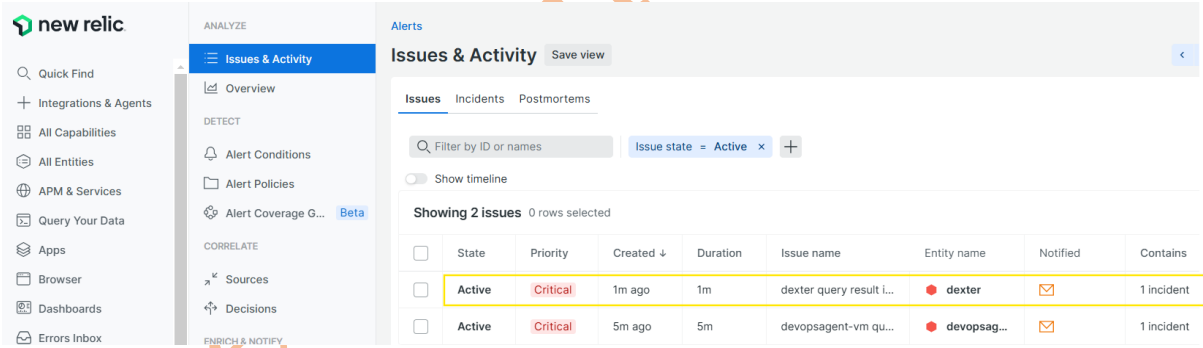


An Email had been triggered to the Group Email ID as shown in the screenshot attached below. After the team received this Email on their Email Id, they will investigate its RCA (Root Cause Analysis) and will check the CPU usage using the command **htop**. Team will check whether any unnecessary crontab,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

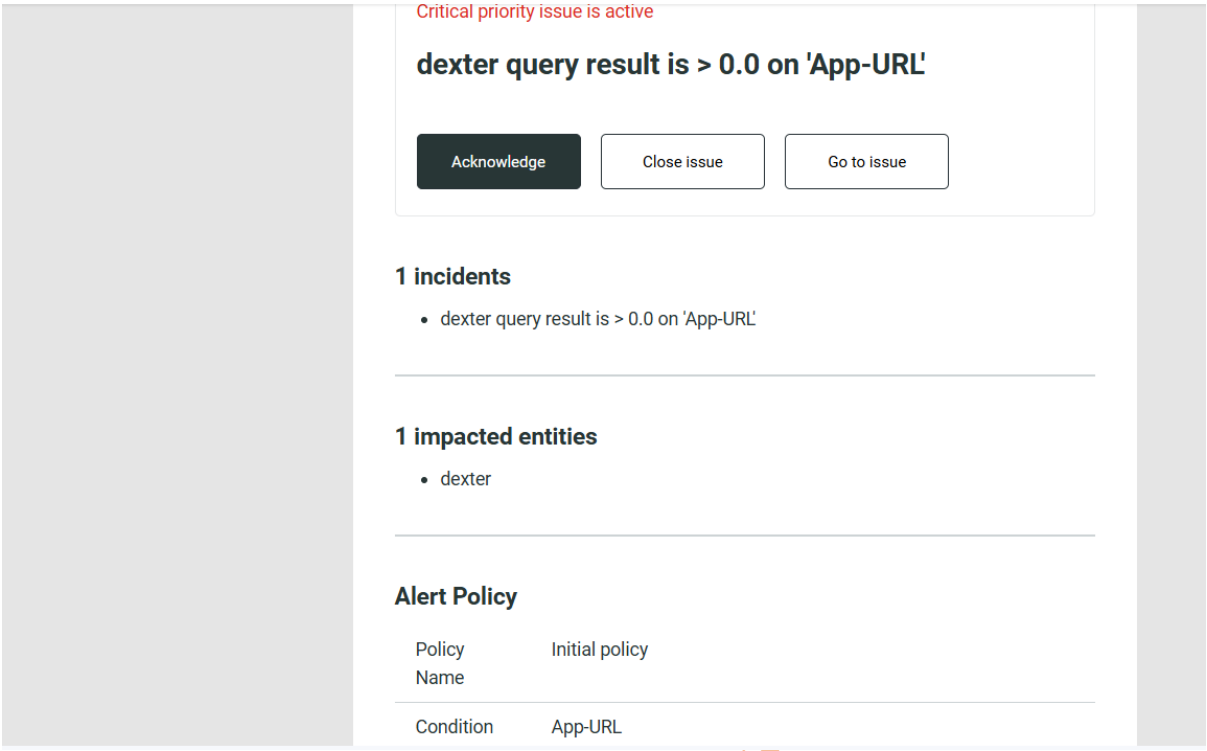


The Alert for Synthetic Monitoring is also in Active condition as shown in the screenshot attached below.



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 who had initiated that Event.

.



Reference: - https://github.com/Colt/YelpCamp.git