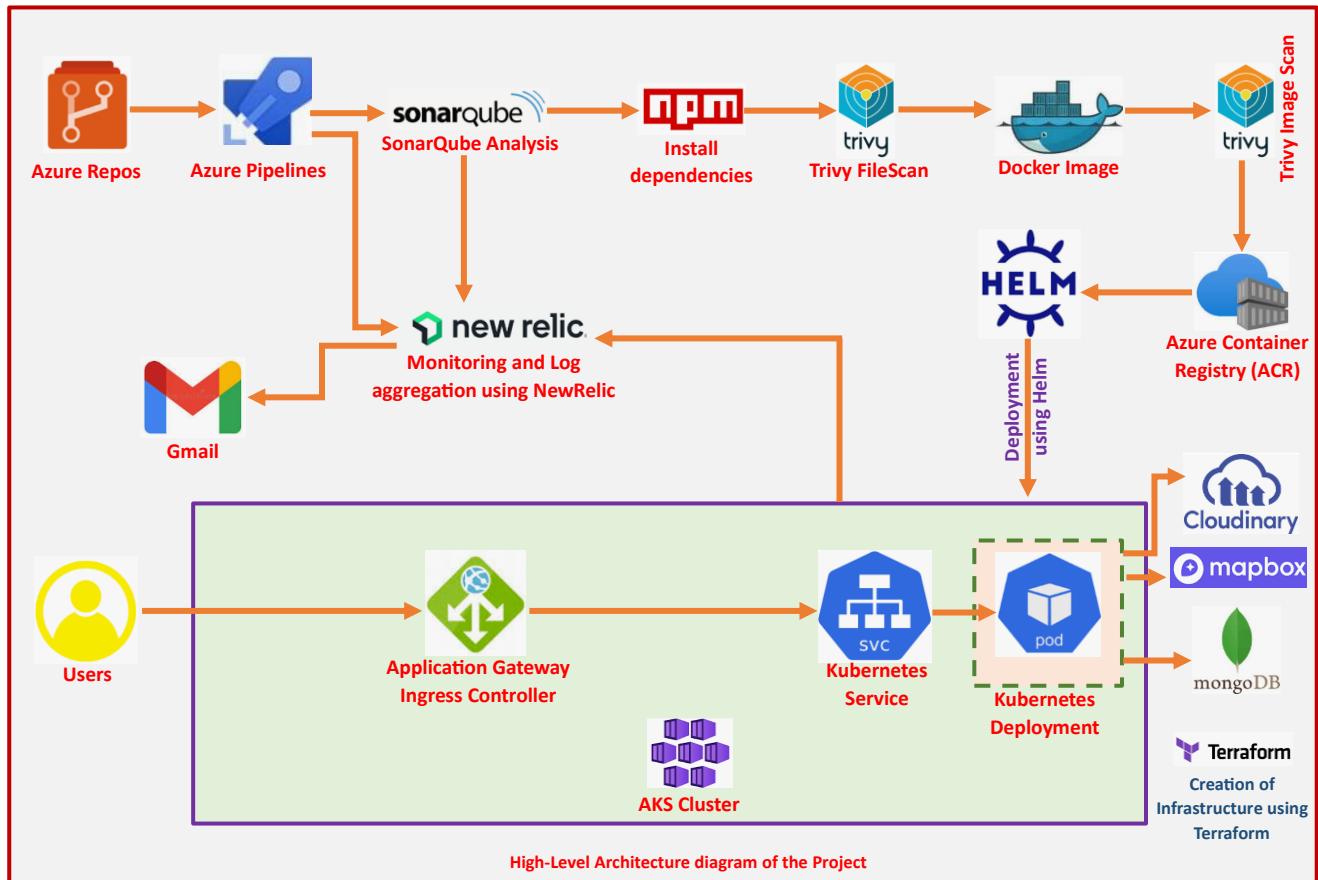


## DevOps Project YelpCamp Azure



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/singhrithesh85/helm-chart-yelp-camp.git>. Credentials used in this project had been changed later.

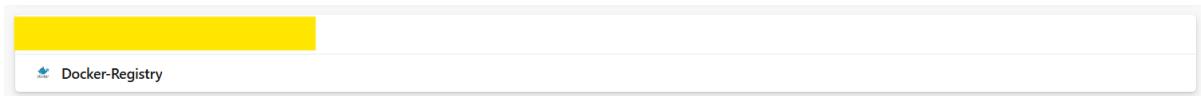
Using Cloudinary I had uploaded the images to the website. Mapbox had been used to locate a location using Maps and MongoDB is the database used for this project.

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.

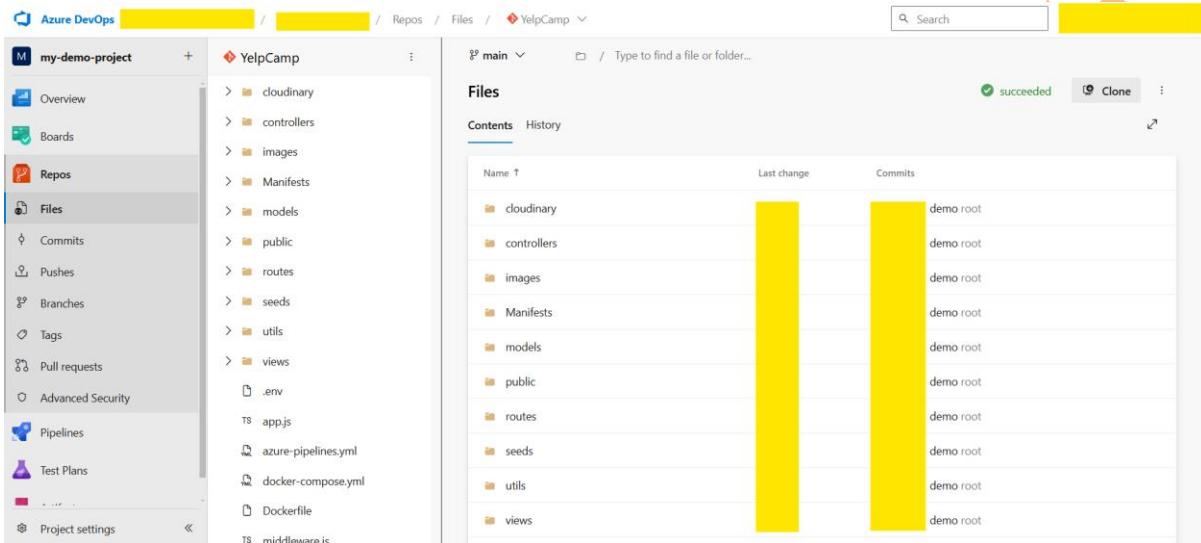
To monitor Jenkins Job using NewRelic you need to install OpenTelemetry Plugin in Jenkins and do the configuration from Jenkins Systems and for monitoring SonarQube the procedure to install the NewRelic Agent has been discussed later in this project.

The Source code for this project is present in the GitHub Repository <https://github.com/singhrithesh85/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
```

Riit

```
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
```

```
[root@[REDACTED] ~]# kubectl apply -f secret.yaml
secret/mysecret created
[root@[REDACTED] ~]# 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
[root@[REDACTED] ~]# 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: 2[REDACTED]
Type:        kubernetes.io/service-account-token

Data
====

token: 9 bytes
[REDACTED]
ca.crt: 1761 bytes
```

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:

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```





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 image displays four screenshots illustrating the configuration of various services:

- MongoDB Configuration:** Shows the MongoDB Atlas interface with a Node.js driver setup. It includes instructions for installation via npm and provides a connection string template: `mongodb+srv://[REDACTED]@dexter.[REDACTED].mongodb.net/?retryWrites=true&w=majority&appName=dexter`. A note says to replace `cdb_password` with the password for the `mongo` database user.
- Mapbox Configuration:** Shows the Mapbox account page under "Access tokens". It features a "Default public token" section with a redacted token, last modified time, and URLs. It also lists "Current billing period usage" for Temporary Geocoding API and Maps, and links to Documentation, See how billing works, Contact sales, Help, and Playground.
- Cloudinary Configuration:** Shows the Cloudinary interface under "Getting Started". It displays the "Upload, Optimize and Transform" section with a code snippet for index.js:

```

import { v2 as cloudinary } from 'cloudinary';

(async function() {
    // Configuration
    cloudinary.config({
        cloud_name: '[REDACTED]',
        api_key: '[REDACTED]',
        api_secret: '[REDACTED]' // Click 'View API Keys' above to copy your API secret
    });
}

```

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_aXXXXXXXXXXXXXXXXXXXXXXXXXXXXd'
    - 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.

The screenshot shows the Azure DevOps Pipelines interface. At the top right, there is a blue button labeled "New pipeline" and a three-dot menu icon. Below the header, there are tabs for "Recent", "All", and "Runs", with "Recent" being the active tab. A search bar labeled "Filter pipelines" is located to the right of the tabs. Under the "Recently run pipelines" section, there is a table with two columns: "Pipeline" and "Last run". The first row shows a green checkmark icon next to "YelpCamp", followed by the text "• Updated azure-pipelines.yml" and "Individual CI for main".

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

The terminal output displays the results of several kubectl commands:

- `kubectl get all -n yelp-camp`: Shows a table of resources including a pod, service, deployment, and replicaset, all in a healthy state (READY).
- `kubectl get ing -n yelp-camp`: Shows a table of an ingress resource named "yelpcamp-ingress" with one host mapping to an external IP.

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=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX XXX -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.

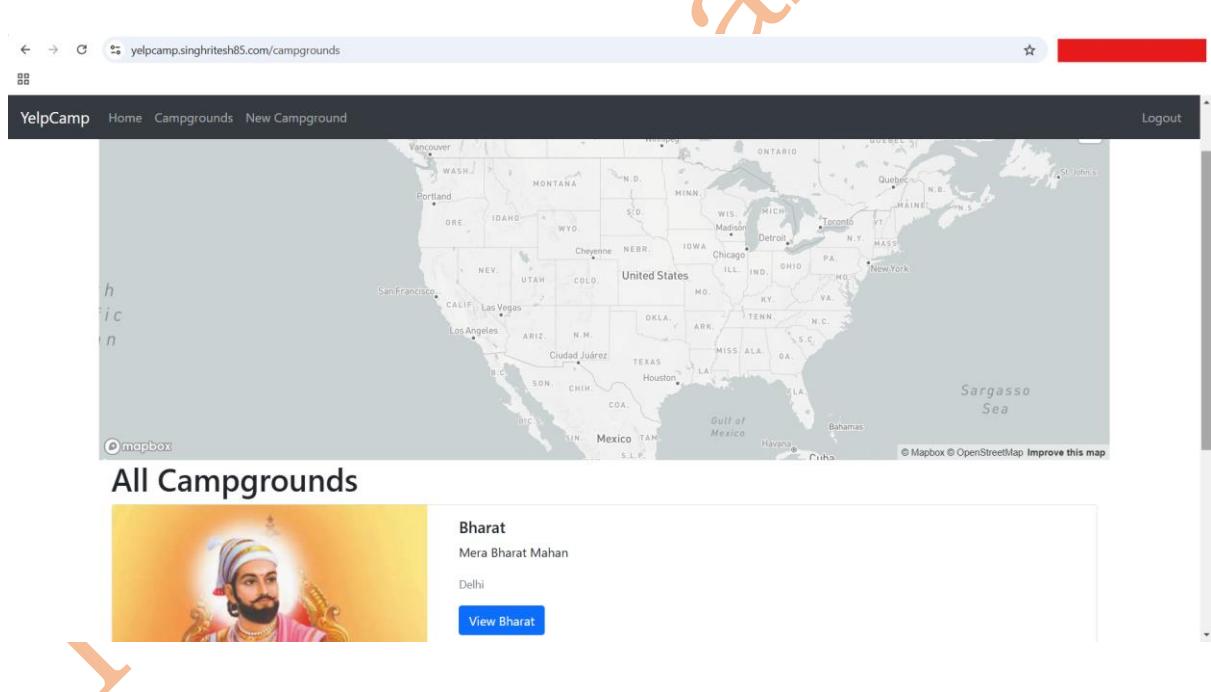
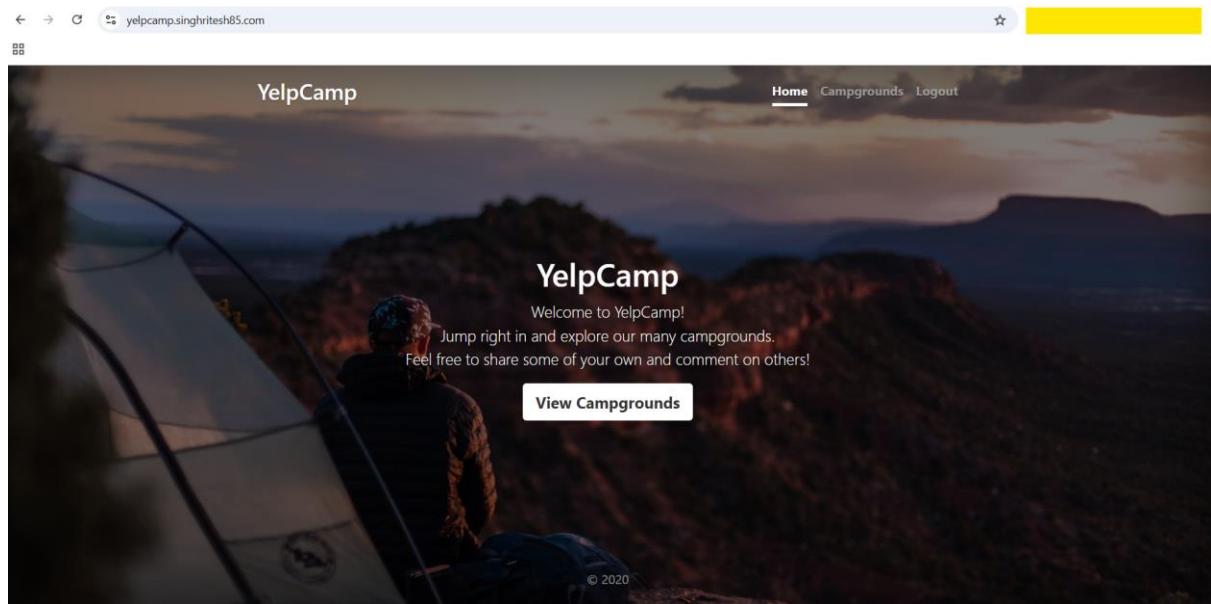
The screenshot shows the SonarQube web interface. At the top, there's a navigation bar with links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. A search bar at the top right allows you to "Search for projects..." and includes a "Create Project" button. Below the navigation, there's a "Filters" section with dropdowns for "My Favorites" and "All". A search bar for "Search by project name or key" is present. The main area displays "1 project(s)" named "yelpcamp" with a status of "Passed". Below the project name, there are several metrics: Bugs (0), Vulnerabilities (0), Hotspots Reviewed (0.0%), Code Smells (5), Coverage (0.0%), Duplications (0.0%), and Lines (10k). The "Reliability" section shows a single A rating. The "Security" section also shows a single A rating. A footer note states "SonarQube™ technology is powered by SonarSource SA".

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.

The screenshot shows the Azure DNS Zone management interface for the domain "singhritesh85.com". On the left, there's a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Properties, Locks, DNS Management, and Recordsets. The "Recordsets" option is currently selected. The main pane shows a table of record sets:

Name	Type	TTL	Value	Alias resource type	Alias target
sonarqube	NS	172800	[REDACTED]		
	SOA	3600	[REDACTED]		
sonarqube	A	3600	[REDACTED]		
yelpcamp	A	3600	[REDACTED]		

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.

The screenshot shows the MongoDB Atlas Data Services interface. On the left, there's a sidebar with options like Overview, DATABASE, Clusters, SERVICES, SECURITY, and Advanced. The main area shows a database named 'test' with a collection named 'campgrounds'. A query is being run, and the results show two documents. The first document has fields: \_id, geometry, reviews, title, location, price, description, images, author, and \_\_v. The second document has similar fields but with different values.

This screenshot is similar to the one above, showing the same MongoDB Atlas interface. It displays a query result for a different document in the 'campgrounds' collection. The document includes fields such as \_id, geometry, reviews, title, location, price, description, images, author, and \_\_v. The visual style is identical to the first screenshot, with orange annotations highlighting specific parts of the interface.

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

The screenshot shows the NewRelic Integrations & Agents page for the Node.js data source. It provides instructions for setting up monitoring. Step 1: Add 'newrelic' as a dependency to your package.json file. Step 2: In the first line of your app's main module, add: require('newrelic');. There are also links for more install options and package.json release notes.

**Integrations & Agents**

← Integrations & Agents / Node.js

Installing on account: [REDACTED] [Leave Feedback](#)

**Data source**  
**Node.js**  
Pinpoint and solve issues down to the line of code with Node.js monitoring from New Relic. With features like service maps and error analytics, our Node.js agent helps you get the full picture of your app environment.

Select instrumentation method  
 Enter your credentials  
 Instrument your container  
 Test the connection

**3 Add this ENV line to your Dockerfile.**  
This line allows the agent to run without a configuration file.

Dockerfile 1 line 33.0 B [Copy to clipboard](#)

```
1 ENV NEW_RELIC_NO_CONFIG_FILE=true
```

**Integrations & Agents**

← Integrations & Agents / Node.js

Installing on account: [REDACTED] [Leave Feedback](#)

**Data source**  
**Node.js**  
Pinpoint and solve issues down to the line of code with Node.js monitoring from New Relic. With features like service maps and error analytics, our Node.js agent helps you get the full picture of your app environment.

Select instrumentation method  
 Enter your credentials  
 Instrument your container  
 Test the connection

**4 Set the config options via ENV directives**  
In addition to setting distributed tracing and logging, you can set other agent configuration options [\[?\]](#).

Analyze your AI application's responses  
Turn this on to enable AI monitoring for your AI application.

**Protect your license key**  
Do not include your license key in your Dockerfile or Docker image. For more information, see our documentation on license key security.  
[See our docs](#)

Dockerfile 2 lines 71.0 B [Copy to clipboard](#)

```
1 ENV NEW_RELIC_DISTRIBUTED_TRACING_ENABLED=true
2 ENV NEW_RELIC_LOG=stdout
```

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=[REDACTED] L
ENV NEW_RELIC_APP_NAME="yelpcamp"
```

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

**new relic** **APM & Services**

Quick Find [+ Create a workload](#) [+ Add data](#)

Search by entity name Entity Type = Service - APM, Service - OpenTelemetry [\[?\]](#)

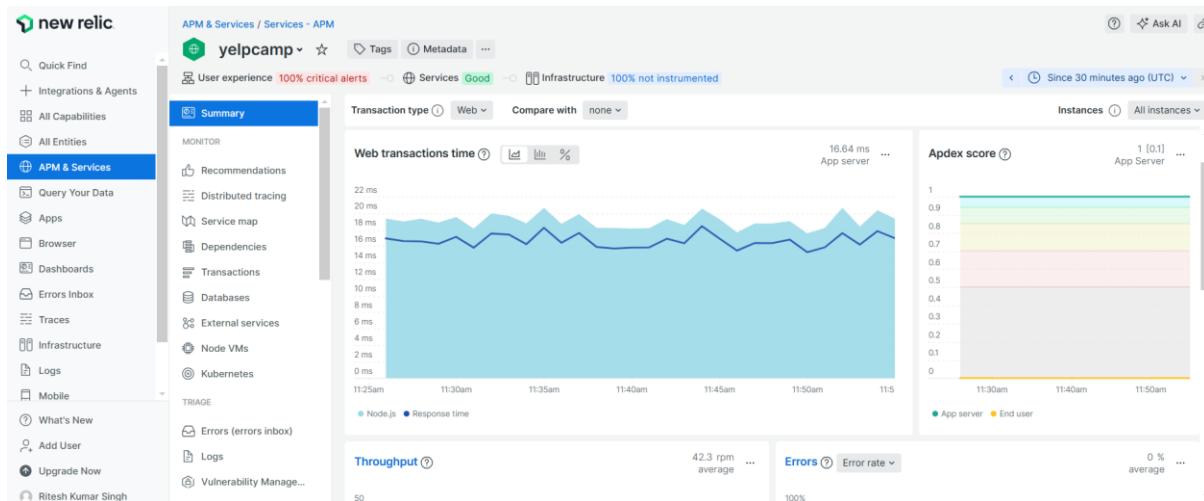
All Capabilities All Entities [List](#) [Navigator](#) [Lookout](#)

**APM & Services**

Query Your Data Apps Browser

**Services - APM** [View all \(1\)](#)

Name	Response time (ms)	Throughput	Error rate
yelpcamp	18 ms	24.4 rps	0%



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

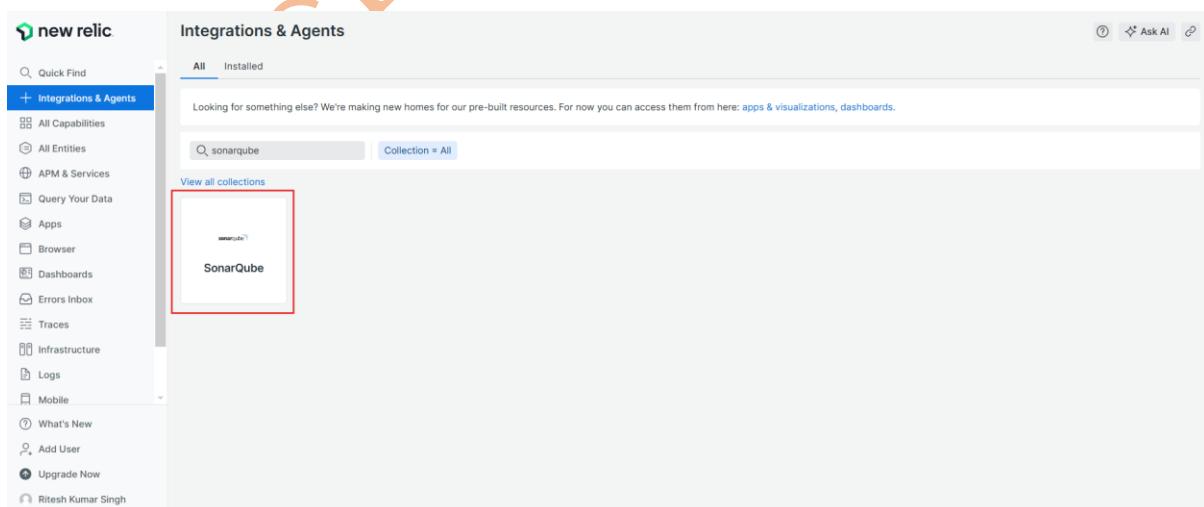
$$\text{Appdex Score} = \frac{\text{Satisfied Request} + (\text{Tolerated Request} / 2)}{\text{Total Number of Request}}$$

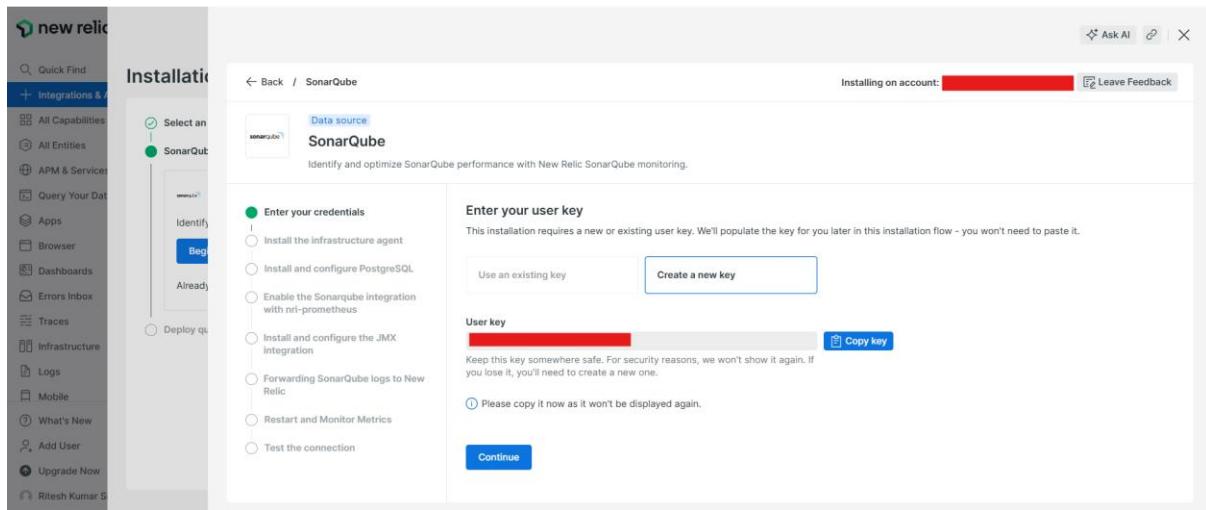
Total Request = Satisfied Request + Tolerated Request + Frustrated Request

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.

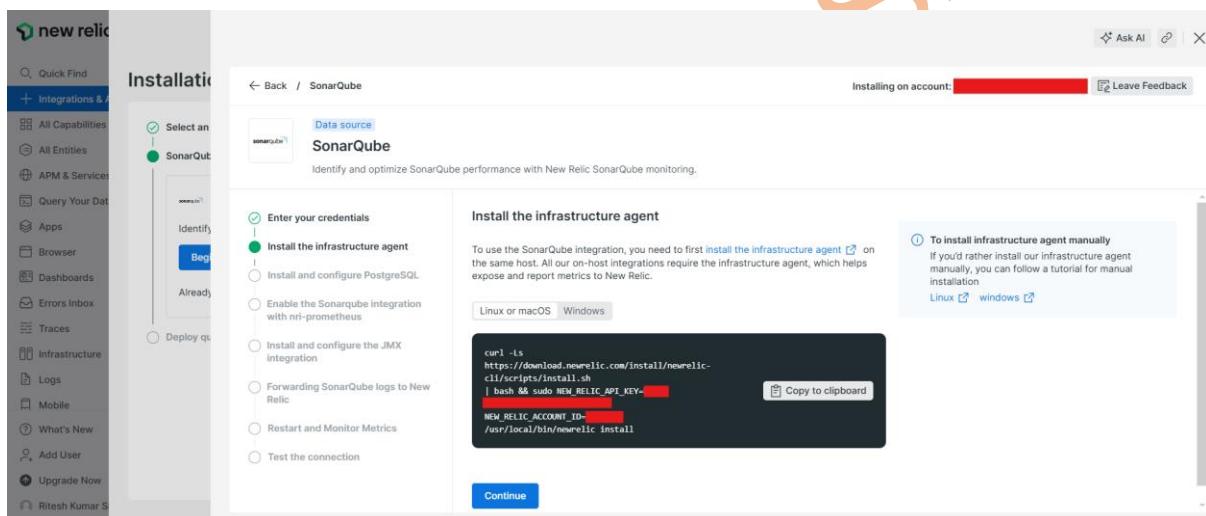
## Monitoring SonarQube with NewRelic

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.

```
[root@SonarQube-Server ~]# curl -Ls https://download.newrelic.com/install/newrelic-cli/scripts/install.sh | bash && sudo NEW_RELIC_API_KEY=[REDACTED] NEW_RELIC_ACCOUNT_ID=[REDACTED] /usr/local/bin/newrelic install
Installing New Relic CLI v0.97.1
Installing to /usr/local/bin

[New Relic]
[New Relic]

Welcome to New Relic. Let's set up full stack observability for your environment.
Our Data Privacy Notice: https://newrelic.com/termsandconditions/services-notices

✓ Connecting to New Relic Platform
Connected

Installing New Relic
```

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 (>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]:[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 (>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`

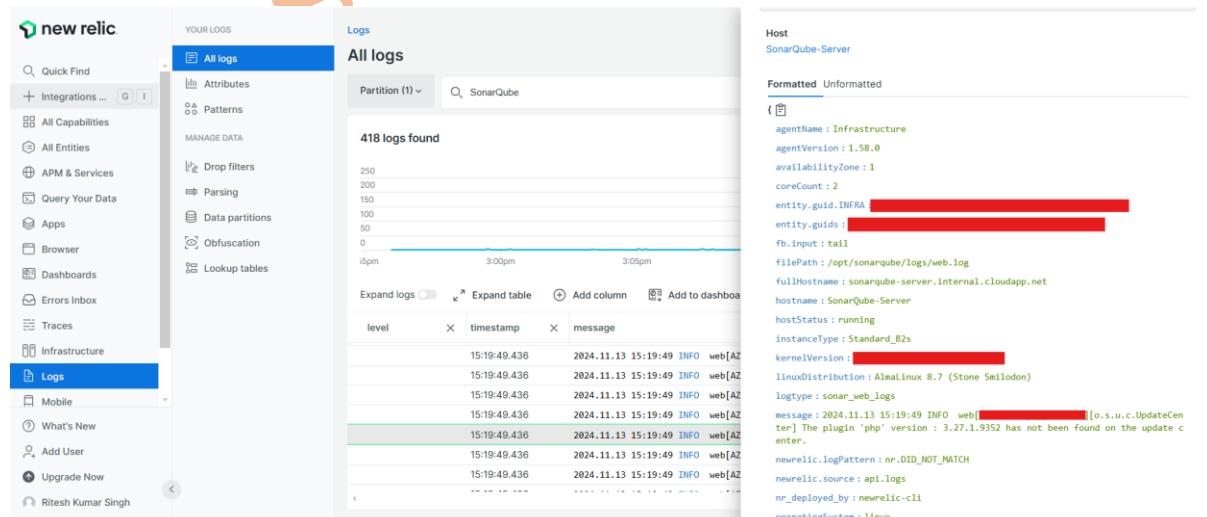
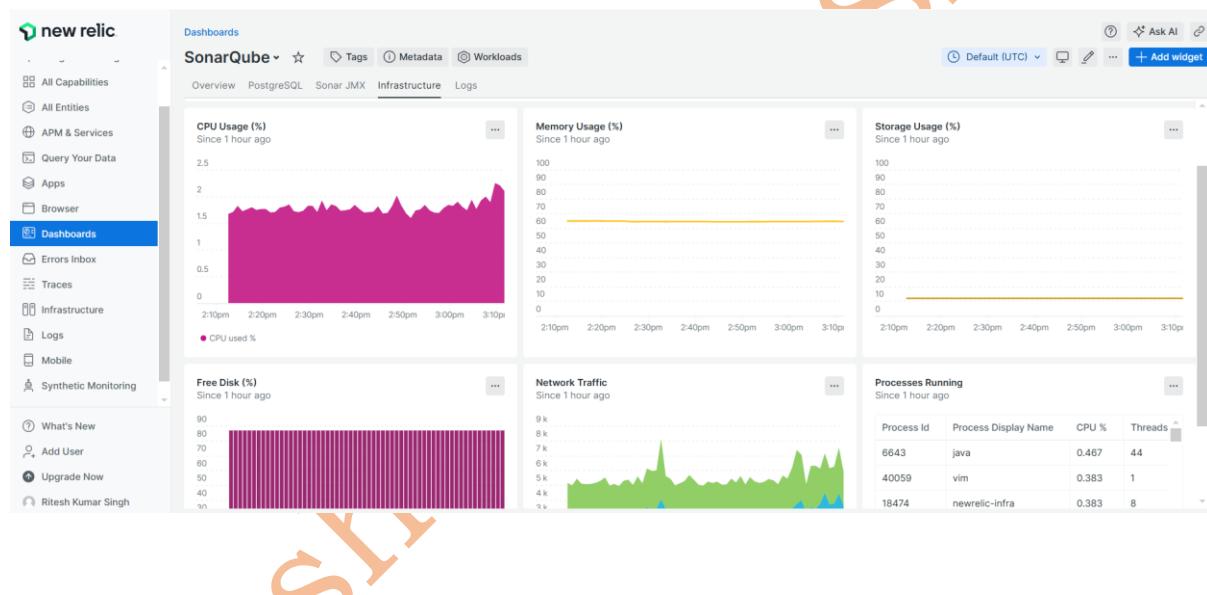
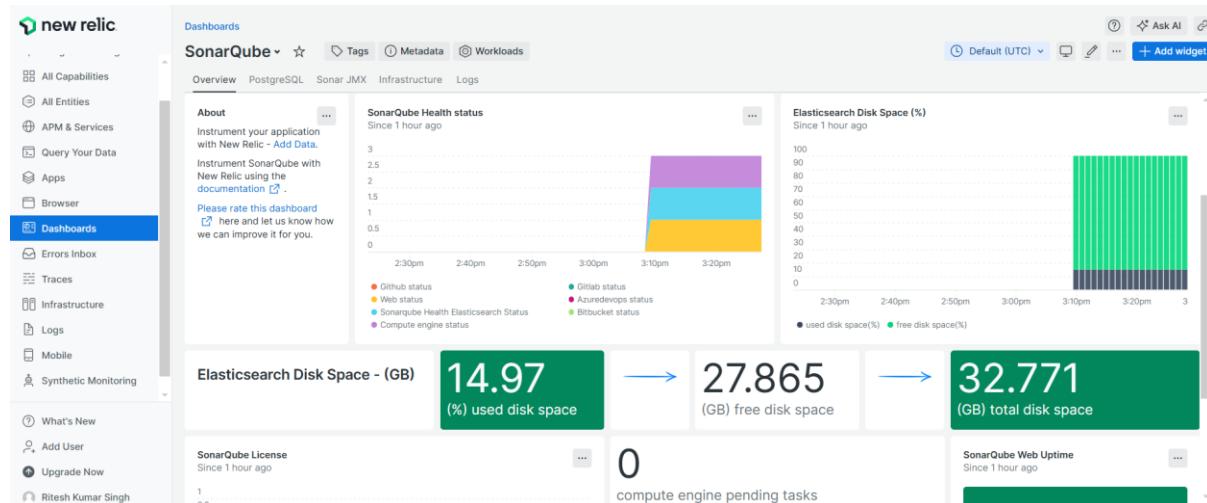
```
logs:
  - name: cloud-init.log
    file: /var/log/cloud-init.log
    attributes:
      logtype: linux_cloud-init
  - name: messages
    file: /var/log/messages
    attributes:
      logtype: linux_messages
  - name: secure
    file: /var/log/secure
    attributes:
      logtype: linux_secure
  - 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 opt]# 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.

The screenshot shows the NewRelic interface under 'Integrations & Agents'. On the left sidebar, 'Kubernetes' is selected. The main panel is titled 'Kubernetes' and contains a brief description: 'Our Kubernetes monitoring solution gives you visibility into your Kubernetes clusters and workloads in minutes, whether your clusters are hosted on-premises or in the cloud.' Below this, a vertical list of steps is shown: 'Select instrumentation method', 'Enter your credentials', 'Configure the Kubernetes integration', 'Select additional data', 'Gather Log data', 'Install the Kubernetes integration', and 'Test the connection'. Step 1, 'Install the Kubernetes integration', is expanded, showing a command line with placeholder values for license keys and cluster names. A 'Copy to clipboard' button is present. Step 2, 'Download the values configuration file (optional)', is also visible.

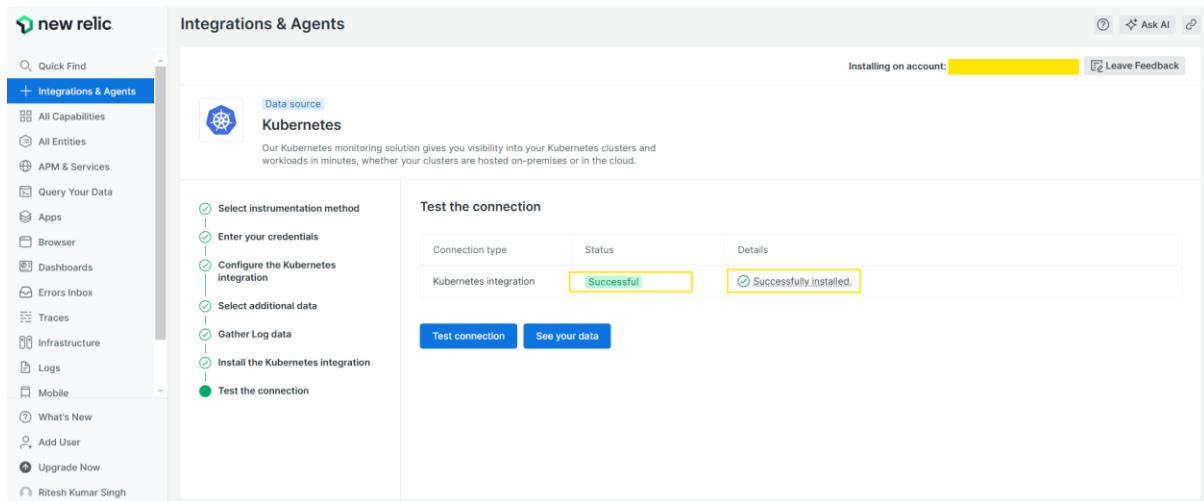
I ran the above command from a Linux machine where I kept the kubeconfig file and kubectl and helm was installed.

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

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.

The screenshot shows the New Relic interface for installing the infrastructure agent on Linux. The left sidebar is visible with various monitoring options like All Entities, APM & Services, and Infrastructure. The main panel is titled 'Integrations & Agents' and specifically for 'Linux'. It shows a penguin icon and the text 'Data source: Linux'. Below this, there are three options: 'Enter your credentials' (selected), 'Install agent' (highlighted in green), and 'Test the connection'. The 'Install agent' section contains a command-line instruction:

```
curl -Ls https://download.newrelic.com/install/newrelic-cli/scripts/install.sh | bash && sudo NEW_RELIC_API_KEY=[REDACTED] /usr/local/bin/newrelic install
```

Below the command are two radio button options: 'Automatically answer "yes" to all install prompts. We'll stop the installer if there's an error.' and 'Use a proxy'. At the bottom, there's a 'Tags (optional)' section with a key-value pair input field.

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

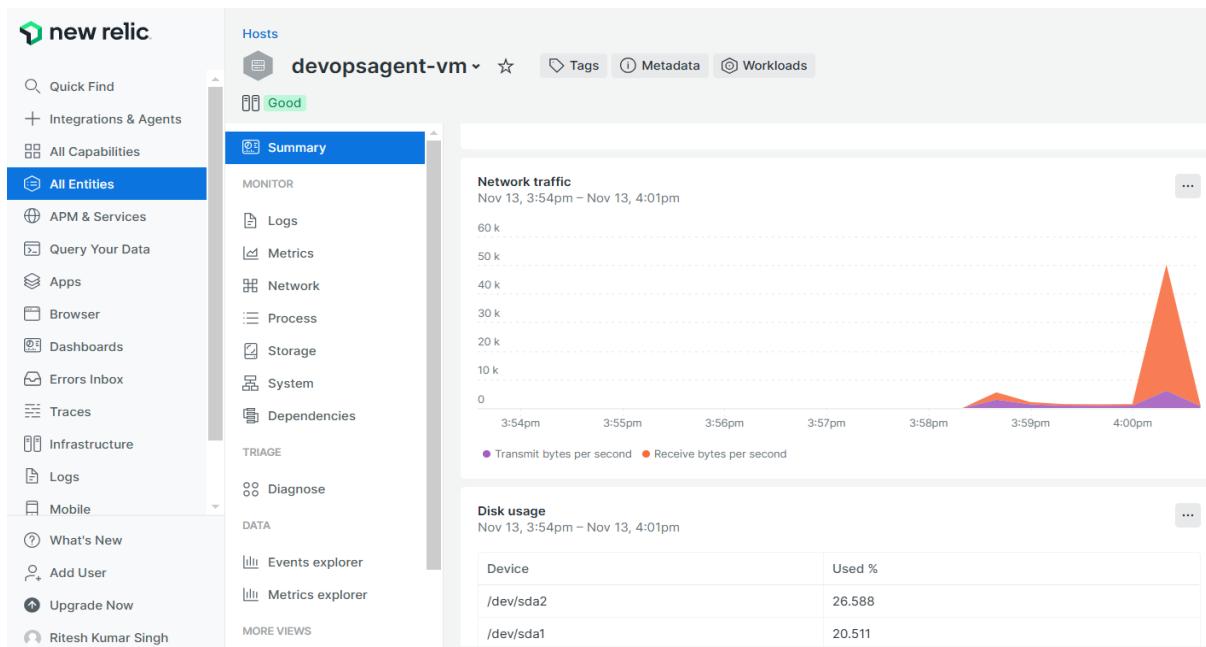
```
[root@devopsagent-vm ~]# curl -Ls https://download.newrelic.com/install/newrelic-cli/scripts/install.sh | bash && sudo NEW_RELIC_API_KEY=[REDACTED] /usr/local/bin/newrelic install
Installing New Relic CLI v0.97.1
Installing to /usr/local/bin
```

The terminal output shows the command being run and the New Relic CLI version being installed. The final line indicates it is installing to the /usr/local/bin directory.

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.

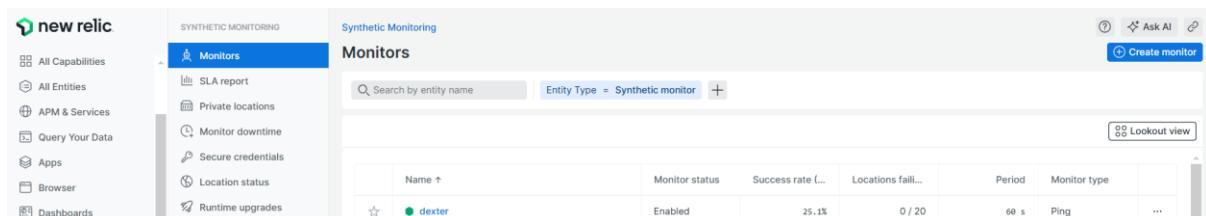
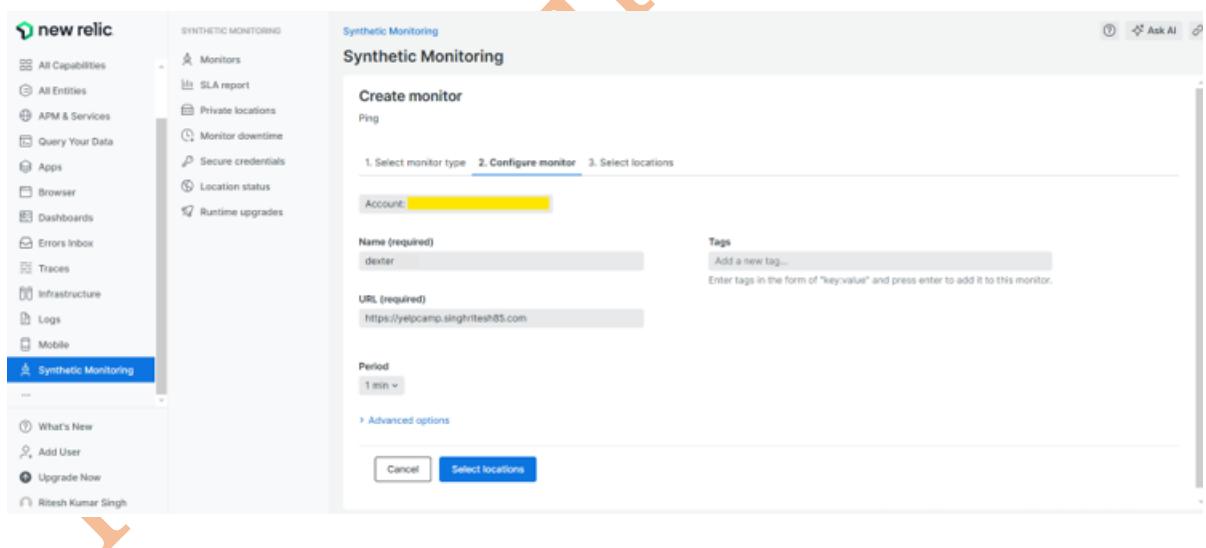
The screenshot shows the New Relic Infrastructure dashboard for the host 'devopsagent-vm'. The left sidebar is visible with 'All Entities' selected. The main panel displays the 'Summary' view for the host. It includes several monitoring cards:

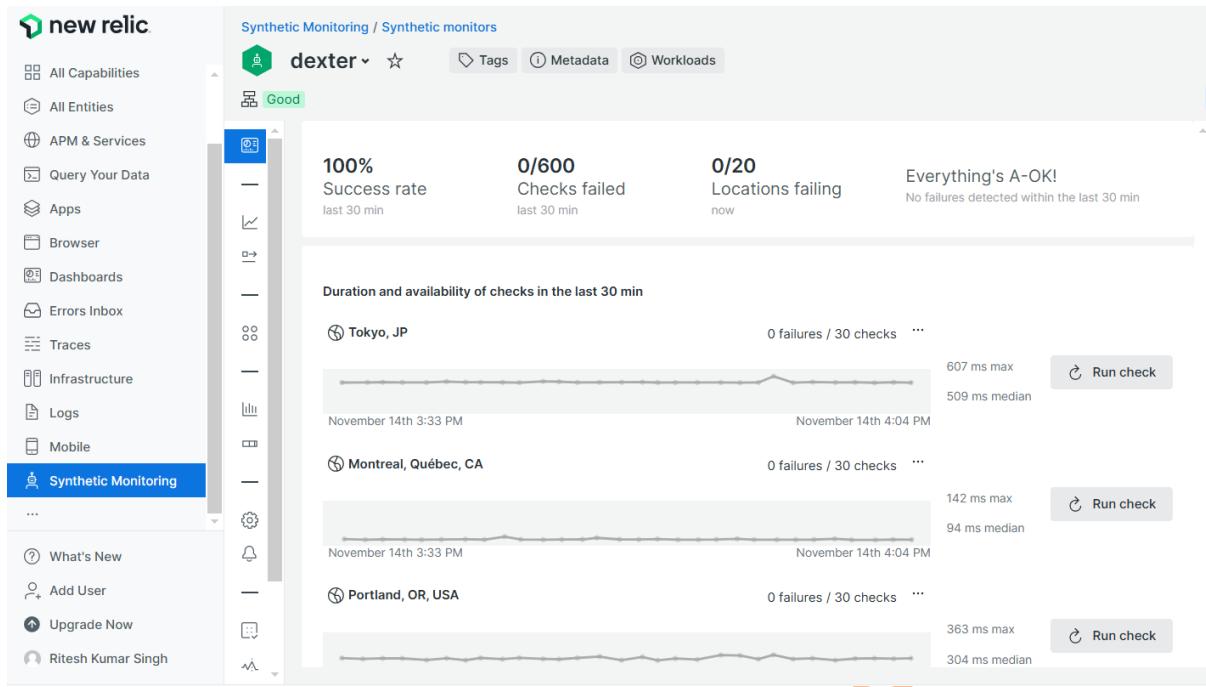
- CPU usage (%)**: A line graph showing CPU usage over time from Nov 13, 3:54pm to Nov 13, 4:01pm. The usage fluctuates between 0% and 100%.
- Memory usage (%)**: A line graph showing memory usage over the same period. It also fluctuates between 0% and 100%.
- Logs**, **Metrics**, **Network**, **Process**, **Storage**, **System**, and **Dependencies** cards for further monitoring.
- TRIAGE** and **DATA** sections with links to 'Diagnose', 'Events explorer', and 'Metrics explorer'.



## 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.

Destinations (2) See notifications log

Type	Name ↑	URL/Details	Last Updated	Updated By	Used By	Enabled
	Email	[REDACTED]	[REDACTED]	Ritesh Kumar Singh	2 workflows	<input checked="" type="checkbox"/>

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

The screenshot shows the 'Set thresholds' step of the alert condition configuration. The configuration includes:

- Tell us where to look:** Set thresholds
- Fine-tune your signal:**
  - Data aggregation:** Window duration: 1 minutes
  - Streaming method:** Event timer (selected)
  - Gap filling strategy:** Fill data gaps with: [REDACTED]
- Set condition thresholds:**
  - Open incidents with a:** Severity level: Critical
  - When a query returns a value:** above 0 at least once in 1 minutes
  - Add threshold:** + Add lost signal threshold

At the bottom right, the 'Save condition' button is highlighted with a yellow box.

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 screenshot shows the 'Alerts / Alert Policies' section in New Relic. The 'Initial policy' is selected. The table below lists two alert conditions:

Alert condition	Query	Thresholds	Type	Open issues	Last modified	...
App-URL	SELECT filter(count(*), WHERE ...)	Critical: above 0 at least once in 1 minute Create a warning threshold	NRQL Query	0	[Redacted]	...
CPU-Utilization	SELECT average(cpuPercen...)	Critical: above 70 for at least 5 minutes Create a warning threshold	NRQL Query	0	[Redacted]	...

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.

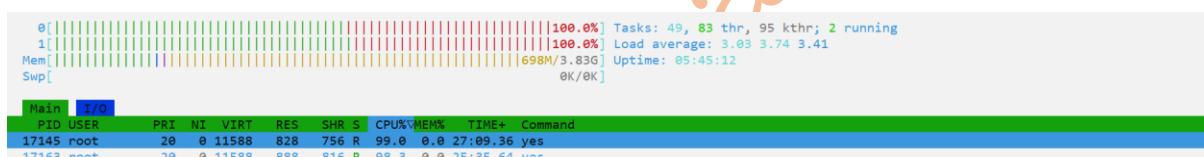
The screenshot shows the 'Edit workflow' page. In the 'Filter data' section, the 'Policy' dropdown is set to 'Initial policy [57/0]'. In the 'Notify' section, there is a list of destinations: 'Activated', 'Acknowledged', and 'Closed'. The 'Update workflow' button is visible at the bottom right.

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.

The screenshot shows the New Relic interface under the 'ANALYZE' tab, specifically the 'Issues & Activity' section. The left sidebar includes options like Overview, Alert Conditions, Alert Policies, Alert Coverage G... (Beta), Correlate, Muting Rules, Workflows, Destinations, and General. The main area displays a search bar with 'Filter by ID or names' and 'Issue state = Active'. A message at the top right says 'You're seeing the latest issues' and 'No issues in sight—it's time to relax.' It also shows system statistics: 100.0% CPU usage, 2 tasks running, 3.03 load average, and 05:45:12 uptime.

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.

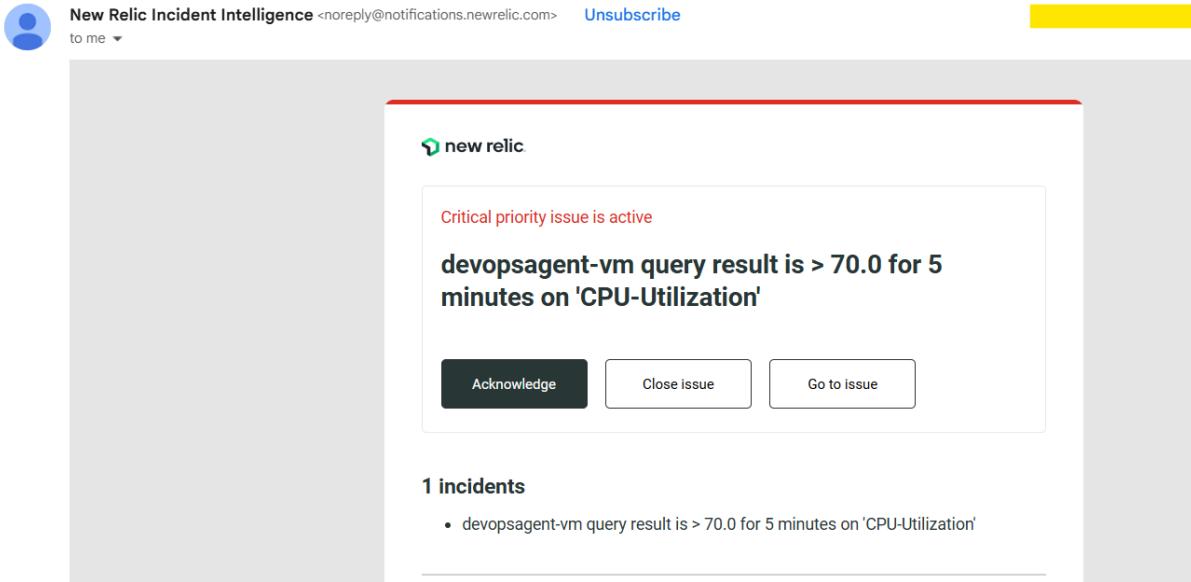
The screenshot shows the New Relic interface under the 'ANALYZE' tab, specifically the 'Issues & Activity' section. The left sidebar includes options like Overview, Alert Conditions, Alert Policies, Alert Coverage G... (Beta), Correlate, Muting Rules, Workflows, Destinations, and General. The main area displays a search bar with 'Filter by ID or names' and 'Issue state = Active'. A message at the top right says 'You're seeing the latest issues'. It shows 'Showing 1 issues' with 0 rows selected. A single alert is listed:

State	Priority	Created	Duration	Issue name	Entity name	Notified	Contains	Actions taken
Active	Critical	5m ago	5m	devopsagent-vm qu...	● devopsag...	✉	1 incident	...

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.

devopsagent-vm query result is > 70.0 for 5 minutes on 'CPU-Utilization' [Inbox](#) [x](#)



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

The screenshot shows the "Issues & Activity" section of the New Relic interface. The left sidebar has categories like "Quick Find", "Integrations & Agents", "All Capabilities", "All Entities", "APM & Services", "Query Your Data", "Apps", "Browser", "Dashboards", and "Errors Inbox". The main area is titled "Issues & Activity" and shows "Showing 2 issues" with a "Beta" badge. A filter bar at the top says "Issue state = Active". Below is a table with columns: State, Priority, Created, Duration, Issue name, Entity name, Notified, and Contains. Two rows are listed:

State	Priority	Created	Duration	Issue name	Entity name	Notified	Contains
Active	Critical	1m ago	1m	dexter query result i...	dexter	<a href="#">✉</a>	1 incident
Active	Critical	5m ago	5m	devopsagent-vm qu...	devopsag...	<a href="#">✉</a>	1 incident

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.

Critical priority issue is active

**dexter query result is > 0.0 on 'App-URL'**

**Acknowledge**   **Close issue**   **Go to issue**

**1 incidents**

- dexter query result is > 0.0 on 'App-URL'

---

**1 impacted entities**

- dexter

---

**Alert Policy**

Policy Name	Initial policy
Condition	App-URL

**Reference:** - <https://github.com/Colt/YelpCamp.git>