Configuration Manual

MSc Research Project
Cloud Computing

Ritesh Kumar Rout
Student ID: 21127069

School of Computing
National College of Ireland

Supervisor: Prof. Sai Ranjan Emani

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ritesh Kumar Rout
Student ID:	21127069
Programme:	Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Prof. Sai Ranjan Emani
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	1497
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Ritesh Kumar Rout
Date:	12th December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ritesh Kumar Rout
21127069

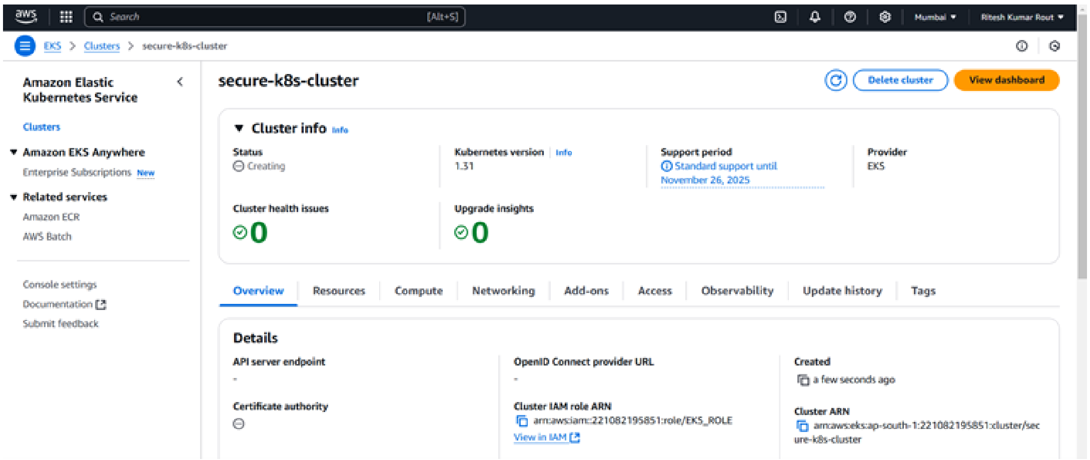
1 Overview

This configuration manual provides step-by-step instructions for setting up and configuring a secure Kubernetes environment using Amazon Elastic Kubernetes Service (EKS). It is intended to ensure a production-grade setup that adheres to security best practices.

2 Prerequisites

- Active AWS account (https://aws.amazon.com/console/)
- AWS CLI
- Kubectl (Kubernetes CLI) (https://kubernetes.io/releases/download/)
- Docker (https://www.docker.com/products/docker-desktop/)
- Create an IAM role with the appropriate permissions for EKS operations to ensure the cluster functions correctly.
- Networking configurations should be in place, including the creation of a Virtual Private Cloud (VPC) with both public and private subnets.

3 Setting Up the EKS Cluster



secure-k8s-cluster

Cluster info

Status: Creating

Kubernetes version: 1.31

Support period: Standard support until November 26, 2025

Provider: EKS

Cluster health issues: 0

Upgrade insights: 0

Details

API server endpoint: -

Certificate authority: ⊖

OpenID Connect provider URL: -

Cluster IAM role ARN: arn:aws:iam::221082195851:role/EKS_ROLE

Created: a few seconds ago

Cluster ARN: arn:aws:eks:ap-south-1:221082195851:cluster/secure-k8s-cluster

Figure 1: Setting Up the EKS Cluster

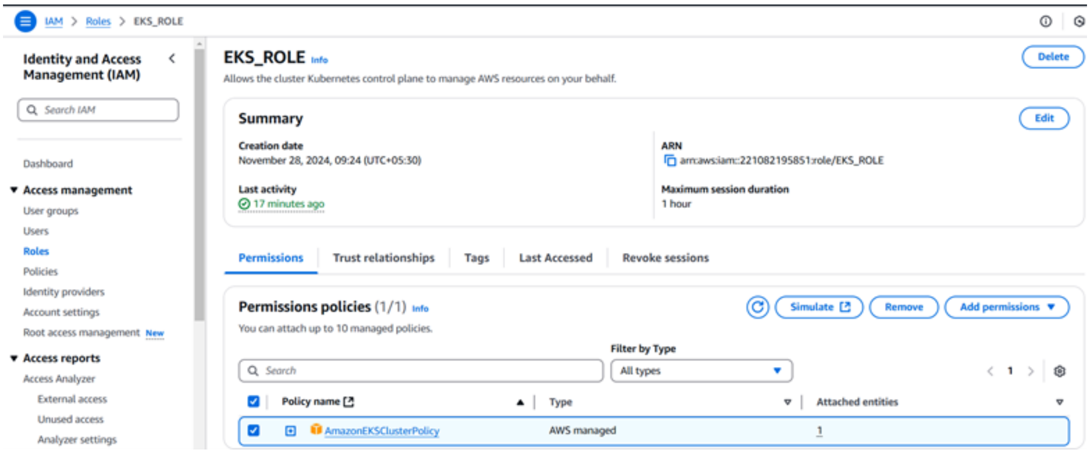
1. In order to create the EKS cluster, the cluster name is specified, the proper Kubernetes version, and the cluster type, as seen Figure 1.

2. IAM roles granted to the EKS cluster for resource management. or the networking configurations.

3. Setup a Virtual Private Cloud (VPC) with public and private subnets, route tables and the internet gateways.

4. Checked with kubectl commands using ”aws eks –region region-name update-kubeconfig –name cluster-name”.

4 Configuring EKS Cluster Policy



EKS_ROLE

Allows the cluster Kubernetes control plane to manage AWS resources on your behalf.

Summary

Creation date: November 28, 2024, 09:24 (UTC+05:30)

ARN: arn:aws:iam::221082195851:role/EKS_ROLE

Last activity: 17 minutes ago

Maximum session duration: 1 hour

Permissions policies (1/1)

You can attach up to 10 managed policies.

Filter by Type: All types

☑	Policy name	Type	Attached entities
☑	AmazonEKSClusterPolicy	AWS managed	1

Figure 2: Create an EKS Cluster Policy

1. The IAM role created with the cluster has the AmazonEKSClusterPolicy, which gives permissions needed for the EKS control plane figure 2.

2. This policy enables the Kubernetes control plane to delegate API calls from the user to AWS services like the node management and workload scheduling feature.

3. Perform test Configuration ”kubectl auth can-i create pod”.

5 Setting Up Worker Nodes

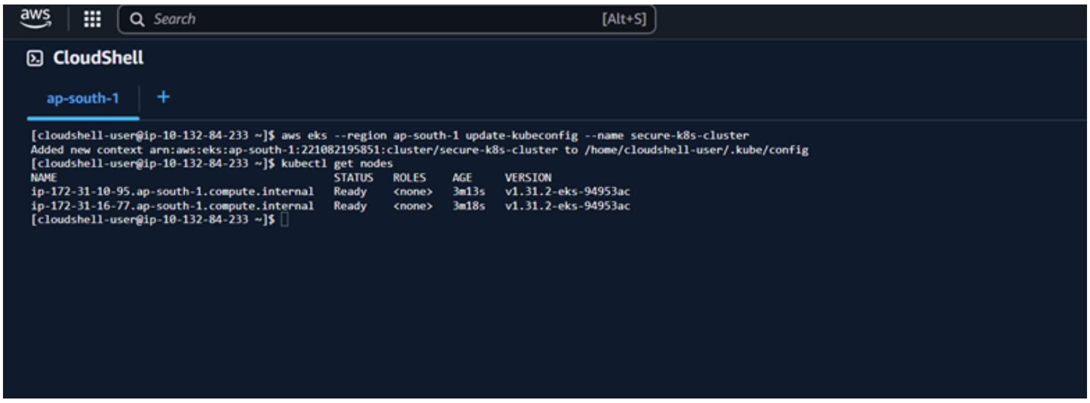
1. Before configuring worker nodes for an EKS cluster, an IAM role for the nodes needs to be created.

2. The IAM role gives the permission needed for the EC2 instance to communicate via any other AWS Services, like pulling the container images from the ECR and communicating with the other resources within the VPC.

3. After configuring the IAM role, you can now provision the node group. This means choosing the correct instance types, and disk sizes, and checking that the Kubernetes version is supported.

4. Verify Node Command ”kubectl get nodes”.

6 Connecting to the EKS Cluster



```
[cloudshell-user@ip-10-132-84-233 ~]$ aws eks --region ap-south-1 update-kubeconfig --name secure-k8s-cluster
Added new context arn:aws:eks:ap-south-1:221082195851:cluster/secure-k8s-cluster to /home/cloudshell-user/.kube/config
[cloudshell-user@ip-10-132-84-233 ~]$ kubectl get nodes
NAME                                         STATUS   ROLES    AGE     VERSION
ip-172-31-10-95.ap-south-1.compute.internal  Ready    <none>   3m13s   v1.31.2-eks-94953ac
ip-172-31-16-77.ap-south-1.compute.internal  Ready    <none>   3m18s   v1.31.2-eks-94953ac
[cloudshell-user@ip-10-132-84-233 ~]$
```

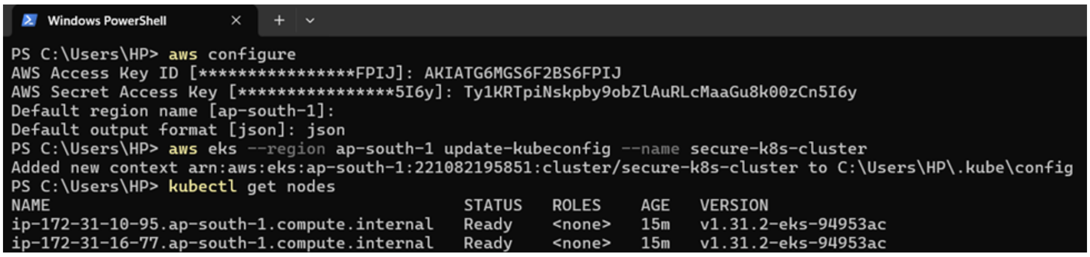
Figure 3: Connecting to the EKS Cluster

1. AWS CloudShell Environment Data for EKS Cluster named ap-south-1 and the output includes details about the cluster, the IAM role associated with it, its status (”Ready”), and the version of the EKS control plane specified Figure 3.Command: aws eks –region ap-south-1 update-kubeconfig –name secure-k8s-cluster

2. For configuring a policy for the EKS cluster, the AmazonEKSClusterPolicy should be attached to the IAM role that is associated with the cluster.

3. This is actually a policy that allows the EKS control plane to talk to AWS services on behalf of the user to manage resources such as nodes and schedule workloads in a Kubernetes environment.

7 Implementing Role-Based Access Control (RBAC)



```
PS C:\Users\HP> aws configure
AWS Access Key ID [****************FPIJ]: AKIATG6MGS6F2BS6FPIJ
AWS Secret Access Key [****************5I6y]: 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
Default region name [ap-south-1]:
Default output format [json]: json
PS C:\Users\HP> aws eks --region ap-south-1 update-kubeconfig --name secure-k8s-cluster
Added new context arn:aws:eks:ap-south-1:221082195851:cluster/secure-k8s-cluster to C:\Users\HP\.kube\config
PS C:\Users\HP> kubectl get nodes
NAME                                          STATUS   ROLES    AGE   VERSION
ip-172-31-10-95.ap-south-1.compute.internal   Ready    <none>   15m   v1.31.2-eks-94953ac
ip-172-31-16-77.ap-south-1.compute.internal   Ready    <none>   15m   v1.31.2-eks-94953ac
```

Figure 4: Enable RBAC (Role-Based Access Control)

1. Figure 4 displays information regarding an AWS CloudShell environment with the example of an EKS (Amazon Elastic Kubernetes Service) cluster called ap-south-1. To connect with cmd you need to provide these following information:

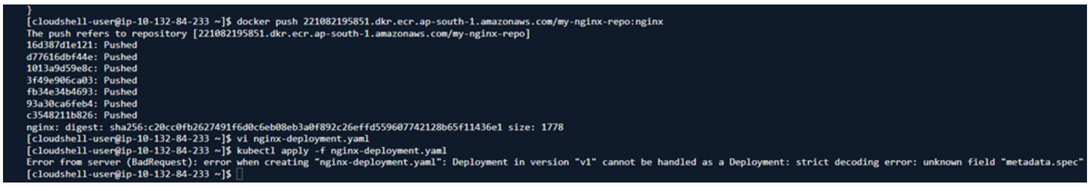
   - bash : aws configure

- AWS Access Key
- AWS Secret Key
- Region
- Default Output Format: JSON

2. The IAM role of the cluster, status and EKS control plane software version are present the use of a RBAC (Role-Based Access Control) configuration applied to a Kubernetes cluster via kubectl command.

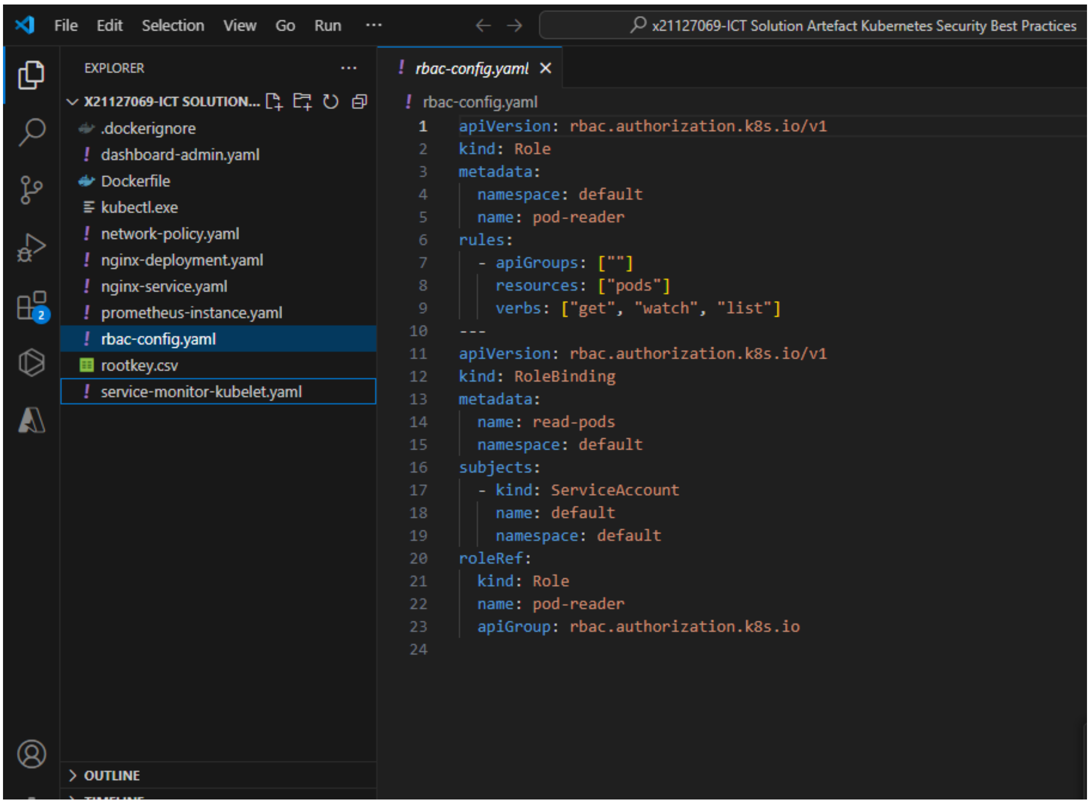
3. The IAM role of the cluster, status and EKS control plane software version are present the other illustrates the use of a RBAC (Role-Based Access Control) configuration applied to a Kubernetes cluster via kubectl command.

8 Securing Container Images



```
}
[cloudshell-user@ip-10-132-84-233 ~]$ docker push 221082195851.dkr.ecr.ap-south-1.amazonaws.com/my-nginx-repo:nginx
The push refers to repository [221082195851.dkr.ecr.ap-south-1.amazonaws.com/my-nginx-repo]
16d387d1e121: Pushed
d77616dbf44e: Pushed
1013a9d59e8c: Pushed
3f49e906ca03: Pushed
fb34e34b4693: Pushed
93a30ca6feb4: Pushed
c3548211b826: Pushed
nginx: digest: sha256:c20cc0fb2627491f6d0c6eb08eb3a0f892c26effd559607742128b65f11436e1 size: 1778
[cloudshell-user@ip-10-132-84-233 ~]$ vi nginx-deployment.yaml
[cloudshell-user@ip-10-132-84-233 ~]$ kubectl apply -f nginx-deployment.yaml
Error from server (BadRequest): error when creating "nginx-deployment.yaml": Deployment in version "v1" cannot be handled as a Deployment: strict decoding error: unknown field "metadata.spec"
[cloudshell-user@ip-10-132-84-233 ~]$ 
```

Figure 5: Securing Container Images

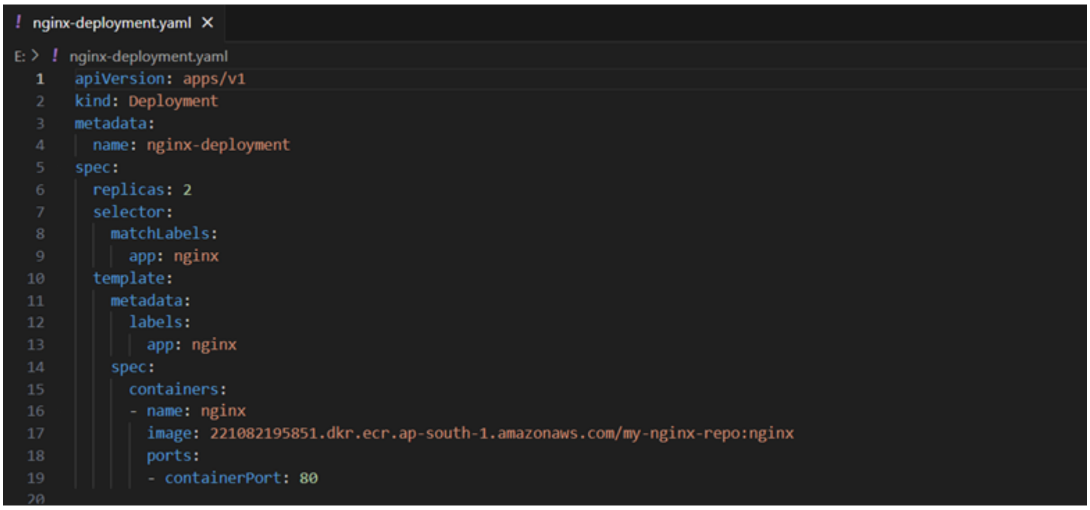


```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "watch", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
  - kind: ServiceAccount
    name: default
    namespace: default
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

Figure 6: RBAC-config.yaml

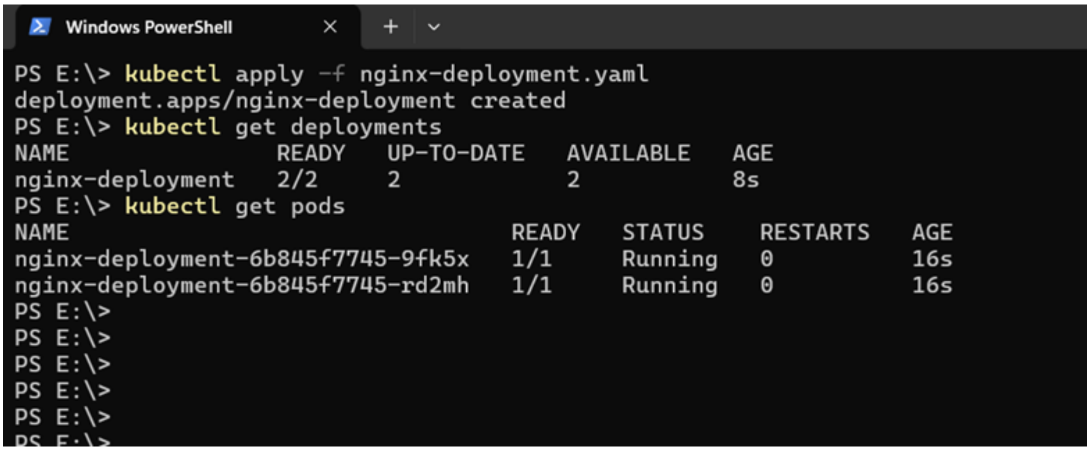
1. The image 5 provides us details about an AWS CloudShell environment with info for EKS (Amazon Elastic Kubernetes Service) cluster with name: ap-south-1.
2. It shows details like IAM role associated with the cluster, status, and version of EKS control plane software.
3. The image 5 shows how a RBAC (RoleBased Access Control) configuration is applied to a kubernetes cluster with the kubectl command.
4. Specifically, it applies a file "rbac-config.yaml" in YAML format. yaml" specifying roles with their associated permissions and role bindings connecting users (or service accounts) to those roles 6. "Bash Command : kubectl apply -f nginx-service.yaml"
5. Figure 6, we are pushing a container image to the AWS Elastic Container Registry (ECR) using the Docker CLI. It contains commands to log in to ECR, get token and pushes the image to ECR Repo.

9 Deploying Applications



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: 221082195851.dkr.ecr.ap-south-1.amazonaws.com/my-nginx-repo:nginx
        ports:
        - containerPort: 80
```

Figure 7: Nginx Deployment.yaml

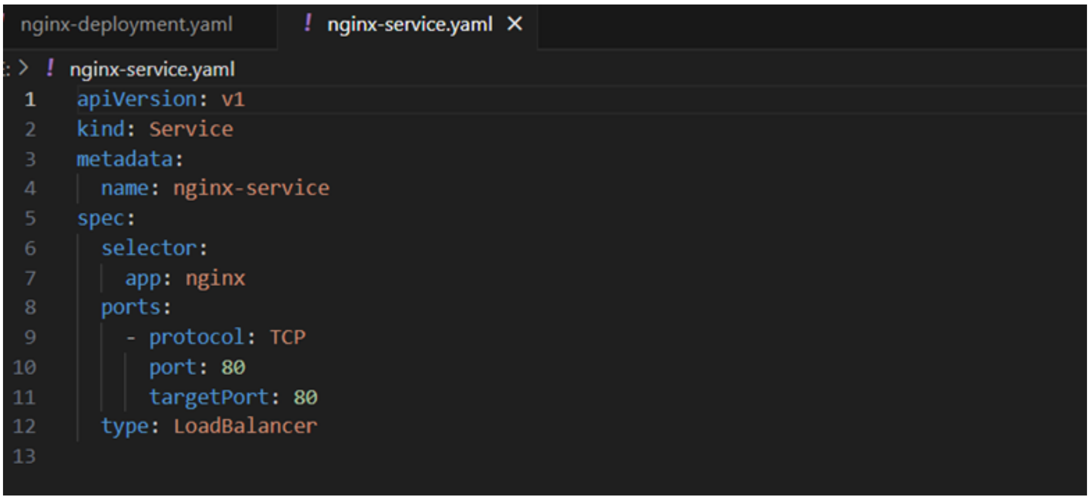


```
PS E:\> kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment created
PS E:\> kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   2/2     2            2           8s
PS E:\> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-6b845f7745-9fk5x   1/1     Running   0          16s
nginx-deployment-6b845f7745-rd2mh   1/1     Running   0          16s
PS E:\>
PS E:\>
PS E:\>
PS E:\>
PS E:\>
PS E:\>
```

Figure 8: Deployment YAML for the Nginx Image

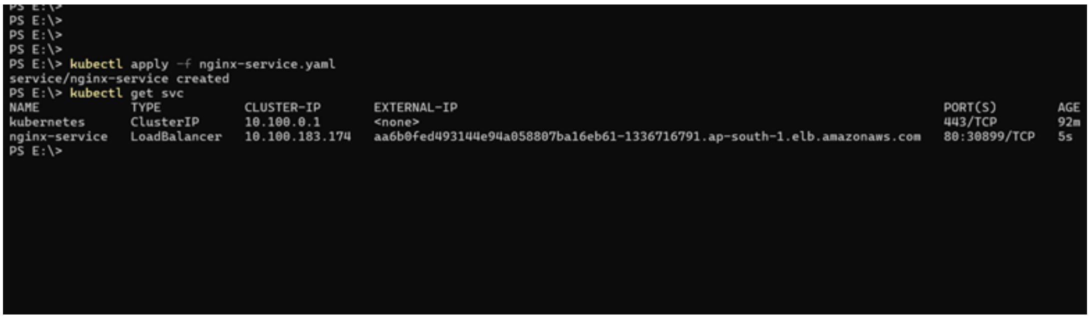
1. The image 7 a YAML configuration file for deploying an NGINX application in a Kubernetes environment. The key details include:
    - API version: ”apps/v1”
    - Kind: Deployment
    - Metadata: Name, Spec, Replicas, Selector, Template (with app label and container image)
    - Port: Container port set to 80
2. The image 8 demonstrates the deployment and verification process using the Kubernetes CLI kubectl. It shows the following:
    - Applying the ”nginx-deployment.yaml” file to create the deployment. ”Use command kubectl apply -f nginx-deployment.yaml”
    - Checking the deployments with their status and available/ready replicas. Use Command ”kubectl get deployment”.
    - Listing the running pods and their status. Using command : ”kubectl get pods”.
    - Port: Container port set to 80.
3. This YAML-based deployment configuration and the use of kubectl commands allow developers to manage the application’s lifecycle within the Kubernetes cluster, ensuring consistent and reliable deployments.

10 Exposing Applications



```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

Figure 9: Nginx Service.yaml



```
PS E:\>
PS E:\>
PS E:\>
PS E:\>
PS E:\> kubectl apply -f nginx-service.yaml
service/nginx-service created
PS E:\> kubectl get svc
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP                                                                PORT(S)        AGE
kubernetes      ClusterIP      10.100.0.1       <none>                                                                     443/TCP        92m
nginx-service   LoadBalancer   10.100.183.174   aa6b0fed493144e94a058807ba16eb61-1336716791.ap-south-1.elb.amazonaws.com   80:30899/TCP   5s
PS E:\>
```

Figure 10: Expose the Nginx Deployment

1. The image 9 shows a YAML configuration file for a Kubernetes Service of type "LoadBalancer". The key details include:

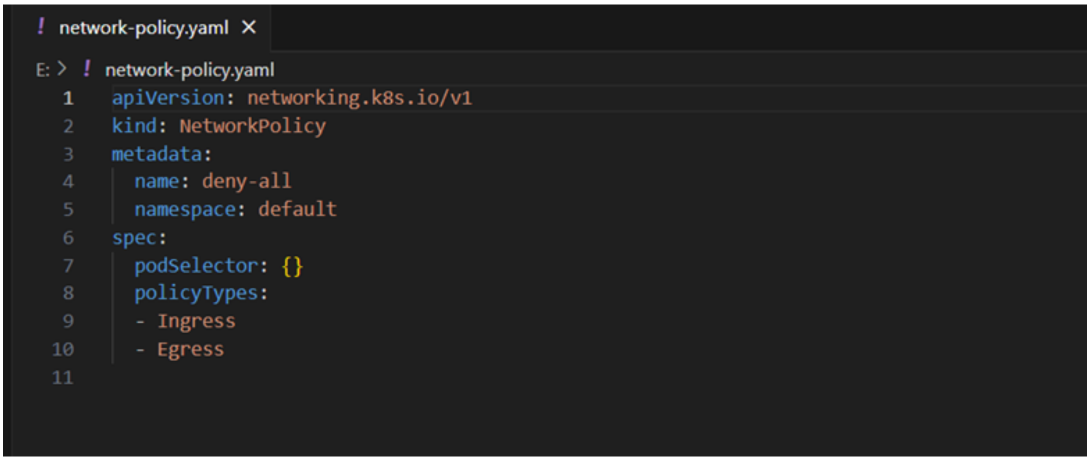
    - API version: "v1"
    - Kind: Service
    - Metadata: Name is "nginx-service"
    - Spec: Selector is "app: nginx", with ports defined for TCP on port 80 and target port 80.
    - Type: LoadBalancer

2. The image 10 demonstrates the creation and verification of this Kubernetes Service using the kubectl command-line tool. It shows the following:

    - Applying the "nginx-service.yaml" file to create the service. Command "kubectl apply -f nginx-service.yaml".
    - Listing the created services, which includes the "nginx-service" with its assigned external IP address.

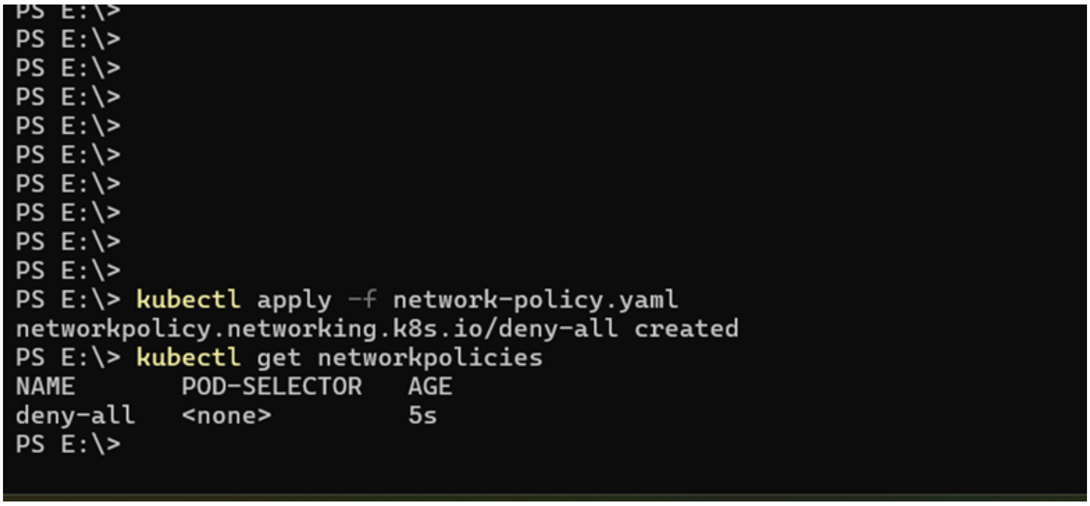
3. By defining the Service as a LoadBalancer type, the application running within the Kubernetes cluster is exposed externally, allowing traffic to reach it through the assigned public IP address. This enables users or other systems to access the application from outside the cluster. Command "kubectl get svc".

11 Configuring Network Policies



```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all
  namespace: default
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

Figure 11: Network-Policy.yaml



```
PS E:\>
PS E:\>
PS E:\>
PS E:\>
PS E:\>
PS E:\>
PS E:\>
PS E:\>
PS E:\>
PS E:\>
PS E:\> kubectl apply -f network-policy.yaml
networkpolicy.networking.k8s.io/deny-all created
PS E:\> kubectl get networkpolicies
NAME       POD-SELECTOR   AGE
deny-all   <none>         5s
PS E:\>
```

Figure 12: Enable Network Policies

1. The image 11 represents a YAML configuration file for a Kubernetes NetworkPolicy with the API version as networking. k8s.

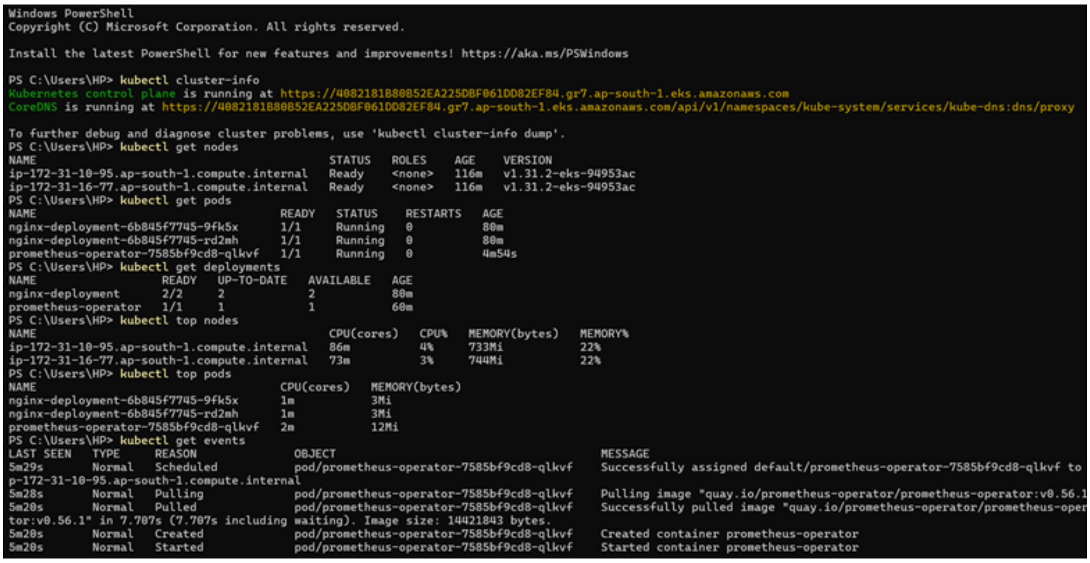
2. The apiVersion here is ”networking.k8s.io/v1” and the kind is NetworkPolicy. It specifies the name as deny-all in metadata and namespace as default.

3. Here, the spec contains a blank podSelector, and set two policyType to Ingress and Egress.

4. The second image shows the applied and verified NetworkPolicy based on above NetworkPolicy using kubectl command line tool.

5. Figure 12 illustrates the use of the ”network-policy yaml” file for creating NetworkPolicy and created NetworkPolicies (shown deny-all policy with nil pod selector, age=5s. A NetworkPolicy can specify ingress and egress to either allow or deny traffic, and by configuring a NetworkPolicy that deny ingress and egress can block all communications to and from the pods.

12 Implementing Real-Time Monitoring



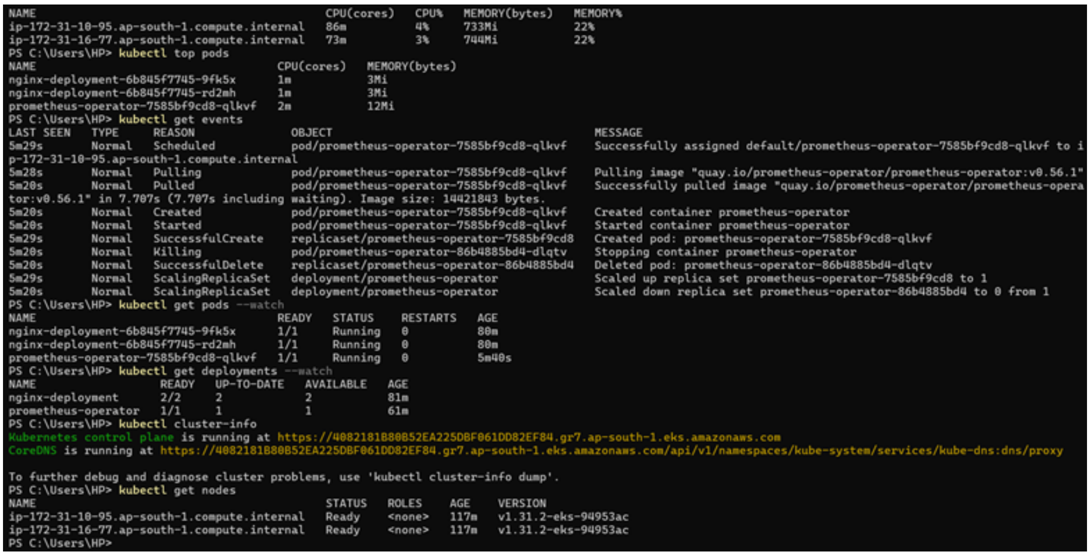
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\HP> kubectl cluster-info
Kubernetes control plane is running at https://4082181B80B52EA225DBF061DD82EF84.gr7.ap-south-1.eks.amazonaws.com
CoreDNS is running at https://4082181B80B52EA225DBF061DD82EF84.gr7.ap-south-1.eks.amazonaws.com/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
PS C:\Users\HP> kubectl get nodes
NAME                                          STATUS   ROLES    AGE    VERSION
ip-172-31-10-95.ap-south-1.compute.internal   Ready    <none>   116m   v1.31.2-eks-94953ac
ip-172-31-16-77.ap-south-1.compute.internal   Ready    <none>   116m   v1.31.2-eks-94953ac
PS C:\Users\HP> kubectl get pods
NAME                                       READY   STATUS    RESTARTS   AGE
nginx-deployment-6b845f7745-9fk5x          1/1     Running   0          80m
nginx-deployment-6b845f7745-rd2mh          1/1     Running   0          80m
prometheus-operator-7585bf9cd8-qlkvf       1/1     Running   0          4m54s
PS C:\Users\HP> kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment      2/2     2            2           80m
prometheus-operator   1/1     1            1           60m
PS C:\Users\HP> kubectl top nodes
NAME                                          CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
ip-172-31-10-95.ap-south-1.compute.internal   86m          4%     733Mi           22%
ip-172-31-16-77.ap-south-1.compute.internal   73m          3%     744Mi           22%
PS C:\Users\HP> kubectl top pods
NAME                                       CPU(cores)   MEMORY(bytes)
nginx-deployment-6b845f7745-9fk5x          1m           3Mi
nginx-deployment-6b845f7745-rd2mh          1m           3Mi
prometheus-operator-7585bf9cd8-qlkvf       2m           12Mi
PS C:\Users\HP> kubectl get events
LAST SEEN   TYPE     REASON              OBJECT                                     MESSAGE
5m29s       Normal   Scheduled           pod/prometheus-operator-7585bf9cd8-qlkvf   Successfully assigned default/prometheus-operator-7585bf9cd8-qlkvf to
p-172-31-10-95.ap-south-1.compute.internal
5m28s       Normal   Pulling             pod/prometheus-operator-7585bf9cd8-qlkvf   Pulling image "quay.io/prometheus-operator/prometheus-operator:v0.56.1
5m20s       Normal   Pulled              pod/prometheus-operator-7585bf9cd8-qlkvf   Successfully pulled image "quay.io/prometheus-operator/prometheus-oper
tor:v0.56.1" in 7.707s (7.707s including waiting). Image size: 14421843 bytes.
5m20s       Normal   Created             pod/prometheus-operator-7585bf9cd8-qlkvf   Created container prometheus-operator
5m20s       Normal   Started             pod/prometheus-operator-7585bf9cd8-qlkvf   Started container prometheus-operator
```

Figure 13: Real-Time Monitoring with Prometheus



```
NAME                                          CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
ip-172-31-10-95.ap-south-1.compute.internal   86m          4%     733Mi           22%
ip-172-31-16-77.ap-south-1.compute.internal   73m          3%     744Mi           22%
PS C:\Users\HP> kubectl top pods
NAME                                       CPU(cores)   MEMORY(bytes)
nginx-deployment-6b845f7745-9fk5x          1m           3Mi
nginx-deployment-6b845f7745-rd2mh          1m           3Mi
prometheus-operator-7585bf9cd8-qlkvf       2m           12Mi
PS C:\Users\HP> kubectl get events
LAST SEEN   TYPE     REASON              OBJECT                                     MESSAGE
5m29s       Normal   Scheduled           pod/prometheus-operator-7585bf9cd8-qlkvf   Successfully assigned default/prometheus-operator-7585bf9cd8-qlkvf to i
p-172-31-10-95.ap-south-1.compute.internal
5m28s       Normal   Pulling             pod/prometheus-operator-7585bf9cd8-qlkvf   Pulling image "quay.io/prometheus-operator/prometheus-operator:v0.56.1"
5m20s       Normal   Pulled              pod/prometheus-operator-7585bf9cd8-qlkvf   Successfully pulled image "quay.io/prometheus-operator/prometheus-opera
tor:v0.56.1" in 7.707s (7.707s including waiting). Image size: 14421843 bytes.
5m20s       Normal   Created             pod/prometheus-operator-7585bf9cd8-qlkvf   Created container prometheus-operator
5m20s       Normal   Started             pod/prometheus-operator-7585bf9cd8-qlkvf   Started container prometheus-operator
5m29s       Normal   SuccessfulCreate    replicaset/prometheus-operator-7585bf9cd8  Created pod: prometheus-operator-7585bf9cd8-qlkvf
5m20s       Normal   Killing             pod/prometheus-operator-86b4885bd4-dlqtv   Stopping container prometheus-operator
5m20s       Normal   SuccessfulDelete    replicaset/prometheus-operator-86b4885bd4  Deleted pod: prometheus-operator-86b4885bd4-dlqtv
5m29s       Normal   ScalingReplicaSet   deployment/prometheus-operator             Scaled up replica set prometheus-operator-7585bf9cd8 to 1
5m20s       Normal   ScalingReplicaSet   deployment/prometheus-operator             Scaled down replica set prometheus-operator-86b4885bd4 to 0 from 1
PS C:\Users\HP> kubectl get pods --watch
NAME                                       READY   STATUS    RESTARTS   AGE
nginx-deployment-6b845f7745-9fk5x          1/1     Running   0          80m
nginx-deployment-6b845f7745-rd2mh          1/1     Running   0          80m
prometheus-operator-7585bf9cd8-qlkvf       1/1     Running   0          5m40s
PS C:\Users\HP> kubectl get deployments --watch
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment      2/2     2            2           81m
prometheus-operator   1/1     1            1           61m
PS C:\Users\HP> kubectl cluster-info
Kubernetes control plane is running at https://4082181B80B52EA225DBF061DD82EF84.gr7.ap-south-1.eks.amazonaws.com
CoreDNS is running at https://4082181B80B52EA225DBF061DD82EF84.gr7.ap-south-1.eks.amazonaws.com/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
PS C:\Users\HP> kubectl get nodes
NAME                                          STATUS   ROLES    AGE    VERSION
ip-172-31-10-95.ap-south-1.compute.internal   Ready    <none>   117m   v1.31.2-eks-94953ac
ip-172-31-16-77.ap-south-1.compute.internal   Ready    <none>   117m   v1.31.2-eks-94953ac
PS C:\Users\HP>
```

Figure 14: Real-Time Monitoring with Prometheus

1. The image 13 provides the information of one AWS EKS (Elastic Kubernetes Service) cluster, which we can verify also that one of, Node, Deployment and Pods are running in that.

2. This includes information on node status, roles, versions, and the resource utilization metrics.

3. The second image has more specific information about the cluster, including details of individual pod and deployment.

4. It displays metrics such as CPU, memory for the individual pods and deployments and various events related to the Prometheus operator that is being used to monitor the cluster.

5. Figure 14 together shows Usage of Kubernetes tools and command to fetch the state and performance of the EKS cluster.

6. The collected data may be utilized by administrators, for the purpose of keeping track of the health and resource use of the Kubernetes environment, providing increased operational visibility and visibility into any problems that may arise.

7. Commands:

    - kubectl cluster-info
    - kubectl get nodes
    - kubectl get deployments
    - kubectl top nodes
    - kubectl get events
    - kubectl get pods –watch

13 Troubleshooting

1. Logs are one of the most important things to check when troubleshooting a Kubernetes environment to identify any errors or unexpected behavior in the system.

2. If a pod crashes due to an error or some kind of misconfiguration, in the output window you can only see the status of the pod and not what actually happens inside the pod, so we can use the kubectl logs command to see what actually happened inside the pod.

3. The other verification that is carried out is of the resources inside the cluster by using kubectl get deployments and kubectl get pods, checking whether all the deployment are running as expected and no pods are in a stuck or failing state.

4. Also, by checking network policies your cluster might have with kubectl get networkpolicies, it can reveal problems with network restrictions or misconfigurations.

5. The majority of problems can be solved with some level of efficiency by logging systemically now to troubleshoot logs, resources and network settings.