Kubernetes on AWS EKS Technical Whitepaper

Contents

1 Deploying Mobius Stack on AWS EKS........ 2

Prerequisites........ 2

Downloading Docker images and tools........ 2

Setting up your environment........ 3

Deploying EKS cluster........ 3

Granting S3 permissions........ 5

Configuring EFS Cluster........ 5

OIDC Authentication and Authorization for Mobius View........ 10

OIDC Authentication and Authorization for Mobius View with NPM........ 10

OIDC Authentication and Authorization for Mobius View without NPM........ 17

Deploying Kubernetes NGINX Ingress Controller........ 27

Installing using Helm........ 28

Configuring TLS certificates for Ingress Controller........ 28

Deploying Mobius........ 28

Prerequisites........ 28

Deploying Mobius Server........ 30

Deploying Mobius View........ 32

Testing the Deployment........ 41

1 Deploying Mobius Stack on AWS EKS

This blueprint describes the deployment of Mobius stack on AWS EKS.

Prerequisites

Before starting the deployment, ensure that the following components are installed and configured:

- Create an AWS account with admin permissions. For more information on managing permissions in AWS, see Manage IAM Permissions.
- The OIDC provider (Keycloak) must be deployed and visible in AWS EKS environment.

  Note: You can deploy the OIDC provider either on a standalone EC2 instance in the same VPC as the EKS cluster or as a Kubernetes pod within the EKS cluster.

- The following software tools must be available.
  - AWS CLI - Installing or updating the latest version of the AWS CLI

    Note: AWS CLI should be configured to access the AWS account that has administrative permissions to deploy the required resources.

  - eksctl - Installing or updating eksctl
  - kubectl - Installing or updating kubectl
  - Helm3 - Installing Helm
- The following AWS resources must be available for successful implementation of the Mobius stack deployment.
  - Create S3 buckets to store archives. For information on creating S3 buckets, see Creating a bucket.
  - Install Postgres database and create an RDS Aurora Postgres DB with required permissions to access it from AWS EKS cluster as described in Creating an Amazon Aurora DB cluster.
  - Create EFS to store NFS persistent volumes as described in Creating Amazon EFS file systems.
  - Create private ECR Repositories named mobius-server and mobius-view to store the Mobius View and Mobius Server docker images. For more information on creating ECR repositories, see Creating a private repository.

Downloading Docker images and tools

Retrieve the Mobius Docker images and push them to your Elastic Container Registry.

For information on retrieving the Mobius View and Mobius Server docker images, see *Deploying Mobius on Kubernetes in Mobius Server documentation*.

For information on pushing docker images to ECR, see Pushing a Docker image.

Setting up your environment

Deploying EKS cluster

There are a variety of methods to deploy an EKS cluster. This section describes a simple deployment using the eksctl utility. For more information on various methods for deploying an EKS cluster, see Getting started with Amazon EKS.

The eksctl utility configures an EKS cluster and uses AWS cloud formation to create the necessary components. You can add `--dry-run` to the command to simulate the command and output the manifest that it generates for the cluster so that you can track the steps that are executed.

A sample --dry-run output is shown below:

```
$ eksctl create cluster --name=mobius --nodes=2 --region=us-east-1 --
instance-types=t2.2xlarge --dry-run
apiVersion: eksctl.io/v1alpha5
availabilityZones:
- us-east-1b
- us-east-1d
cloudWatch:
  clusterLogging: {}
iam:
  vpcResourceControllerPolicy: true
  withOIDC: false
kind: ClusterConfig
kubernetesNetworkConfig:
  ipFamily: IPv4
managedNodeGroups:
  - amiFamily: AmazonLinux2
    desiredCapacity: 2
    disableIMDSv1: false
    disablePodIMDS: false
    iam:
      withAddonPolicies:
      albIngress: false
      appMesh: false
      appMeshPreview: false
      autoScaler: false
      awsLoadBalancerController: false
      certManager: false
      cloudWatch: false
      ebs: false
      efs: false
      externalDNS: false
      fsx: false
      imageBuilder: false
      xRay: false
...
```

Creating EKS cluster

1. Create a EKS cluster with the following eksctl command:

```
$ eksctl create cluster --name=mobius --nodes=2 --region=us-east-1 --
instance-types=t2.2xlarge
2022-05-05 20:59:14 [?] eksctl version 0.95.0
2022-05-05 20:59:14 [?] using region us-east-1
2022-05-05 20:59:14 [?] setting availability zones to [us-east-1b us-
east-1a]
...
2022-05-05 20:59:14 [?] you can enable it with 'eksctl utils update-
cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)}
 --region=us-east-1 --cluster=mobius'
2022-05-05 20:59:14 [?]
2 sequential tasks: { create cluster control plane "mobius",
    2 sequential sub-tasks: {
        wait for control plane to become ready,
        create managed nodegroup "ng-8474a47f",
    }
}
2022-05-05 20:59:14 [?] building cluster stack "eksctl-mobius-cluster"
2022-05-05 20:59:15 [?] deploying stack "eksctl-mobius-cluster"
2022-05-05 20:59:45 [?] waiting for CloudFormation stack "eksctl-mobius-
cluster"
...
2022-05-05 21:09:16 [?] waiting for CloudFormation stack "eksctl-mobius-
cluster"
2022-05-05 21:11:17 [?] building managed nodegroup stack "eksctl-mobius-
nodegroup-ng-8474a47f"
2022-05-05 21:11:18 [?] deploying stack "eksctl-mobius-nodegroup-
ng-8474a47f"
2022-05-05 21:11:18 [?] waiting for CloudFormation stack "eksctl-mobius-
nodegroup-ng-8474a47f"
...
```

```
2022-05-05 21:14:31 [?] kubectl command should work with "C:\\Users\
\test\\.kube\\config", try 'kubectl get nodes'
2022-05-05 21:14:31 [?] EKS cluster "mobius" in "us-east-1" region is
 ready
```

This command creates an EKS cluster in the us-east-1 region with two t2.2xlarge ec2 nodes.

> **Note:** There are many parameters that can be configured for this command. For more information on the eksctl utility, see Introduction. The eksctl utility configures your kubectl configuration one completed to be able to connect to the EKS cluster.

2. You can verify the EKS cluster using the folowing kubectl get nodes command:

```
$ kubectl get nodes -o wide
NAME         STATUS ROLES    AGE VERSION             INTERNAL-IP
 EXTERNAL-IP  OS-IMAGE   KERNEL-VERSION                CONTAINER-RUNTIME
<yours here> Ready  <none>   17h v1.22.6-eks-7d68063 192.168.18.198
 <yours here> Amazon Linux  2 5.4.188-104.359.amzn2.x86_64
 docker://20.10.13
<your here>  Ready  <none>   17h v1.22.6-eks-7d68063 192.168.36.12
  <yours here> Amazon Linux  2 5.4.188-104.359.amzn2.x86_64
 docker://20.10.13
```

Granting S3 permissions

Grant S3 bucket permissions for the s3 bucket resource you have defined to use with Mobius. Grant the S3 permissions to the eksctl-[cluster name]-nodegroup-ng-XXXXX-NodeInstanceRole-XXXXXXXXXXXX role.

Configuring EFS Cluster

This section describes the steps for creating an EFS file system and configuring the EKS cluster to mount Kubernetes persistent volumes to that EFS file system.

Creating EFS file system

After creating the EKS cluster, create an EFS file system to attach to the EKS cluster.

1. Create the EFS file system in the same VPC as the EKS cluster.

> **Note:** You can find the VPC either by using the AWS console or AWS CLI to identify the VPC used by EKSCTL when creating the cluster.

2. Obtain the VPC ID for the cluster through AWS CLI using the following command.

```
$ aws eks describe-cluster --name <your cluster name> | grep v
pcId
          "vpcId": "vpc-038132e83b22b2cd6",
```

3. Navigate through the EC2 security groups to find the security group that belongs to the cluster using the AWS console.

    Note: The cluster name in the security group must be of the format: `eks-cluster-sg-<cluster name>-....`

4. Get the security group for the cluster nodes through the AWS CLI using the following command.

```
$ aws ec2 describe-security-groups --filters Name=group-
name,Values=eks-cluster-sg-mobius2* --query "SecurityGroups[*].
{Name:GroupName,ID:GroupId}"
[
    {
        "Name": "eks-cluster-sg-mobius-1225538347",
        "ID": "sg-0ba9d4f9342375b45"
    }
]
```

Note: Use the Cluster VPC and the security group for the cluster nodes to add to the Mount Targets of the EFS file system, when creating the cluster.

For more information on creating the EFS file system, see https://docs.aws.amazon.com/efs/latest/ug/gs-step-two-create-efs-resources.htmland https://docs.aws.amazon.com/efs/latest/ug/manage-fs-access.html.

Note: The EFS file system ID is required for performing the next steps to configure the cluster to access the EFS file system.

Configuring EFS CSI driver

Kubernetes must be configured to access EFS for shared storage between Kubernetes nodes. For more information regarding setting up and configuring EFS usage in EKS using the EFS CSI driver see, https://docs.aws.amazon.com/eks/latest/userguide/efs-csi.html.

Associating an IAM OIDC Provider

To access an AWS EFS file system using service accounts, the EKS cluster requires IAM roles for service accounts, and an IAM OIDC provider must exist for your cluster.

```
$ eksctl utils associate-iam-oidc-provider --cluster=mobius --approve --
region=us-east-1
2022-05-10 21:21:51 [?] eksctl version 0.96.0
2022-05-10 21:21:51 [?] using region us-east-1
2022-05-10 21:21:51 [?] will create IAM Open ID Connect provider for
 cluster "mobius" in "us-east-1"
2022-05-10 21:21:51 [?] created IAM Open ID Connect provider for cluster
 "mobius" in "us-east-1"
```

Creating an IAM Policy and Service Account

For Kubernetes to access AWS resources like EFS, IAM roles and policies must be associated with Kubernetes service accounts.

Follow the instructions provided in the below topics.

- Creating IAM policy on page 7
- Creating IAM service account on page 7

Creating IAM policy

1. Create an iam-policy json file with the permissions to be associated with the IAM role. You can download an example json file and modify as per your requirements using the following command.

```
curl -o iam-policy-example.json https://raw.githubusercontent.com/
kubernetes-sigs/aws-efs-csi-driver/v1.3.7/docs/iam-policy-example.json
```

2. Create the IAM policy in AWS based on your .json file using the `aws iam create-policy` command as shown below.

```
$ aws iam create-policy
--policy-name AmazonEKS_EFS_CSI_Driver_Policy --policy-document file://
iam-policy-example.json
{
    "Policy": {
        "PolicyName": "AmazonEKS_EFS_CSI_Driver_Policy",
        "PolicyId": "ANPAV5CIUGJWU47J5PLHW",
        "Arn": "arn:aws:iam::<your account>:policy/
AmazonEKS_EFS_CSI_Driver_Policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2022-05-10T20:27:05+00:00",
        "UpdateDate": "2022-05-10T20:27:05+00:00"
    }
}
```

Creating IAM service account

1. Create the service account to be used to access EFS file system using the following command.

```
eksctl create iamserviceaccount \
    --cluster <your cluster name> \
    --namespace kube-system \
    --name efs-csi-controller-sa \
    --attach-policy-arn arn:aws:iam::<your account>:policy/
AmazonEKS_EFS_CSI_Driver_Policy \
    --approve \
    --region <regions code>
```

Note: Replace <*your cluster name*> with the name of the cluster you have deployed, and <*your account*> with your actual AWS account ID that the Kubernetes system is running on.

Example:

```
$ eksctl create iamserviceaccount --cluster mobius --namespace
 kube-system --name efs-csi-controller-sa --attach-policy-arn
 arn:aws:iam::<your account>:policy/AmazonEKS_EFS_CSI_Driver_Policy --
approve --region us-east-1
2022-05-10 21:22:03 [?] eksctl version 0.96.0
2022-05-10 21:22:03 [?] using region us-east-1
2022-05-10 21:22:04 [?] 1 iamserviceaccount (kube-system/efs-csi-
controller-sa) was included (based on the include/exclude rules)
2022-05-10 21:22:04 [!] serviceaccounts that exist in Kubernetes will be
 excluded, use --override-existing-serviceaccounts to override
2022-05-10 21:22:04 [?] 1 task: {
    2 sequential sub-tasks: {
      create IAM role for serviceaccount "kube-system/efs-csi-
controller-sa",
      create serviceaccount "kube-system/efs-csi-controller-sa",
    } }2022-05-10 21:22:04 [?] building iamserviceaccount stack "eksctl-
mobius-addon-iamserviceaccount-kube-system-efs-csi-controller-sa"
2022-05-10 21:22:04 [?] deploying stack "eksctl-mobius-addon-
iamserviceaccount-kube-system-efs-csi-controller-sa"
2022-05-10 21:22:04 [?] waiting for CloudFormation stack "eksctl-mobius-
addon-iamserviceaccount-kube-system-efs-csi-controller-sa"
2022-05-10 21:22:34 [?] waiting for CloudFormation stack "eksctl-mobius-
addon-iamserviceaccount-kube-system-efs-csi-controller-sa"
2022-05-10 21:22:34 [?] created serviceaccount "kube-system/efs-csi-
controller-sa"
```

Deploying CSI driver

Perform the following commands to deploy the AWS CSI driver.

```
$ helm repo add aws-efs-csi-driver https://kubernetes-sigs.github.io/
aws-efs-csi-driver/
"aws-efs-csi-driver" has been added to your repositories

$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "aws-efs-csi-driver" chart
 repository
Update Complete. ? Happy Helming!?

$ helm upgrade -i aws-efs-csi-driver aws-efs-csi-driver/aws-efs-csi-
driver \
> --namespace kube-system \
> --set image.repository=602401143452.dkr.ecr.us-east-1.amazonaws.com/
eks/aws-efs-csi-driver \
> --set controller.serviceAccount.create=false \
> --set controller.serviceAccount.name=efs-csi-controller-sa
Release "aws-efs-csi-driver" does not exist. Installing it now.
NAME: aws-efs-csi-driver
LAST DEPLOYED: Tue May 10 21:26:07 2022
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
To verify that aws-efs-csi-driver has started, run:

kubectl get pod -n kube-system -l "app.kubernetes.io/name=aws-efs-csi-
driver,app.kubernetes.io/instance=aws-efs-csi-driver"

$ kubectl get pod -n kube-system -l "app.kubernetes.io/name=aws-efs-csi-
driver,app.kubernetes.io/instance=aws-efs-csi-driver"
NAME READY STATUS RESTARTS AGE
efs-csi-controller-5b6b57569b-2s8r8 3/3 Running 0 57s
efs-csi-controller-5b6b57569b-lgf2t 3/3 Running 0 57s
efs-csi-node-2zrx7 3/3 Running 0 57s
efs-csi-node-6qqgr 3/3 Running 0 57s
```

Creating storage class

For EFS to be utilized, a Kubernetes storage class must be defined to access the AWS EFS file system. A sample `storageclass.yaml` file is provided below. Edit this file based on your requirement.

Note: Replace <*your EFS fs ID*> with the actual EFS file system ID you created above.

For more information on creating storage classes and using EFS CSI driver, see https://github.com/kubernetes-sigs/aws-efs-csi-driver/tree/master/examples/kubernetes.

storageclass.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: efs.csi.aws.com
parameters:
  provisioningMode: efs-ap
  fileSystemId: <your EFS fs id>
  directoryPerms: "700"
  gidRangeStart: "1000" # optional
  gidRangeEnd: "2000" # optional
  basePath: "/nfs" # optional
```

You can create the storage class using the following command.

```
kubectl apply -f storageclass.yaml
```

OIDC Authentication and Authorization for Mobius View

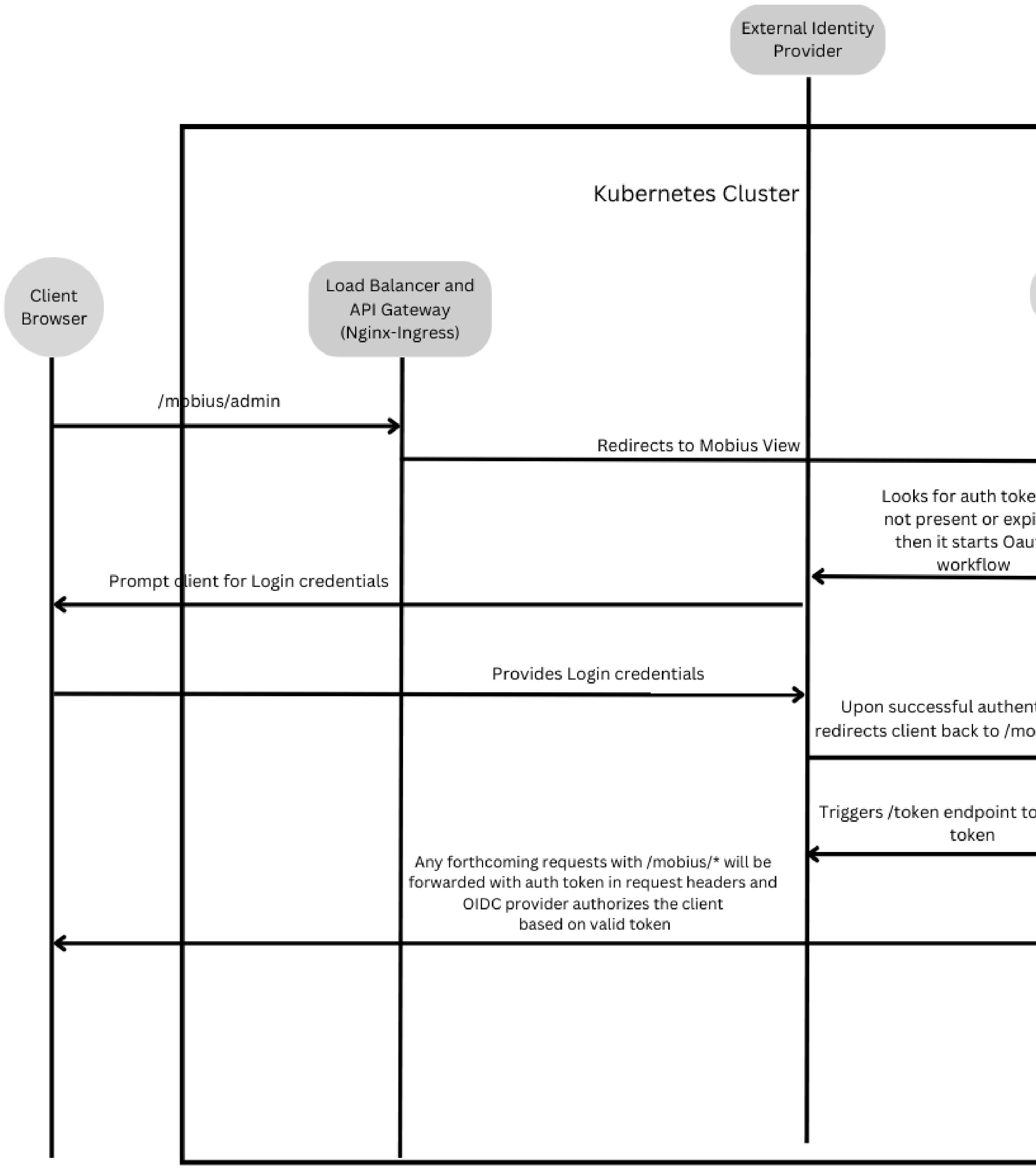
Details about the required configurations to be made during Mobius deployment on Kubernetes platform to use OAUTH2-PROXY as a backend OIDC provider instead of a NPM client for authentication or authorization.

- OIDC Authentication and Authorization for Mobius View with NPM on page 10
- OIDC Authentication and Authorization for Mobius View without NPM on page 17

OIDC Authentication and Authorization for Mobius View with NPM

In this scenario, API gateway, Mobius View, and other Mobius components are available within a Kubernetes cluster and authorization server (Keycloak) is outside the Kubernetes cluster and the user request goes through API gateway.

Authorization Flow


External Identity Provider
Kubernetes Cluster
Client Browser
Load Balancer and API Gateway (Nginx-Ingress)
/mobius/admin
Redirects to Mobius View
Looks for auth toke
not present or expi
then it starts Oau
workflow
Prompt client for Login credentials
Provides Login credentials
Upon successful authen
redirects client back to /mo
Triggers /token endpoint to
token
Any forthcoming requests with /mobius/* will be forwarded with auth token in request headers and OIDC provider authorizes the client based on valid token

To enable OIDC authentication and authorization for Mobius View with NPM,

1. Get or create self-signed certificates that have the hostname for the load balancer that will be used to route to the Kubernetes cluster. Refer TLS Secrets, for more information on creating certificates.
2. Create a config map using the generated .crt file to import the .crt file into the JVM truststore of the Mobius View container.

```
kubectl -n <NAMESPACE> create configmap customcert --from-file=tls.crt
```

3. Deploy Mobius View using `mobiusview.yaml` for helm containing overriding values with relevance to OIDC NPM disabled.

```
helm install mobius-view -n <NAMESPACE> mobiusview.tgz -f
 <mobiusview>.yaml
```

The `<mobiusview>.yaml` file is as follows:

- **Mobius View 12.1 and later**

```
service:
#name is the name of the port
 type: ClusterIP
#name is the name of the port
 port: 8080

asg:
  vendor:
    type: ASG
...
  security:
    enabled: true
    openidConnectTwo:
      enabled: true
      validate: true
      client:
       jwkUrl: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms/<your_security_realm>/protocol/openid-
connect/certs"
# Either provide jwkUrl or jwkBase
      #jwkbase: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms"
      serviceTokenSettings:
        enabled: false
      mapping:
        username: "preferred_username"
        groups: "groups"
    oidc:
      npm:
        enabled: true
        oidcconfig:
          issuer: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms/<your_security_realm>"
          redirecturi: ""
          clientid: "<your_client_ID>"
          responsetype: "code"
          scope: "openid profile email"
          logouturl: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms/<your_security_realm>/protocol/openid-
connect/logout"
          keycloak: true

additionalVolumes:
  - name: customcert
    configMap:
      name: customcert
additionalVolumeMounts:
  - name: customcert
    mountPath: /etc/pki/tls/custom/tls.crt
    subPath: tls.crt
    readOnly: false
```

– **Mobius View 12.0 and earlier**

```
service:
#name is the name of the port
 type: ClusterIP
#name is the name of the port
 port: 8080

asg:
  vendor:
    type: "ASG"
...
  security:
    enabled: true
    openidConnectTwo:
      enabled: true
      validate: true
      client:
        jwkUrl: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms/<your_security_realm>/protocol/openid-
connect/certs"
# Either provide jwkUrl or jwkBase
        #jwkbase: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms"
      serviceTokenSettings:
        enabled: false
    oidc:
      npm:
        enabled: true
        oidcconfig:
          issuer: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms/<your_security_realm>"
          redirecturi: ""
          clientid: "<your_client_ID>"
          responsetype: "code"
          scope: "openid profile email"
          logouturl: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms/<your_security_realm>/protocol/openid-
connect/logout"
          keycloak: true

additionalVolumes:
  - name: customcert
    configMap:
      name: customcert
additionalVolumeMounts:
  - name: customcert
    mountPath: /etc/pki/tls/custom/tls.crt
    subPath: tls.crt
    readOnly: false
```

4. Do the following to configure external OIDC service and deploy nginx-ingress.
    a. Create a service for external OIDC provider using the below command.

```
kubectl apply -f -n <NAMESPACE> <external_oidc>.yml
```

The `<external_oidc>.yml` file is as follows:

```
---
kind: "Service"
apiVersion: "v1"
metadata:
  name: "externaloidc"
```

```
spec:
  ports:
  - name: http
    port: 8282
    protocol: TCP
    targetPort: 8282
    nodePort: 0
  - name: https
    port: 9292
    protocol: TCP
    targetPort: 9292
    nodePort: 0
---
kind: "Endpoints"
apiVersion: "v1"
metadata:
  name: "externaloidc"
subsets:
  - addresses:
     - ip: "[IP Address of your External OIDC provider]"
    ports:
      - name: http
        port: 8282
      - name: https
        port: 9292
```

b. Create kube secret containing TLS information using the below command.

```
kubectl create secret tls nginx-cert --key tls.key --cert tls.crt -n
 <NAMESPACE>
```

c. Deploy nginx-ingress with default ssl certifcate pointing to kube secret containing TLS information using the below command.

```
helm upgrade --install ingress-nginx ingress-nginx \
 --repo https://kubernetes.github.io/ingress-nginx \
 --namespace ingress-nginx --create-namespace \
 --set controller.extraArgs.default-ssl-certificate=<NAMESPACE>/
<secret_name>
```

d. Run the below command to apply ingress rules for Mobius View, external OIDC provider, and oauth2-proxy.

```
kubectl apply -f <ingress>.yaml -n <NAMESPACE>
```

The `<ingress>.yaml` file is as follows:

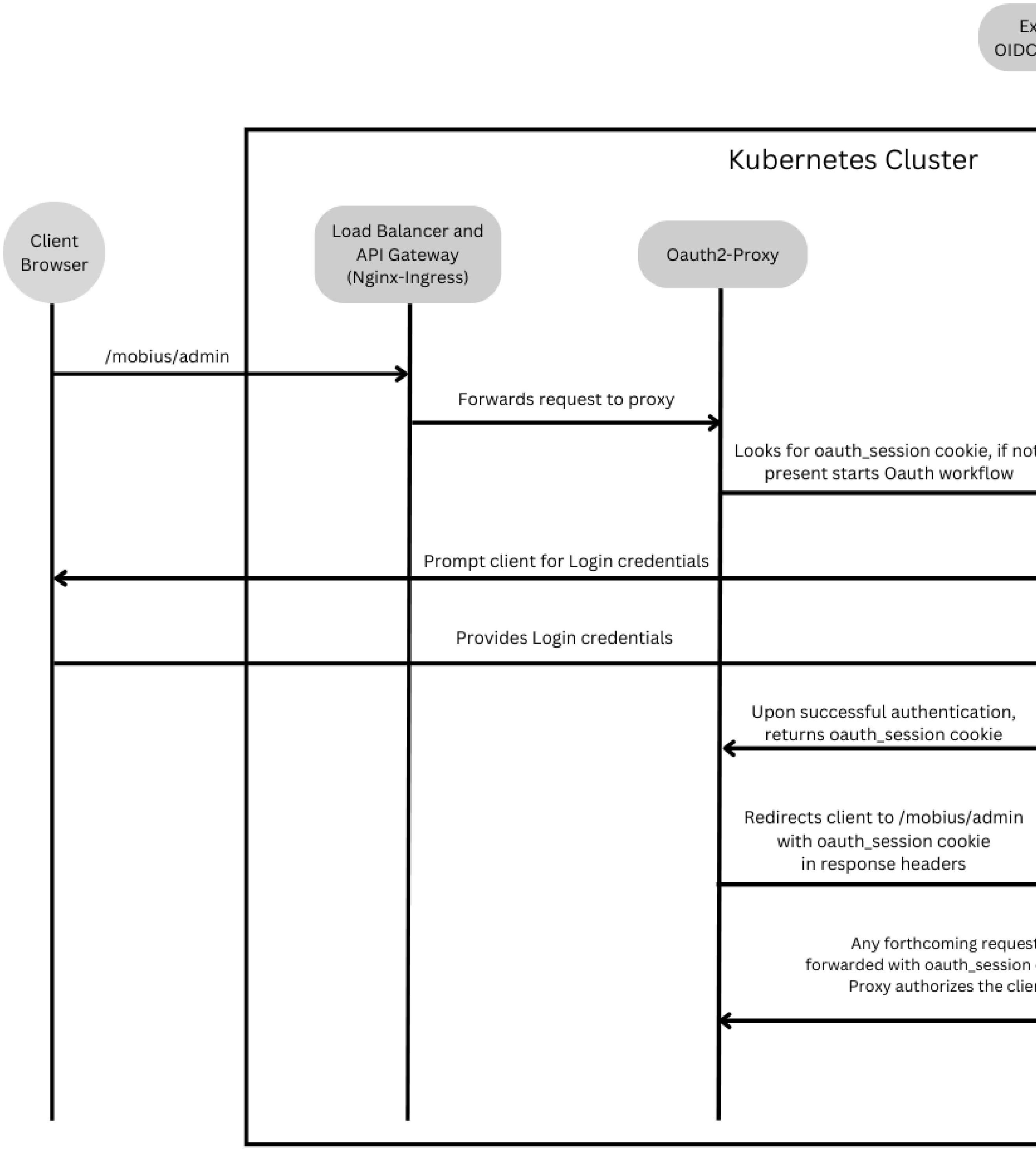
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: authservice
  namespace: <NAMESPACE>
  annotations:
    nginx.ingress.kubernetes.io/proxy-buffer-size: "64k"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"

spec:
  ingressClassName: nginx
  rules:
  - host: <LB_HOST or LOCAL_MACHINE_HOST>
    http:
      paths:
      - path: /auth
        pathType: Prefix
        backend:
          service:
            name: externaloidc
            port:
              number: 9292
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mobiusingress
  namespace: <NAMESPACE>
  annotations:
    nginx.ingress.kubernetes.io/proxy-buffer-size: "64k"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  ingressClassName: nginx
  rules:
  - host: <LB_HOST or LOCAL_MACHINE_HOST>
    http:
      paths:
      - backend:
          service:
              name: mobius-view-mobiusview
              port:
                number: 8080
          path: /mobius
          pathType: Prefix
tls:
  - hosts:
    - <LB_HOST or LOCAL_MACHINE_HOST>
    secretName: <SECRET_CONTAINING_CERTIFICATE>
```

OIDC Authentication and Authorization for Mobius View without NPM

In this scenario, API gateway, Mobius View, oauth2-proxy, and other Mobius components are available within a Kubernetes cluster and authorization server (Keycloak) is outside the Kubernetes cluster and the user request is authenticated at the API gateway through oauth2-proxy.

Authorization Flow


Ex
OIDC
Kubernetes Cluster
Client Browser
Load Balancer and API Gateway (Nginx-Ingress)
Oauth2-Proxy
/mobius/admin
Forwards request to proxy
Looks for oauth_session cookie, if not present starts Oauth workflow
Prompt client for Login credentials
Provides Login credentials
Upon successful authentication, returns oauth_session cookie
Redirects client to /mobius/admin with oauth_session cookie in response headers
Any forthcoming request forwarded with oauth_session Proxy authorizes the clien

To enable OIDC authentication and authorization for Mobius View without NPM after setting up keycloak,

1. Get or create self-signed certificates that have the hostname for the load balancer that will be used to route to the Kubernetes cluster. Refer TLS Secrets, for more information on creating certificates.
2. Create a config map using the generated .crt file to import the .crt file into the JVM truststore of the

Mobius View container.

```
kubectl -n <NAMESPACE> create configmap customcert --from-file=tls.crt
```

3. Deploy Mobius View using `<mobiusview>.yaml` for helm containing overriding values with relevance to OIDC NPM disabled.

```
helm install mobius-view -n <NAMESPACE> mobiusview.tgz -f
 <mobiusview>.yaml
```

The `<mobiusview>.yaml` file is as follows:

Note: `<mobiusview>.yaml` shown below contains only overriding values for the OIDC NPM disabled scenario.

- **Mobius View 12.1 and later**

```
service:
#name is the name of the port
  type: ClusterIP
#name is the name of the port
  port: 8080

asg:
  vendor:
    type: ASG
  security:
    enabled: true
    openidConnectTwo:
      enabled: true
      validate: true
      serviceTokenSettings:
        enabled: false
      mapping:
        username: "preferred_username"
        groups: "groups"
      client:
        jwkUrl: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms/<your_security_realm>/protocol/openid-
connect/certs"
    oidc:
      npm:
        enabled: false
```

```
additionalVolumes:
  - name: customcert
    configMap:
      name: customcert
additionalVolumeMounts:
  - name: customcert
    mountPath: /etc/pki/tls/custom/tls.crt
    subPath: tls.crt
    readOnly: false
```

– **Mobius View 12.0 and earlier**

```
service:
#name is the name of the port
  type: ClusterIP
#name is the name of the port
  port: 8080

asg:
  vendor:
    type: "OIDC"
...
  security:
    openidConnect:
      server:
        enabled: true
      inbound: true
      outbound: true
      urlPatterns: "/rest/*,/adminrest/*,/directconnect/*,/cmis/*"
      idpProvider: external
      claimsMapping:
        username: "preferred_username"
        groups: "groups"
      identityService:
        enabled: false
        profile: "http://identity-asg-identity-service.shared/api/Users/
profile"
        groups: "http://identity-asg-identity-service.shared/api/Users/
groups"
      client:
        jwkUrl: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms/<your_security_realm>/protocol/openid-
connect/certs"
   oidc:
     npm:
       enabled: false
additionalVolumes:
  - name: customcert
    configMap:
      name: customcert
additionalVolumeMounts:
  - name: customcert
    mountPath: /etc/pki/tls/custom/tls.crt
    subPath: tls.crt
    readOnly: false
```

4. Define an external service if the OIDC provider is not deployed in the EKS cluster. This will map the external OIDC provider to the Kubernetes services. The below is a sample code for an external Keycloak deployment.

```
---
kind: "Service"
apiVersion: "v1"
metadata:
  name: "external-oidc-service"
spec:
  type: ClusterIP
  ports:
  - name: https
    port: 8443
    targetPort: 8443
  - name: http
    port: 8282
    targetPort: 8282
---
kind: "Endpoints"
apiVersion: "v1"
metadata:
  name: "external-oidc-service"
subsets:
  - addresses:
     - ip: "[IP Address of your External OIDC provider]"
    ports:
      - name: https
        port: 8443
      - name: http
        port: 8282
```

5. Run the below command to deploy oauth2-proxy.

```
helm upgrade -i -n <NAMESPACE> -f <oauth2proxy-values>.yml oauth2-proxy
 oauth2-proxy/oauth2-proxy
```

The `<oauth2proxy-values>.yml`file is as follows:

```
config:
 clientID: OIDCProxy
 clientSecret: "bTH1CoUZlO7wEPW7Zt9wpmIaWLfA3j9K"
 cookieSecret: "UlpPOE8wWUo3cmZtRGxDRllXUmd2bEhaN1VXcGxFclI="
extraArgs:
 provider: oidc
 oidc-issuer-url: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms/MobiusOIDC"
 login-url: "https://<LB_HOST_for_OIDC_Provider or LOCAL_MACHINE_HOST>/
auth/realms/MobiusOIDC/protocol/openid-connect/auth"
 redeem-url: "https://<LB_HOST_for_OIDC_Provider or LOCAL_MACHINE_HOST>/
auth/realms/MobiusOIDC/protocol/openid-connect/token"
 validate-url: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms/MobiusOIDC/protocol/openid-connect/
userinfo"
 set-xauthrequest: false
 provider-display-name: Login
 cookie-secure: "false"
 email-domain: "\"*\""
 set-authorization-header: "true"
 standard-logging: "true"
 auth-logging: "true"
 request-logging: "true"
 skip-provider-button: "true"
 keycloak-group: "groups"
 whitelist-domain: "<LB_HOST_for_OIDC_Provider or LOCAL_MACHINE_HOST>"
 ssl-insecure-skip-verify: "true"
 insecure-oidc-allow-unverified-email: "true"
 scope: "openid"
```

6. Run the below command to create kube secret containing TLS information.

```
kubectl create secret tls nginx-cert --key tls.key --cert tls.crt -n
 <NAMESPACE>
```

7. Deploy nginx-ingress with default ssl certifcate pointing to kube secret containing TLS information using the below command.

```
helm upgrade --install ingress-nginx ingress-nginx \
 --repo https://kubernetes.github.io/ingress-nginx \
 --namespace ingress-nginx --create-namespace \
 --set controller.extraArgs.default-ssl-certificate=<NAMESPACE>/
<secret_name>
```

8. Apply ingress rules for Mobius View, external OIDC provider, and oauth2-proxy.

```
kubectl apply -f ingress.yaml -n <NAMESPACE>
```

The `ingress.yaml` file is as follows:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: oauth2-proxy
  namespace: <NAMESPACE>
  annotations:
    nginx.ingress.kubernetes.io/proxy-buffer-size: "64k"

spec:
  ingressClassName: nginx
  rules:
  - host: <LB_HOST or LOCAL_MACHINE_HOST>
    http:
      paths:
      - path: /oauth2
        pathType: Prefix
        backend:
          service:
            name: oauth2-proxy
            port:
              number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: authservice
  namespace: <NAMESPACE>
  annotations:
    nginx.ingress.kubernetes.io/proxy-buffer-size: "64k"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  ingressClassName: nginx
  rules:
  - host: <LB_HOST or LOCAL_MACHINE_HOST>
    http:
      paths:
      paths:
      - path: /oauth2
        pathType: Prefix
        backend:
          service:
            name: externaloidc
            port:
              number: 9292
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mobiusingress
  namespace: <NAMESPACE>
  annotations:
    nginx.ingress.kubernetes.io/auth-url: "https://<LB_HOST or
 LOCAL_MACHINE_HOST>/oauth2/auth"
    nginx.ingress.kubernetes.io/auth-signin: "https://<LB_HOST or
 LOCAL_MACHINE_HOST>/oauth2/start?rd==https%3A%2F%2F$host$request_uri"
    nginx.ingress.kubernetes.io/proxy-buffer-size: "64k"
    nginx.ingress.kubernetes.io/auth-response-headers: "Authorization"
spec:
  ingressClassName: nginx
  rules:
  - host: <LB_HOST or LOCAL_MACHINE_HOST>
    http:
      paths:
      - backend:
          service:
            name: mobius-view-mobiusview
            port:
              number: 8080
        path: /mobius
        pathType: Prefix
 tls:
   - hosts:
     - <LB_HOST or LOCAL_MACHINE_HOST>
     secretName: <SECRET_CONTAINING_CERTIFICATE>
```

Execute the below scripts to configure Mobius View for admin endpoint ingress, view ingress, and content streams ingress, respectively.

Configuring Mobius View for admin endpoint ingress

```
kind: Ingress
metadata:
  name: <NAME_TO_USE_FOR_THIS_INGRESS>`
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/auth-url: "https://
<YOUR_OAUTH2_HOSTNAME>:443/oauth2/auth"
    nginx.ingress.kubernetes.io/auth-signin: "https://
<YOUR_OAUTH2_HOSTNAME>:443/oauth2/start rd=$escaped_request_uri"
    nginx.ingress.kubernetes.io/proxy-buffer-size: "64k"
    nginx.ingress.kubernetes.io/auth-response-headers: "Authorization"
spec:
  rules:
  - host: <YOUR_HOSTNAME>
    http:
     paths:
     - path: /mobius/
       pathType: Prefix
       backend:
         service:
           name: <YOUR_MOBIUS_VIEW_SERVICE_NAME>
           port:
             number: 80
```

Configuring Mobius View for view ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <NAME_TO_USE_FOR_THIS_INGRESS>`
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/auth-url: "https://
<YOUR_OAUTH2_HOSTNAME>:443/oauth2/auth"
```

```
    nginx.ingress.kubernetes.io/auth-signin: "https://
<YOUR_OAUTH2_HOSTNAME>:443/oauth2/startrd=$escaped_request_uri"
    nginx.ingress.kubernetes.io/proxy-buffer-size: "64k"
    nginx.ingress.kubernetes.io/auth-response-headers: "Authorization"
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "60"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "60"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "60"
    nginx.ingress.kubernetes.io/configuration-snippet: |
more_set_headers "Content-Security-Policy: connectsrc
https://<YOUR_HOSTNAME>/ blob:; default-src blob: data:
'self'; img-src blob: data: 'self'; script-src 'self' 'sha256-
wSk7Pac68P5NGz0ckYIUSA8nd7eh8zkveKcseL24KB0='; style-src 'self'
 'unsafeinline';";
more_set_headers "X-Content-Type-Options: 'nosniff'";
more_set_headers "X-XSS-Protection: '1; mode=block'";
more_set_headers "X-Frame-Options: 'sameorigin'";
more_set_headers "Strict-Transport-Security: 'max-age=315360000;
includeSubDomains; preload'";
more_set_headers "Access-Control-Allow-Origin: '<YOUR_HOSTNAME>'";
spec:
  rules:
  - host: <YOUR_HOSTNAME>
    http:
     paths:
     - path: /mobius/view/
       pathType: Prefix
       backend:
         service:
           name: <YOUR_MOBIUS_VIEW_SERVICE_NAME>
           port:
             number: 80
```

Configuring Mobius View for content streams ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <NAME_TO_USE_FOR_THIS_INGRESS>`
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/auth-url: "https://
<YOUR_OAUTH2_HOSTNAME>:443/oauth2/auth"
    nginx.ingress.kubernetes.io/auth-signin: "https://
<YOUR_OAUTH2_HOSTNAME>:443/oauth2/startrd=$escaped_request_uri"
```

```
    nginx.ingress.kubernetes.io/proxy-buffer-size: "64k"
    nginx.ingress.kubernetes.io/auth-response-headers: "Authorization"
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "60"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "60"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "60"
    nginx.ingress.kubernetes.io/configuration-snippet: |
more_set_headers "Content-Security-Policy: connectsrc
https://<YOUR_HOSTNAME>/ blob:; default-src blob: data:
'self'; img-src blob: data: 'self'; script-src 'self' 'sha256-
wSk7Pac68P5NGz0ckYIUSA8nd7eh8zkveKcseL24KB0='; style-src 'self'
 'unsafeinline';";
more_set_headers "X-Content-Type-Options: 'nosniff'";
more_set_headers "X-XSS-Protection: '1; mode=block'";
more_set_headers "X-Frame-Options: 'sameorigin'";
more_set_headers "Strict-Transport-Security: 'max-age=315360000;
includeSubDomains; preload'";
more_set_headers "Access-Control-Allow-Origin: '<YOUR_HOSTNAME>'";
spec:
  rules:
  - host: <YOUR_HOSTNAME>
    http:
     paths:
     - path: /mobius/rest/contentstreams
       pathType: Prefix
       backend:
         service:
           name: <YOUR_MOBIUS_VIEW_SERVICE_NAME>
           port:
             number: 80
```

Deploying Kubernetes NGINX Ingress Controller

To route traffic from outside the EKS cluster to pods running on Kubernetes, ingress controller must be used.

This section helps you to deploy the Kubernetes version of the Nginx Ingress Controller. For more information on deploying the Kubernetes NGINX Ingress Controller, see Installation Guide.

Installing using Helm

Creating a namespace

To deploy the Kubernetes NGINX ingress controller in to the `ingress-nginx`namespace, you need to first create the namespace using the following command.

```
$ kubectl create namespace ingress-nginx
namespace/ingress-nginx created
```

Installing helm chart

To install the NGINX Ingress controller execute the following command.

```
helm upgrade --install ingress-nginx ingress-nginx \
  --repo https://kubernetes.github.io/ingress-nginx \
  --namespace ingress-nginx --create-namespace
```

When NGINX deploys, it communicates and configures an external load balancer that can route traffic from the internet to the ingress controller running in this EKS cluster. Setting up ingress rules maps traffic from that load balancer to the configured PDS described in the ingress settings. To limit traffic to this EKS cluster from the internet, you must configure inbound rules in the Security Group of the load balancer or the cluster to whitelist IP addresses or networks you want to provide access.

Configuring TLS certificates for Ingress Controller

For TLS to work properly, the NGINX default certificates must be replaced with the certificates that match your inbound host for the ingress controller in AWS, which is the load balancer name or if a customer DNS is configured then that DNS must be a part of the certificate. Additionally, TLS can be terminated at the load balancer and Kubernetes can work over HTTPS locally.

To create a TLS certificate and store it in a Kubernetes secret, see TLS/HTTPS.

Note: In case of long host names such as those used in AWS load balancers, you must set the name in the SAN section due to size limits for the CN field. After the secret is created, you can update the helm deployment to set the default SSL certificate it uses.

```
helm upgrade --install ingress-nginx ingress-nginx \
 --repo https://kubernetes.github.io/ingress-nginx \
 --namespace ingress-nginx --create-namespace \
 --set controller.extraArgs.default-ssl-certificate=<NAMESPACE>/
<secret_name>
```

Deploying Mobius

Prerequisites

Before deploying Mobius, ensure the following operations are performed.

Creating Mobius namespace

To create a Mobius namespace, execute the following command.

```
$ kubectl create namespace mobius
namespace/mobius created
```

Creating Persistent Volume Claims

A Persistent Volume Claim (PVC) is a request for storage by a pod and can be shared between pods. For Mobius, the following three claims must be defined.

- Persistent Storage - Used for configuration data.
- FTS Storage - Used for full text search indexing data.
- Diagnostic Storage - Used for diagnostic data, such as logs.

mobius-pvc.yaml

```
---
apiVersion: v1

kind: PersistentVolumeClaim
metadata:
  name: mobius
spec:

  accessModes:
    - ReadWriteMany
  storageClassName: efs-sc
  resources:
    requests:
      storage: 5Gi
---
apiVersion: v1

kind: PersistentVolumeClaim
metadata:
  name: mobius-fts
spec:

  accessModes:
    - ReadWriteMany
  storageClassName: efs-sc
  resources:
    requests:
      storage: 5Gi
---
apiVersion: v1

kind: PersistentVolumeClaim
metadata:
  name: mobius-diagnostics
spec:

  accessModes:
    - ReadWriteMany
  storageClassName: efs-sc
  resources:
    requests:
      storage: 5Gi
```

PVCs must be deployed in the namespace in which they will be used. The namespaces must exist prior to creating the PVCs. The following commands create the `mobius` namespace and the three required volume claims.

```
$ kubectl apply -f ~/eks/mobius-pvc.yaml -n mobius
persistentvolumeclaim/mobius created
persistentvolumeclaim/mobius-fts created
persistentvolumeclaim/mobius-diagnostics created
```

Creating Mobius databases in Postgres

For Mobius software to run, a database must be created prior to deployment. Using your database client, create databases for Mobius Server and Mobius View. These will be used in configuring Mobius Server and Mobius View deployments.

Deploying Mobius Server

Use helm to deploy the Mobius Server on Kubernetes. For more information see, *Using Helm chart to deploy Mobius Server in Mobius Server documentation*.

Configuring Mobius Server values file

To deploy Mobius Server you must first configure the parameters for the deployment. The values for the parameters can be set in a `values.yaml` file and passed to the helm deployment tool during deployment. A sample config file is shown below. For more information on this file see, Mobius Server Container Parameters.

Replace the values indicated with the ones from your environment.

values-mobius-server.yaml

```
replicaCount: 1
namespace: mobius
image:
  repository: [Your ECR repository for the Mobius server Docker image]
  tag:  [Your tag for your Mobius Server Docker image in ECR]
  pullPolicy: IfNotPresent
mobius:
  isSaas: "YES"
  sharedFileTemplate: "<AMAZONS3>:://[Your S3 bucket where Mobius
 Archives will be stored]"
  createDocumentServer: "YES"
  rds:
    provider: "POSTGRESQL"
    initOrUpgrade: "YES"
    endpoint: "[Your DB host name]"
    port: "5432"
    protocol: "TCPS"
    user: "[Your database user name]"
    schema: "[Your database name]"
    password: "[Your password for your Database]"

  mobiusDiagnostics:
    persistentVolume:
      enabled: true
      claimName:
 mobius-diagnostics
  fts:
    persistence:
      enabled: true
      claimName: "mobius-fts"
      fqdn: "localhost"
  persistentVolume:
    enabled: true
    claimName: "mobius"
  clustering:
    port: 5701
    kubernetes:
      enabled: true
      namespace: mobius
      serviceName: mobius
```

Deploying helm chart

To deploy Mobius Server using helm, execute the following command.

```
$ helm install mobius-server --namespace mobius -f values-mobius-
server.yaml mobius-<version>.tgz
NAME: mobius
LAST DEPLOYED: Wed May 18 19:16:01 2022
NAMESPACE: mobius
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace mobius -l
 "app.kubernetes.io/name=mobius,app.kubernetes.io/instance=mobius" -o
 jsonpath="    {.items[0].metadata.name}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl port-forward $POD_NAME 8080:80
$
$ kubectl get pods -n mobius
NAME                              READY  STATUS    RESTARTS   AGE
mobius-server-6df4f689dc-f4wld   1/1     Running   0          72s
```

Deploying Mobius View

Use helm to deploy Mobius View on Kubernetes. For more information see, *Using helm chart to deploy Mobius View in Mobius Server documentation*.

Creating Mobius View authorization secret

Mobius View requires a valid license key to work properly. This authorization key must exist in a Kubernetes secret name mobius-license and exist in the namespace that Mobius View is deployed in. To create the mobius-license secret, execute the following command.

```
kubectl create secret generic mobius-license \
  --from-literal=license=[your Mobius license key] -n mobius
```

Creating service client secret

Create a service client secret using the following command.

```
$ kubectl create secret generic asg-mobius-view-service-client-secret \
>   --from-literal=value=deleteme -n mobius
secret/asg-mobius-view-service-client-secret created
```

Configuring external service to OIDC provider

If you are using an OIDC provider that is not deployed in the EKS cluster then it will be necessary to define an external service that will map to that external OIDC provider so Kubernetes services will be able to access it. The following is an example of what an external service might look like for an external Keycloak deployent. For more information on Kubernetes services and Endpoints see, Service.

```
---
kind: "Service"
apiVersion: "v1"
metadata:
  name: "external-oidc-service"
spec:
  type: ClusterIP
  ports:
  - name: https
    port: 8443
    targetPort: 8443
  - name: http
    port: 8282
    targetPort: 8282

---
kind: "Endpoints"
apiVersion: "v1"
metadata:
  name: "external-oidc-service"
subsets:
  -
    addresses:
      -
        ip: "[IP Address of your External OIDC provider]"
    ports:
      - name: https
        port: 8443
      - name: http
        port: 8282
```

Configuring service account to access

Mobius View needs access to some Kubernetes resource to discover other Mobius View pods in the cluster for caching. The following ClusterRole must be provided to the default ServiceAccount in the mobius namespace.

```
mobius-cluster-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: mobius-cluster-role
rules:
- apiGroups:
  - ""
  resources:
  - services
  - endpoints
  - pods
  - node
  verbs:
  - get
  - list
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: mobius-cluster-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: mobius-cluster-role
subjects:
- kind: ServiceAccount
  name: default
  namespace: mobius
```

Perform the following command to grant Mobius View access.

```
$ kubectl apply -f mobius-cluster-role.yaml
clusterrole.rbac.authorization.k8s.io/mobius-cluster-role created
clusterrolebinding.rbac.authorization.k8s.io/mobius-cluster-binding
 created
```

Configuring Mobius View values file

To deploy Mobius View you must first configure the parameters for the deployment. The values for these parameters can be set in a `<values>.yaml` file and passed to the helm deployment tool during deployment. A sample config file is shown below. For more information on this file see, Mobius View Container Parameters.

Configure the `<values>.yaml` file as shown in the below sample for Mobius View 12.1 and later. This sample script provides OIDC authentication details with NPM.

```
#replicaCount is the number of replicas required for this service
replicaCount: 1
namespace: mobius

image:
  repository: [Your ECR repository for the Mobius server Docker image]
  tag: [Your tag for your Mobius Server Docker image in ECR]

master:
  persistence:
    enabled: true
    claimName: mobius
  mobiusViewDiagnostics:
    persistentVolume:
      enabled: true
      claimName: mobius-diagnostics
datasource:
  url: jdbc:postgresql://[Your Postgres hostname]:[your postgres port]/
[name used for the mobiusview database]
  username: [username of the mobiusview database]
  password: [password for mobiusview user]
  driverClassName: org.postgresql.Driver

initRepository:
  enabled: true
  host: "mobius-server"
  port: "8080"
  documentServer: "vdrnetds"

  java:
  opts: ""
```

```
asg:
  vendor:
    type: "ASG"
  bootstrap:
    mobiusAdministratorGroups: "mobiusadmin"
  clustering:
    port: 5701
    kubernetes:
      enabled: true
      namespace: mobius
      serviceName: mobius-view-mobiusview
  security:
    basicauth:
      username: "admin"
      password: "admin"
      groups: "mobiusadmin"
    enabled: true
    urlPatterns: "/rest/*,/adminrest/*,/directconnect/*,/cmis/*"

    openidConnectTwo:
      enabled: true
      type: "identity"
      validate: false
      iamGroups: true
      outbound: true
      client:
        jwkbase: "https://external-keycloak-service:8443/auth/realms"
      mapping:
        username: "preferred_username"
        userid:  "zenith-user-id"
        tenantname: "zenith-tenant-code"
        tenantid: "zenith-tenant-id"
        groups: "groups"
      serviceTokenSettings:
        oidcConfig:
          serviceTokenUrl: "https://[your host]/auth/realms/master/
protocol/openid-connect/token"
          tenantCode: "_"
          serviceTokens:
            - name: "mobiusview"
              clientId: "zenith-mobius-view-s2s"
              clientSecret: "<client_secret>"
              scopes: ["authorization.registration.application"]
          generateTokens:
            - urlPatterns: ["/authorization/*"]
              name: "mobiusview"
    oidc:
      npm:
        enabled: true
        oidcconfig:
          issuer: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms/<your_security_realm>"
          redirecturi: ""
          clientid: "<your_client_ID>"
          responsetype: "code"
          scope: "openid profile email"
          logouturl: "https://<LB_HOST_for_OIDC_Provider or
 LOCAL_MACHINE_HOST>/auth/realms/<your_security_realm>/protocol/openid-
connect/logout"
          keycloak: true
```

Note: Refer OIDC Authentication and Authorization for Mobius View with NPM on page 10 and OIDC Authentication and Authorization for Mobius View without NPM on page 17 to configure OIDC for different versions of Mobius View with and without NPM, respectively.

Deploying helm chart

You can deploy Mobius View using helm using the following command.

```
$ helm install mobius-view --namespace mobius -f values-mobius-view.yaml
 mobiusview-<version>.tgz
coalesce.go:196: warning: cannot overwrite table with non table for java
 (map[opts:])
NAME: mobius-view
LAST DEPLOYED: Thu May 19 21:18:22 2022
NAMESPACE: mobius
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export POD_NAME=$(kubectl get pods --namespace mobius -l
 "app.kubernetes.io/name=mobiusview,app.kubernetes.io/instance=mobius-
view" -o jsonpath="{.items[0].metadata.name}")
  echo "Visit http://127.0.0.1:8080 to use your application"
  kubectl port-forward $POD_NAME 8080:80
$
$ kubectl get pods -n mobius
NAME                                    READY  STATUS    RESTARTS  AGE
mobius-server-5568d8b57b-wbv4v          1/1    Running   0         6d2h
mobius-view-mobiusview-9cd7f654c-hmbjg  1/1    Running   0         6d2h
```

Configuring Ingress Controller

To access the Mobius View deployment outside the EKS cluster, ingress rules must be defined to route traffic from the AWS load balancer for the NGINX Ingress controller to the Mobius View services and your external OIDC provider if used.

Configuring external OIDC Provider Ingress

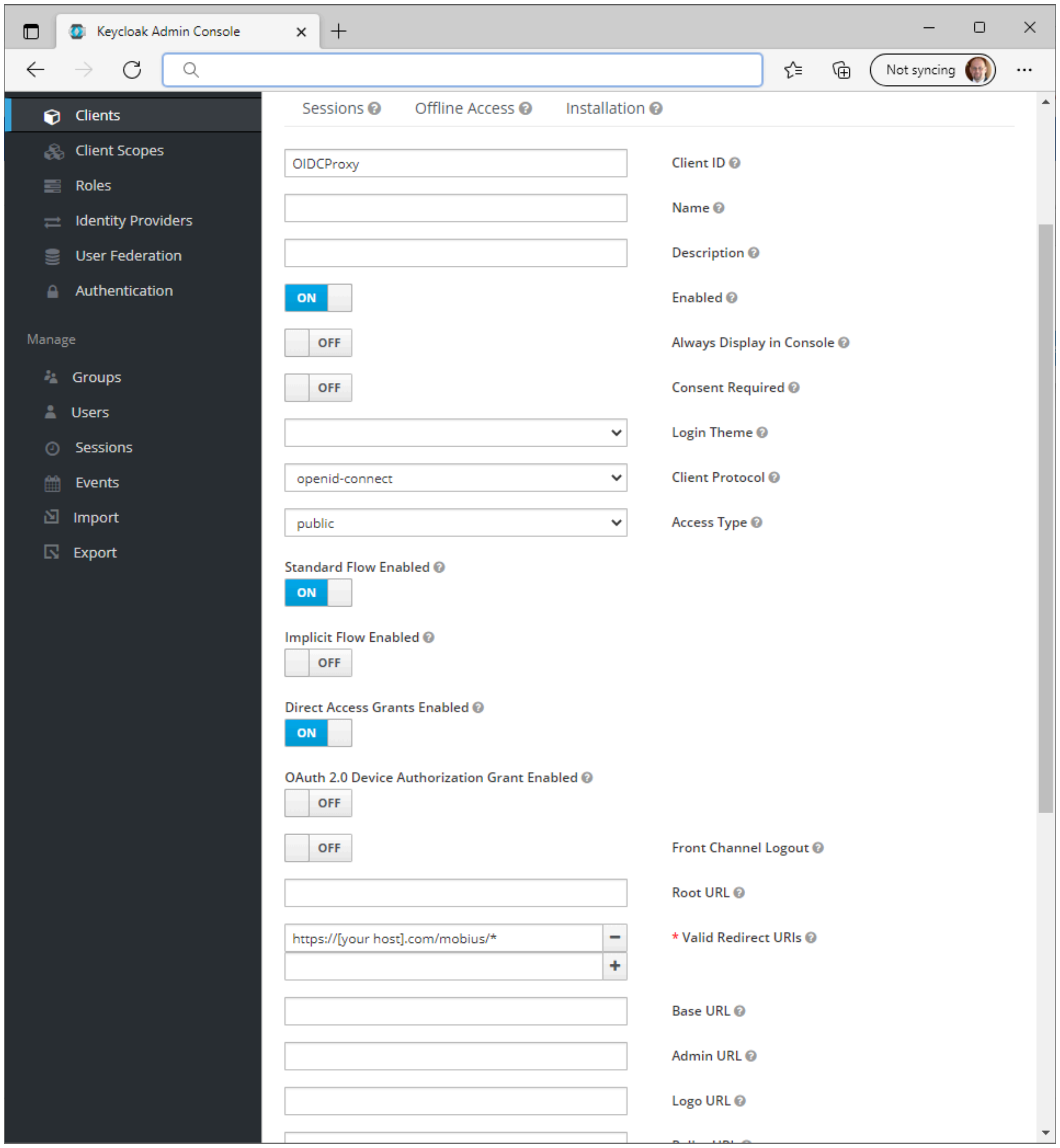
The following is an example of an ingress rule for a keycloak OIDC provider and ingress for Mobius View.

Note: This configuration is optional.

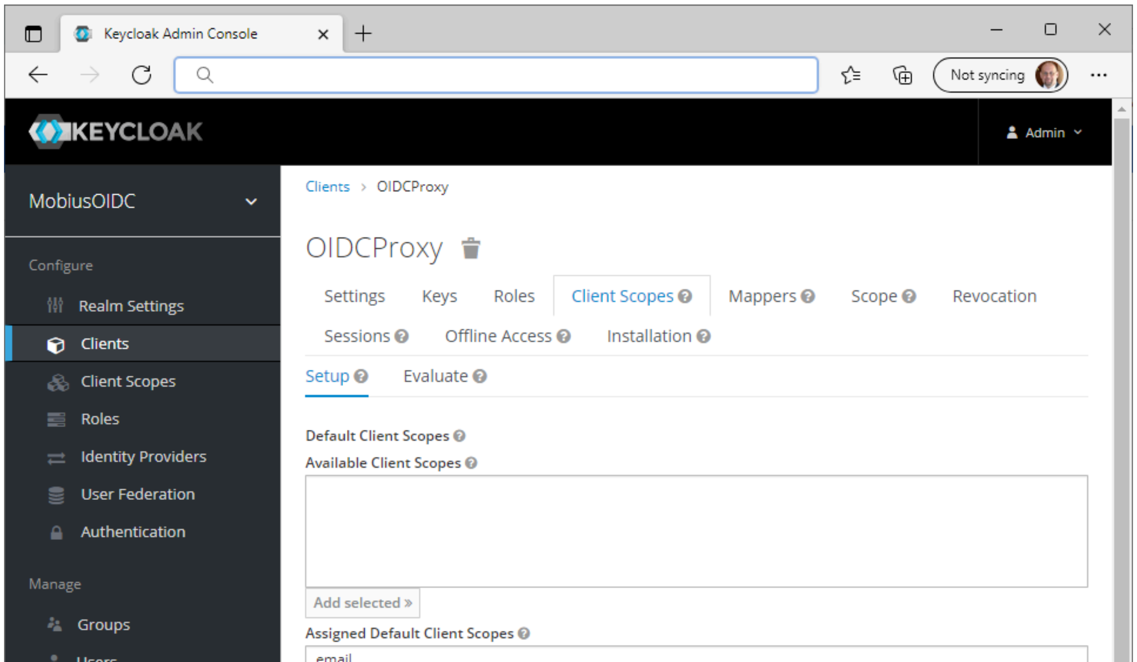
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.org/ssl-services: external-keycloak-service
    nginx.org/location-snippets: |
      proxy_set_header Content-Security-Policy "connect-src
 https://mobius.asgcontent.com/     blob:; default-src blob: data:
 'self'; img-src blob: data: 'self'; script-src 'self' 'sha256-
wSk7Pac68P5NGz0ckYIUSA8nd7eh8zkveKcseL24KB0='; style-src 'self' 'unsafe-
inline';";
      proxy_set_header X-Content-Type-Options 'nosniff';
      proxy_set_header X-XSS-Protection '1; mode=block';
      proxy_set_header X-Frame-Options 'sameorigin';
      proxy_set_header Strict-Transport-Security 'max-age=315360000;
 includeSubDomains;     preload';
      proxy_set_header Access-Control-Allow-Origin
 'mobius.asgcontent.com';
    nginx.org/proxy-connect-timeout: "60s"
    nginx.org/proxy-read-timeout: "60s"
    nginx.org/proxy-send-timeout: "60s"
    nginx.org/proxy-buffer-size: "80k"
    nginx.org/proxy-buffers: "8 80k"
  name: mobius-ingress
  namespace: mobius
spec:
  ingressClassName: nginx
  rules:
    - host: mobius.asgcontent.com
        http:
          paths:
            - backend:
                service:
                  name: mobius-view-mobiusview
                  port:
                    number: 80
              path: /mobius/rest/contentstreams
              pathType: Prefix
            - backend:
                service:
                  name: mobius-view-mobiusview
                  port:
                    number: 80
              path: /mobius/
              pathType: Prefix
            - backend:
                service:
                  name: mobius-view-mobiusview
                  port:
                    number: 80
              path: /mobius/view/
              pathType: Prefix
            - backend:
                service:
                  name: external-keycloak-service
                  port:
                    number: 8443
              path: /auth
              pathType: Prefix
tls:
- hosts:
  - mobius.asgcontent.com
  secretName: mobius-secret
```

Configuring OIDC Provider

Configure your OIDC provider client with the correct callback URLs and users. A sample configuration for Keycloak is given below. See your OIDC provider documentation for more information.



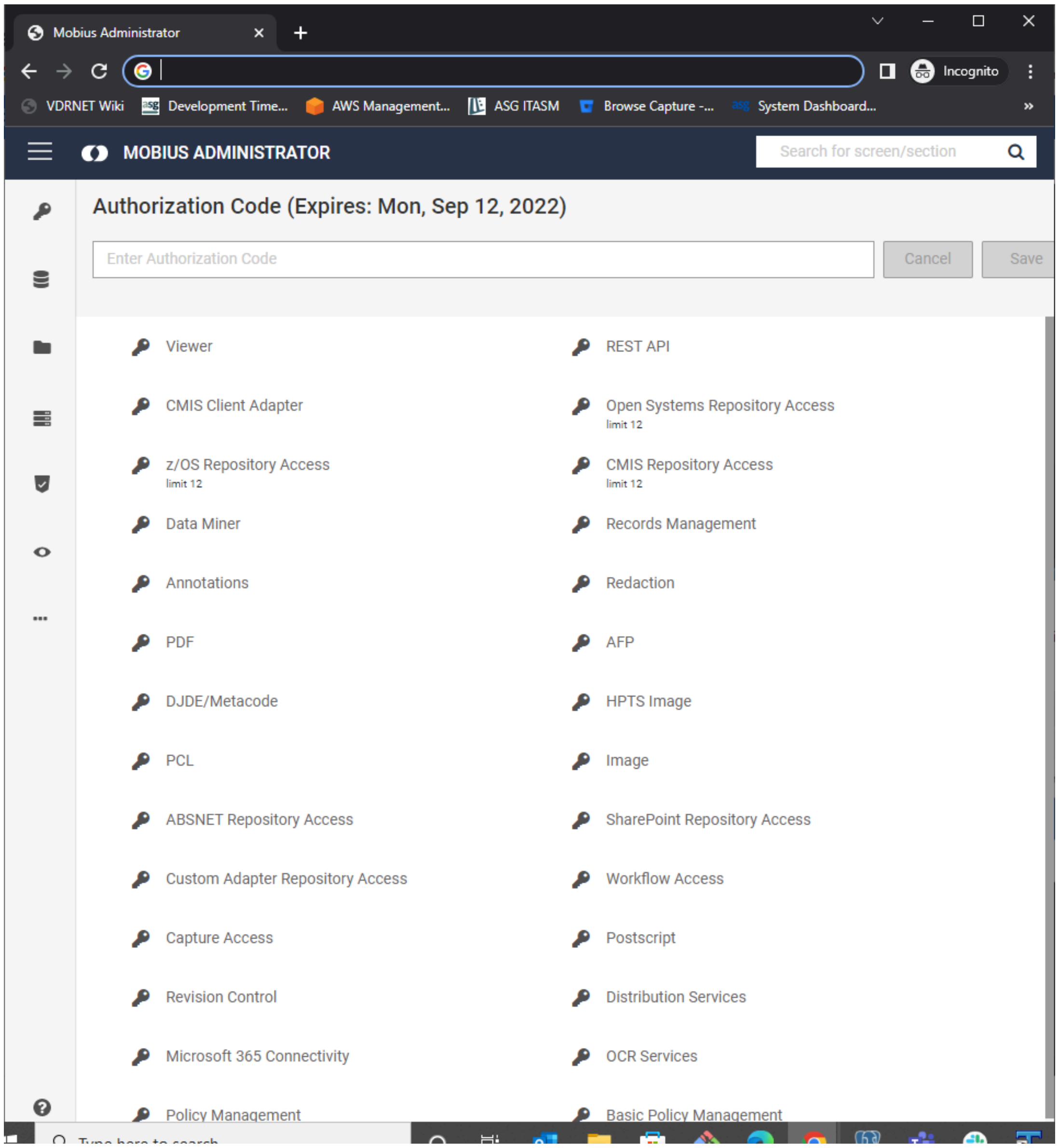
Client ID: OIDCProxy
Name:
Description:
Enabled: ON
Always Display in Console: OFF
Consent Required: OFF
Login Theme:
Client Protocol: openid-connect
Access Type: public
Standard Flow Enabled: ON
Implicit Flow Enabled: OFF
Direct Access Grants Enabled: ON
OAuth 2.0 Device Authorization Grant Enabled: OFF
Front Channel Logout: OFF
Root URL:
* Valid Redirect URIs: https://[your host].com/mobius/*
Base URL:
Admin URL:
Logo URL:



Default Client Scopes
Available Client Scopes
Assigned Default Client Scopes
email

Testing the Deployment

1. Open a browser and access the Mobius View application using the Mobius View URL. For example: `https://[your host]/mobius/admin`.
2. On the OIDC provider's login page, enter the credentials that you have configured for a mobiusadmin user role.



Authorization Code (Expires: Mon, Sep 12, 2022)

Enter Authorization Code

Viewer	REST API
CMIS Client Adapter	Open Systems Repository Access limit 12
z/OS Repository Access limit 12	CMIS Repository Access limit 12
Data Miner	Records Management
Annotations	Redaction
PDF	AFP
DJDE/Metacode	HPTS Image
PCL	Image
ABSNET Repository Access	SharePoint Repository Access
Custom Adapter Repository Access	Workflow Access
Capture Access	Postscript
Revision Control	Distribution Services
Microsoft 365 Connectivity	OCR Services
Policy Management	Basic Policy Management

You will be able to view and access the Mobius View application.