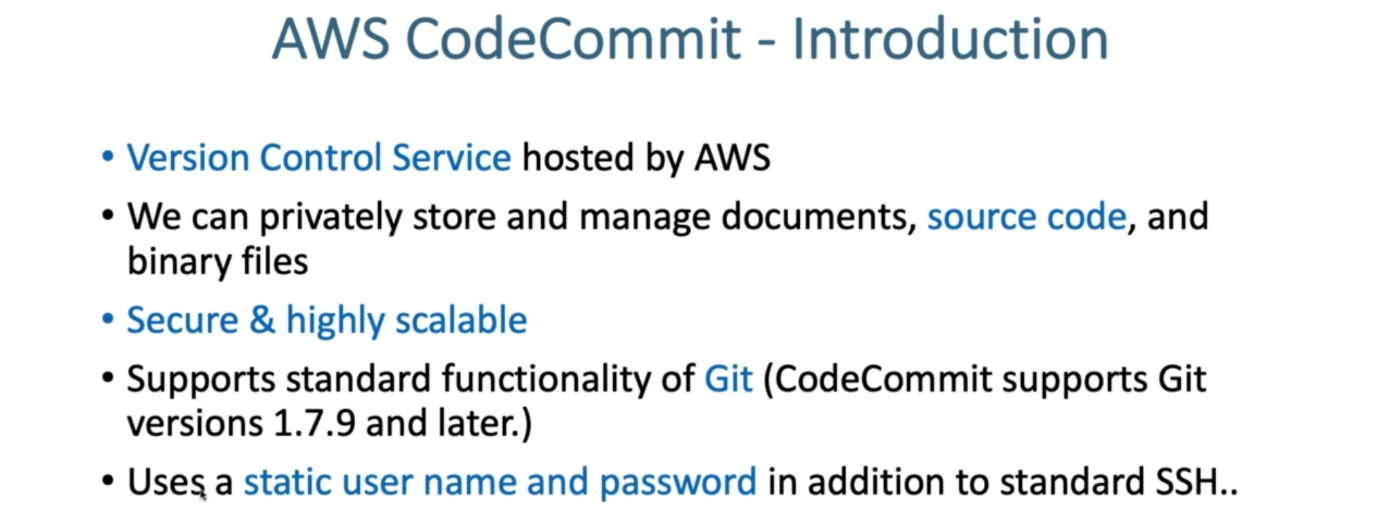
**04. CodeCommit - Create Git Repository, Clone, Copy Manifests and Push**

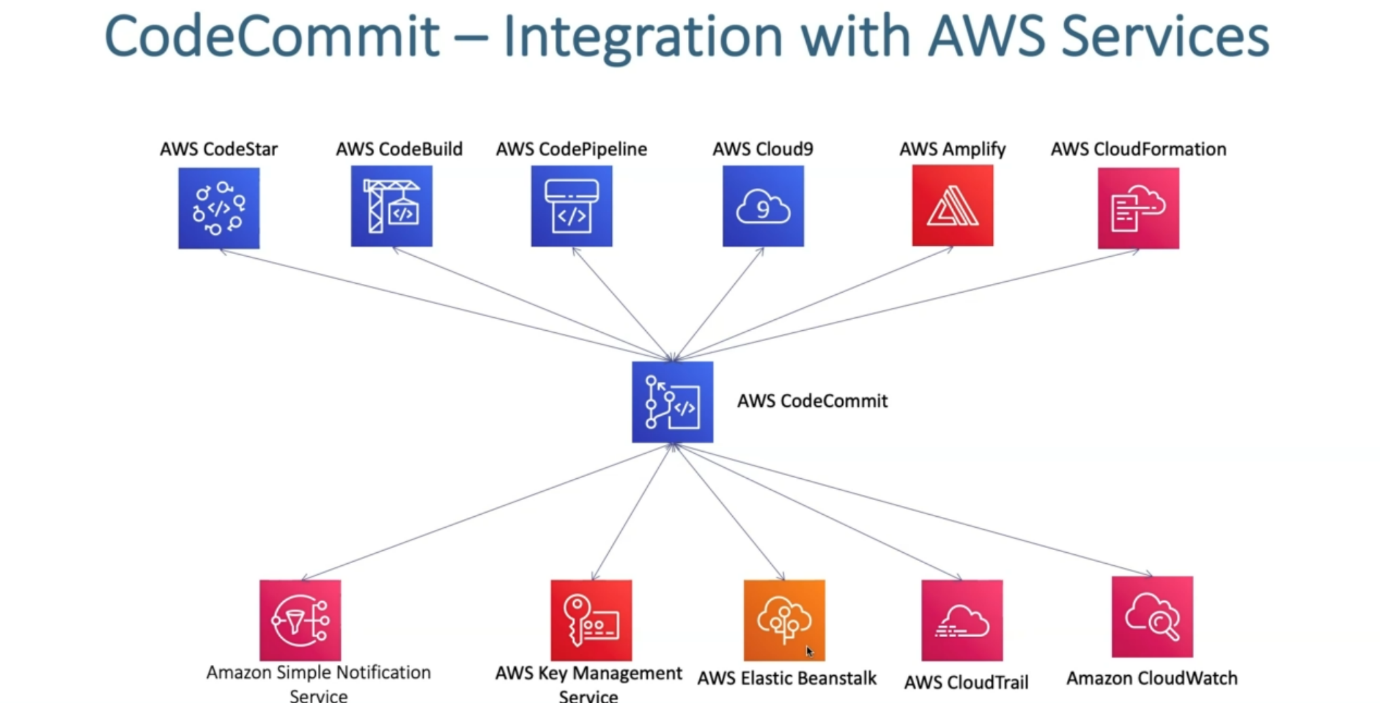


--- we are going to understand on a high level about what is codecommit service in aws and then we are going to create a code commit repository for our EKS devops use case and then we push the content to our remote GitHub repository.

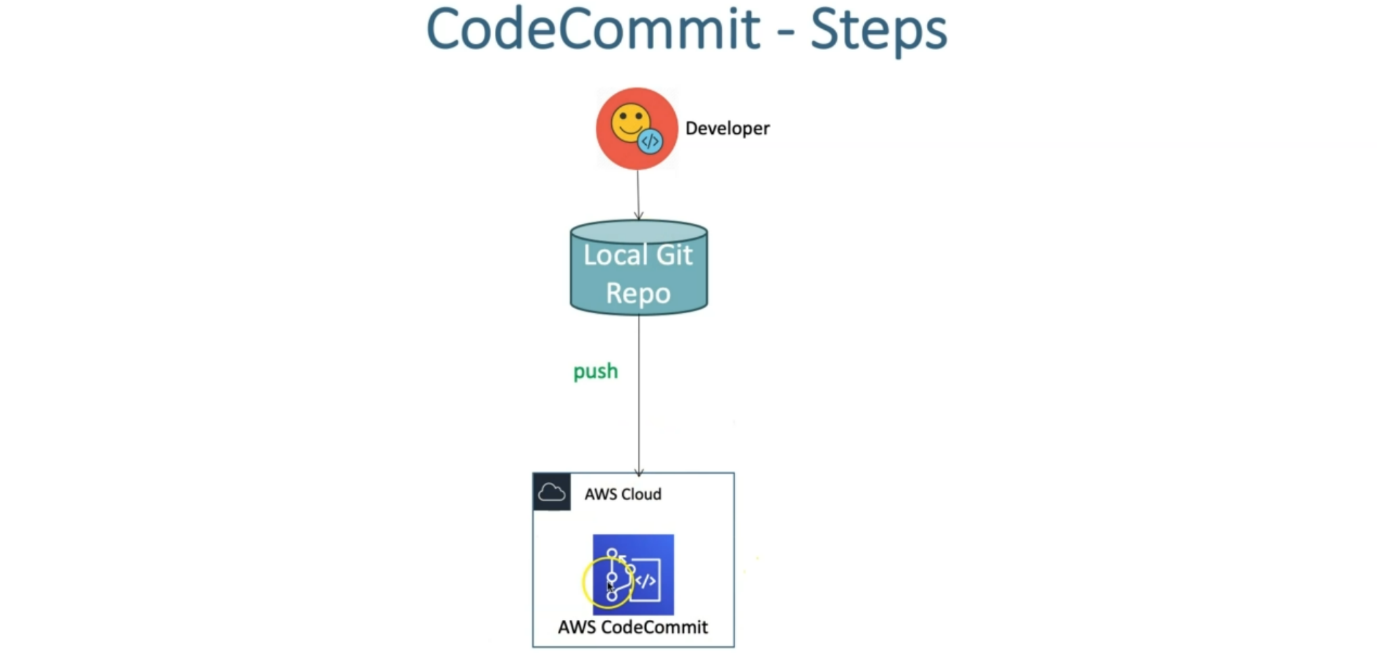
--- let's go back and understand what is this codecommit repository. it is nothing but a version-controlled service hosted by aws.

--- we can privately store and manage documents, source code and then binary files. It is secure and then highly scalable.

**Codecommit integration aws service**



**Codecommit steps**



--- **what are the steps, we are going to perform…?**

--- So, we are going to create a code commit repository and then clone that to our local and

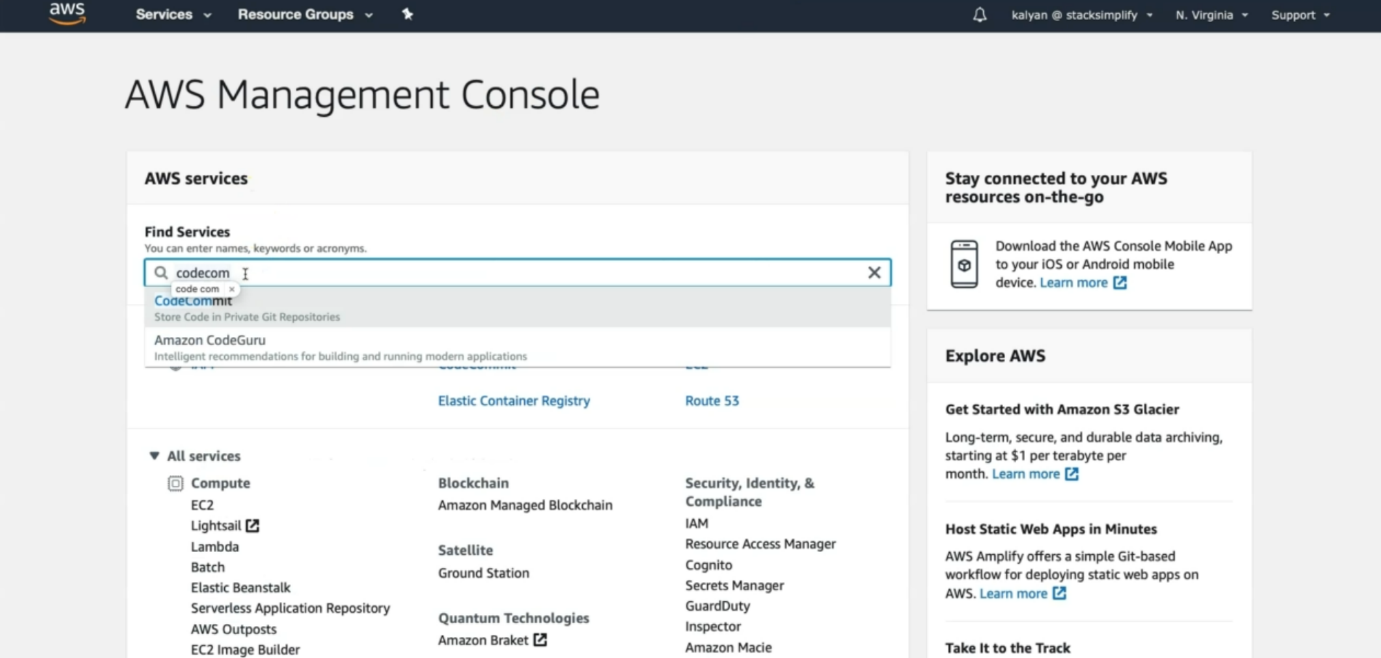
then push some code, which is nothing but our kube manifests or whatever is required for this devops demo. all those things, we are going to push that to local repo and then push the same to our code commit repository.

--- let's go ahead and do that now.

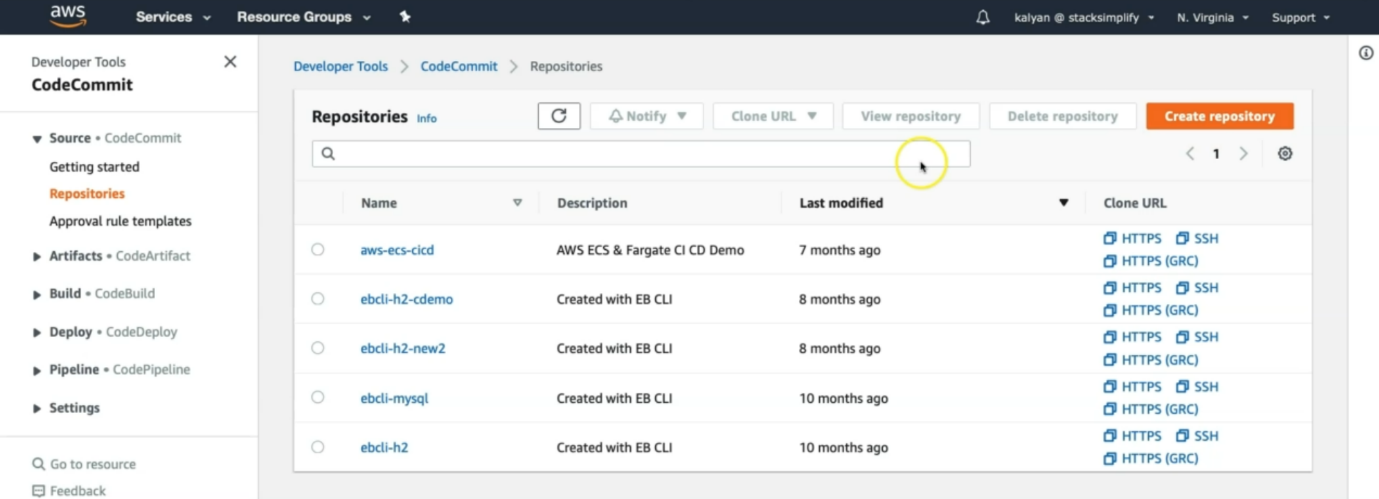
**Create CodeCommit Repository**

--- Code Commit Introduction

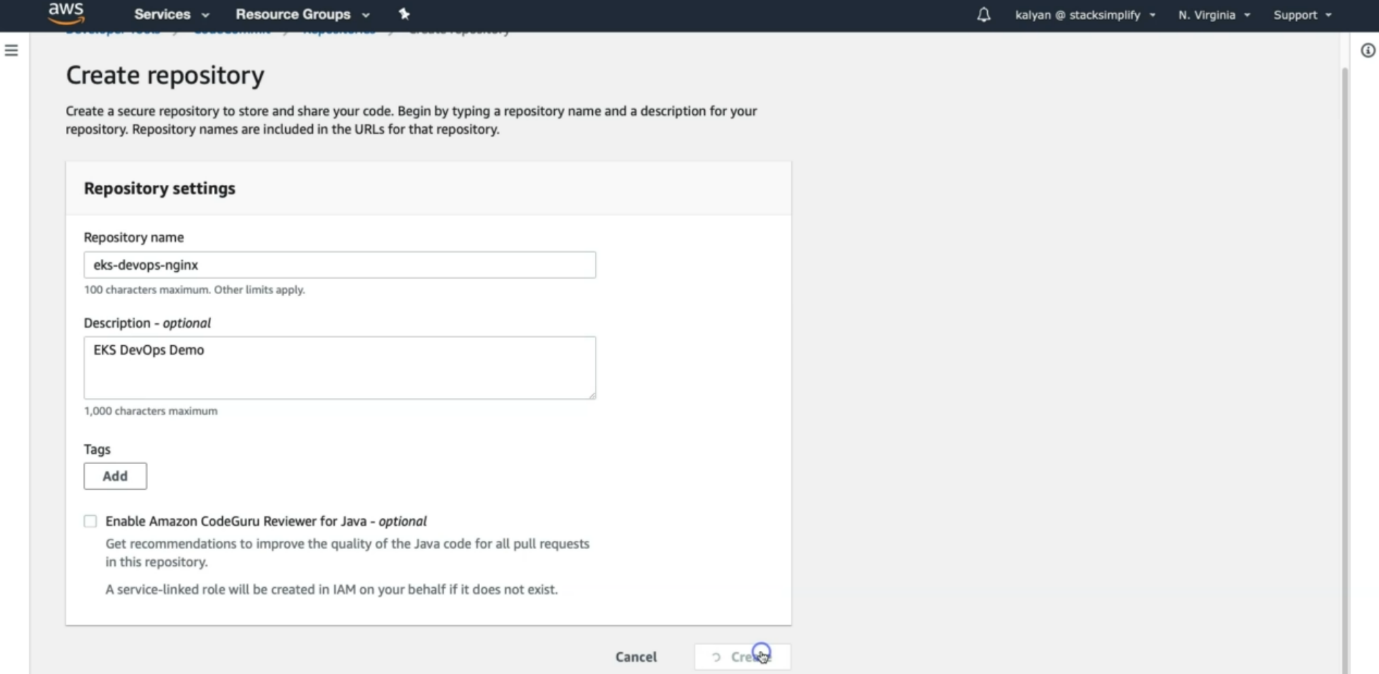
--- Create Code Commit Repository with name as eks-devops-nginx



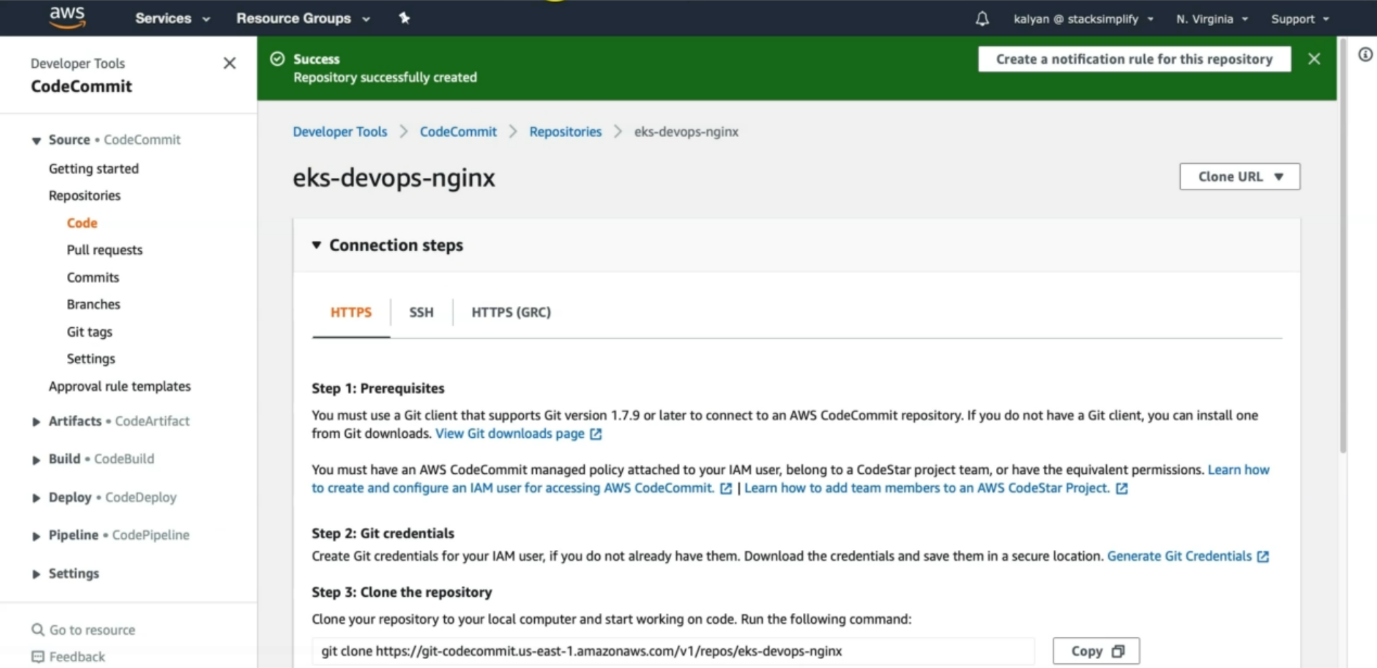
--- search for codecommit.



--- click on create repository.

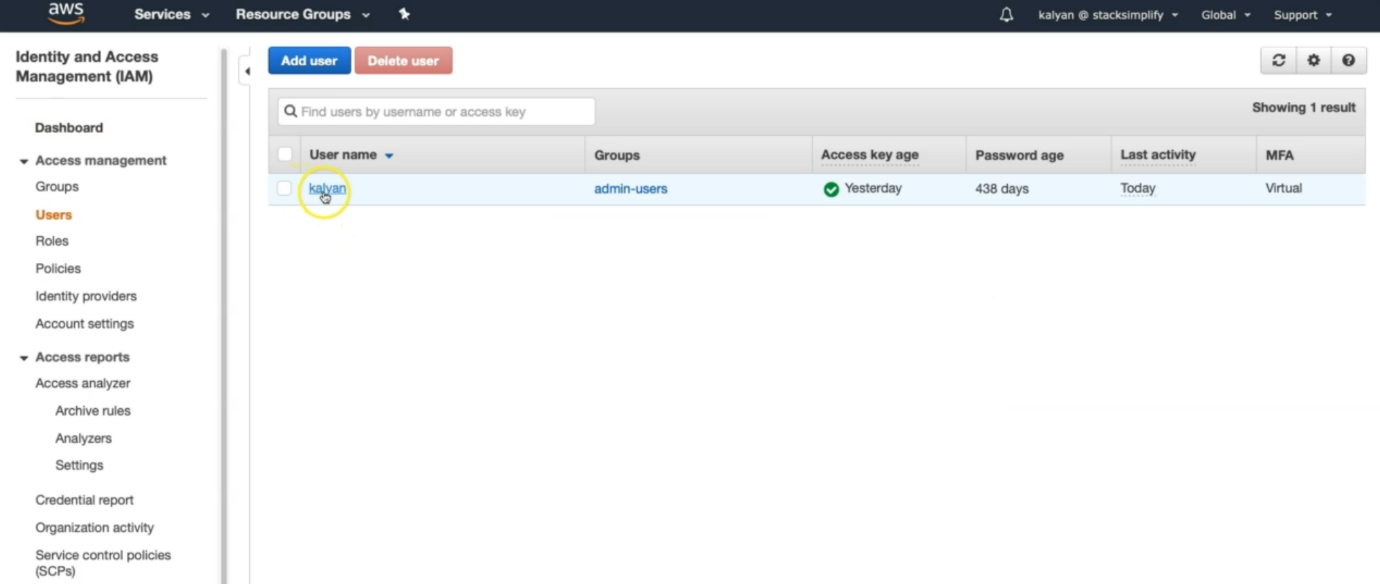


--- click on create.

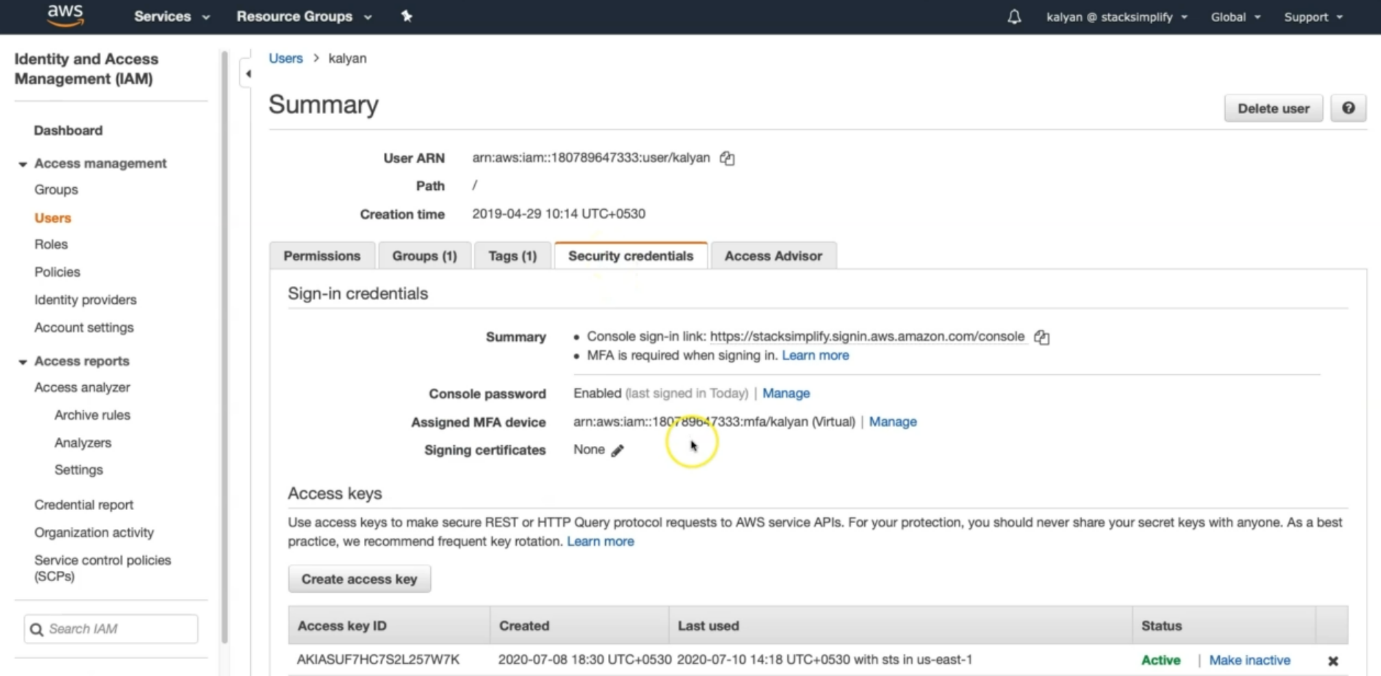


--- my eks-devops-nginx repository got created. Clone the url from here to local.

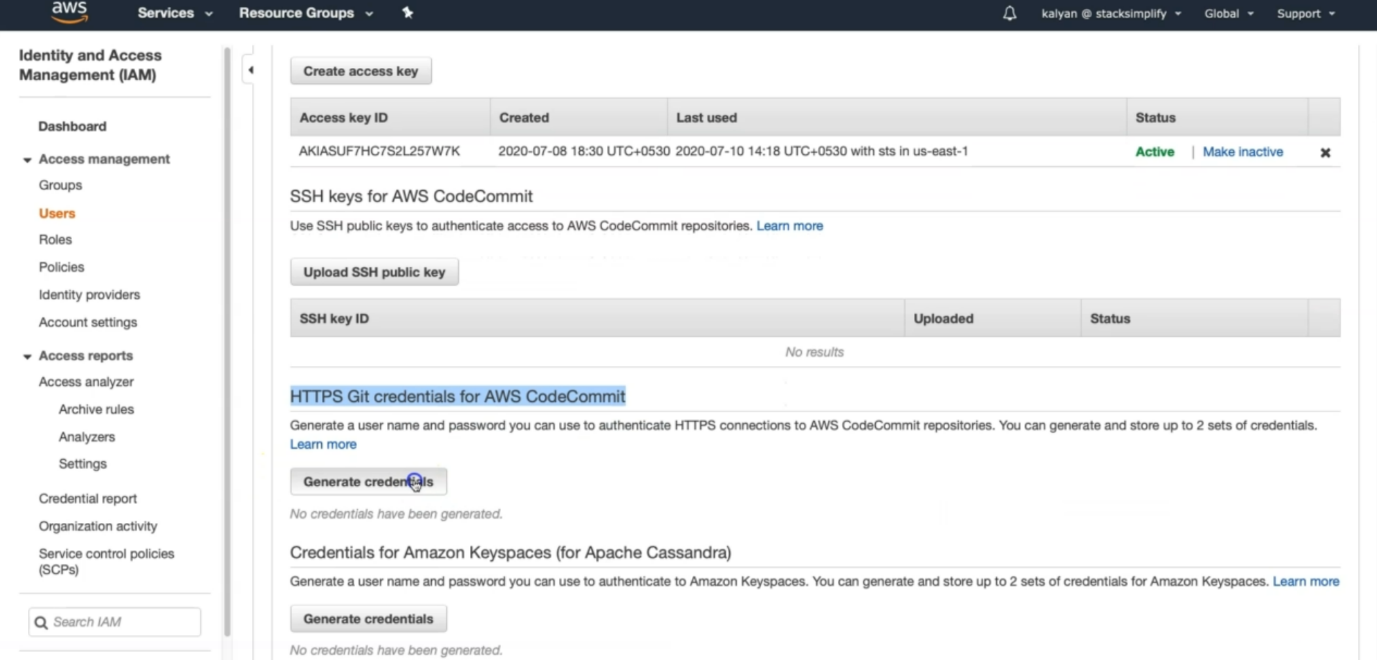
**Create git credentials from IAM Service and make a note of those credentials.**



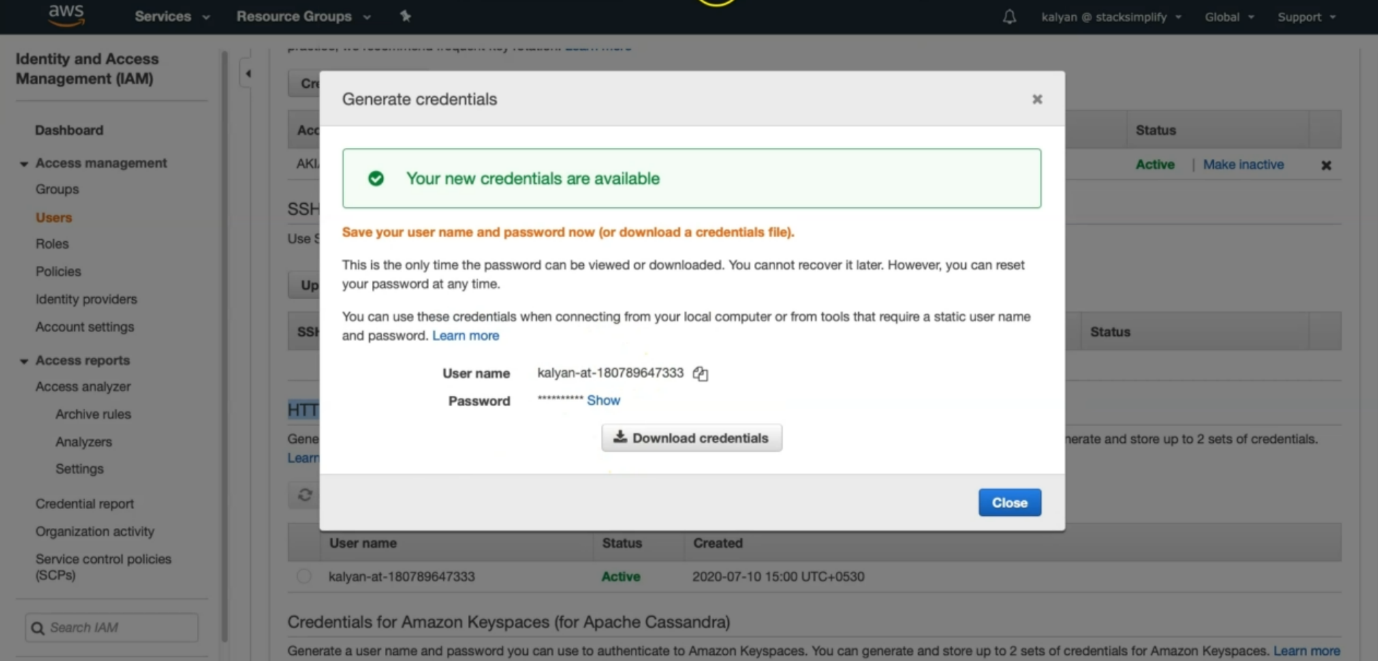
--- click on that user.



--- click on the security credentials and drag below.



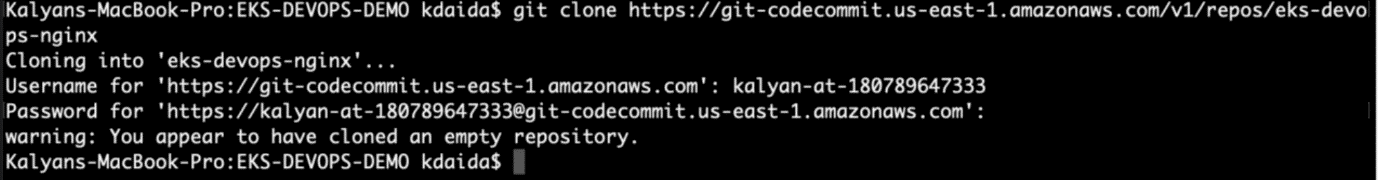
--- click on generate credentials.



--- Clone the git repository from Code Commit to local repository, during the process provide your git credentials generated to login to git repo

**Git clone to local**

--- **git clone https://git-codecommit.us-east-1.amazonaws.com/v1/repos/eks-devops-nginx**



--- Copy all files from course section 11-DevOps-with-AWS-Developer-Tools/Application-Manifests to local repository

1. buildspec.yml
2. Dockerfile
3. app1

index.html # this index.html file will be copied to nginx location.

1. kube-manifests
2. 01-DEVOPS-Nginx-Deployment.yml
3. 02-DEVOPS-Nginx-NodePortService.yml
4. 03-DEVOPS-Nginx-ALB-IngressService.yml

--- **buildspec.yml**

version: 0.2

phases:

  install:

    commands:

      - echo "Install Phase - Nothing to do using latest Amazon Linux Docker Image for CodeBuild which has all AWS Tools - https://github.com/aws/aws-codebuild-docker-images/blob/master/al2/x86\_64/standard/3.0/Dockerfile"

  pre\_build:

      commands:

        # Docker Image Tag with Date Time & Code Buiild Resolved Source Version

        - TAG="$(date +%Y-%m-%d.%H.%M.%S).$(echo $CODEBUILD\_RESOLVED\_SOURCE\_VERSION | head -c 8)"

        # Update Image tag in our Kubernetes Deployment Manifest

        - echo "Update Image tag in kube-manifest..."

        - sed -i 's@CONTAINER\_IMAGE@'"$REPOSITORY\_URI:$TAG"'@' kube-manifests/01-DEVOPS-Nginx-Deployment.yml

        # Verify AWS CLI Version

        - echo "Verify AWS CLI Version..."

        - aws --version

        # Login to ECR Registry for docker to push the image to ECR Repository

        - echo "Login in to Amazon ECR..."

        - $(aws ecr get-login --no-include-email)

        # Update Kube config Home Directory

        - export KUBECONFIG=$HOME/.kube/config

  build:

    commands:

      # Build Docker Image

      - echo "Build started on `date`"

      - echo "Building the Docker image..."

      - docker build --tag $REPOSITORY\_URI:$TAG .

  post\_build:

    commands:

      # Push Docker Image to ECR Repository

      - echo "Build completed on `date`"

      - echo "Pushing the Docker image to ECR Repository"

      - docker push $REPOSITORY\_URI:$TAG

      - echo "Docker Image Push to ECR Completed -  $REPOSITORY\_URI:$TAG"

      # Extracting AWS Credential Information using STS Assume Role for kubectl

      - echo "Setting Environment Variables related to AWS CLI for Kube Config Setup"

      - CREDENTIALS=$(aws sts assume-role --role-arn $EKS\_KUBECTL\_ROLE\_ARN --role-session-name codebuild-kubectl --duration-seconds 900)

      - export AWS\_ACCESS\_KEY\_ID="$(echo ${CREDENTIALS} | jq -r '.Credentials.AccessKeyId')"

      - export AWS\_SECRET\_ACCESS\_KEY="$(echo ${CREDENTIALS} | jq -r '.Credentials.SecretAccessKey')"

      - export AWS\_SESSION\_TOKEN="$(echo ${CREDENTIALS} | jq -r '.Credentials.SessionToken')"

      - export AWS\_EXPIRATION=$(echo ${CREDENTIALS} | jq -r '.Credentials.Expiration')

      # Setup kubectl with our EKS Cluster

      - echo "Update Kube Config"

      - aws eks update-kubeconfig --name $EKS\_CLUSTER\_NAME

      # Apply changes to our Application using kubectl

      - echo "Apply changes to kube manifests"

      - kubectl apply -f kube-manifests/

      - echo "Completed applying changes to Kubernetes Objects"

      # Create Artifacts which we can use if we want to continue our pipeline for other stages

      - printf '[{"name":"01-DEVOPS-Nginx-Deployment.yml","imageUri":"%s"}]' $REPOSITORY\_URI:$TAG > build.json

      # Additional Commands to view your credentials

      #- echo "Credentials Value is..  ${CREDENTIALS}"

      #- echo "AWS\_ACCESS\_KEY\_ID...  ${AWS\_ACCESS\_KEY\_ID}"

      #- echo "AWS\_SECRET\_ACCESS\_KEY...  ${AWS\_SECRET\_ACCESS\_KEY}"

      #- echo "AWS\_SESSION\_TOKEN...  ${AWS\_SESSION\_TOKEN}"

      #- echo "AWS\_EXPIRATION...  $AWS\_EXPIRATION"

      #- echo "EKS\_CLUSTER\_NAME...  $EKS\_CLUSTER\_NAME"

artifacts:

  files:

    - build.json

    - kube-manifests/\*

--- **app1/index.html**

<!DOCTYPE html>

<html>

   <body style="background-color:rgb(228, 250, 210);">

      <h1>Welcome to Stack Simplify - App Version - V1 </h1>

      <h3> DevOps for EKS with AWS Developer Tools</h3>

      <p>Application Name: App1</p>

   </body>

</html>

--- **Dockerfile**

FROM nginx

COPY app1 /usr/share/nginx/html/app1

--- **kube-manifests/01-DEVOPS-Nginx-Deployment.yml**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: eks-devops-deployment

  labels:

    app: eks-devops

spec:

  replicas: 2

  selector:

    matchLabels:

      app: eks-devops

  template:

    metadata:

      labels:

        app: eks-devops

    spec:

      containers:

        - name: eks-devops

          image: CONTAINER\_IMAGE

          ports:

            - containerPort: 80

--- **kube-manifests/02-DEVOPS-Nginx-NodePortService.yml**

apiVersion: v1

kind: Service

metadata:

  name: eks-devops-nodeport-service

  labels:

    app: eks-devops

  annotations:

#Important Note:  Need to add health check path annotations in service level if we are planning to use multiple targets in a load balancer

    alb.ingress.kubernetes.io/healthcheck-path: /app1/index.html

spec:

  type: NodePort

  selector:

    app: eks-devops

  ports:

    - port: 80

      targetPort: 80

--- **kube-manifests/03-DEVOPS-Nginx-ALB-IngressService.yml**

# Annotations Reference:  https://kubernetes-sigs.github.io/aws-alb-ingress-controller/guide/ingress/annotation/

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

  name: eks-devops-ingress-service

  labels:

    app: eks-devops

  annotations:

    # Ingress Core Settings

    kubernetes.io/ingress.class: "alb"

    alb.ingress.kubernetes.io/scheme: internet-facing

    # Health Check Settings

    alb.ingress.kubernetes.io/healthcheck-protocol: HTTP

    alb.ingress.kubernetes.io/healthcheck-port: traffic-port

    alb.ingress.kubernetes.io/healthcheck-interval-seconds: '15'

    alb.ingress.kubernetes.io/healthcheck-timeout-seconds: '5'

    alb.ingress.kubernetes.io/success-codes: '200'

    alb.ingress.kubernetes.io/healthy-threshold-count: '2'

    alb.ingress.kubernetes.io/unhealthy-threshold-count: '2'

    ## SSL Settings

    alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS":443}, {"HTTP":80}]'

    alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:us-east-1:180789647333:certificate/9f042b5d-86fd-4fad-96d0-c81c5abc71e1

    #alb.ingress.kubernetes.io/ssl-policy: ELBSecurityPolicy-TLS-1-1-2017-01 #Optional (Picks default if not used)

    # SSL Redirect Setting

    alb.ingress.kubernetes.io/actions.ssl-redirect: '{"Type": "redirect", "RedirectConfig": { "Protocol": "HTTPS", "Port": "443", "StatusCode": "HTTP\_301"}}'

    # External DNS - For creating a Record Set in Route53

    external-dns.alpha.kubernetes.io/hostname: devops.kubeoncloud.com

spec:

  rules:

    - http:

        paths:

          - path: /\* # SSL Redirect Setting

            backend:

              serviceName: ssl-redirect

              servicePort: use-annotation

          - path: /\*

            backend:

              serviceName: eks-devops-nodeport-service

              servicePort: 80

# Important Note-1: In path based routing order is very important, if we are going to use  "/\*", try to use it at the end of all rules.

--- Commit code and Push to CodeCommit Repo

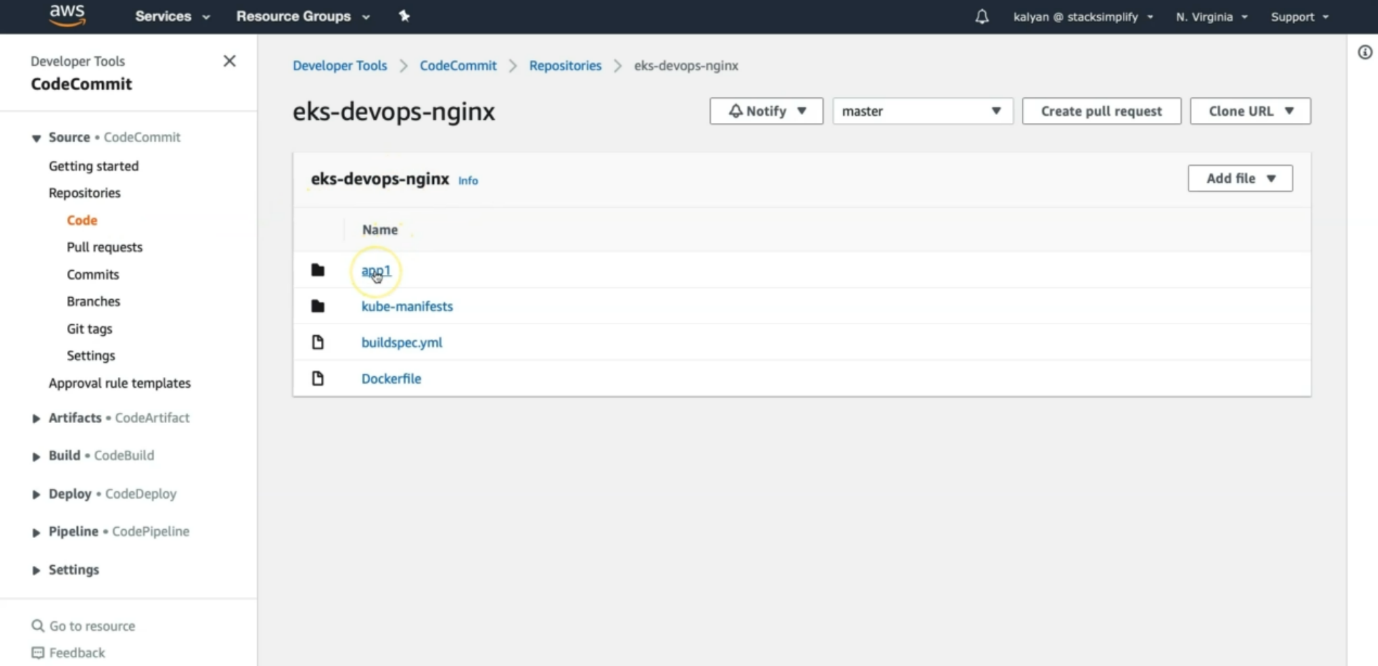
git status

git add .

git commit -am "1 Added all files"

git push

git status



--- **note** – we successfully pushed the code to the aws codecommit repository.

--- here, you can also see the options like pull request, commits, branches, git tags, settings…etc.