How to Create EKS Cluster Using eksctl?

- You can find the source code for this video in my GitHub Repo.

Create IAM User for eksctl

First of all, we need to create a user to run eksctl. If you are just getting started, you may create or use an existing user with admin privileges. A better approach would be to create a user with only the permissions needed to perform its functions. For example, eksctl user should only have permissions to create and manage EKS clusters. Let me show you how to create one.

- Go to AWS console and select `IAM` service.

In this example, we're going to create a user with minimum IAM policies needed to run the main use cases of eksctl. These are the ones used to run the integration tests. We will use both managed IAM policies, and we will have to create our own policies as well.

- Let's create IAM policies first. Give it a name `EksAllAccess` and don't forget to replace `account_id`.

EksAllAccess.json

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:*",
            "Resource": "*"
        },
        {
            "Action": [
                "ssm:GetParameter",
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws:ssm:*:<account_id>:parameter/aws/*",
                "arn:aws:ssm:*::parameter/aws/*"
            ],
            "Effect": "Allow"
        },
        {
             "Action": [
               "kms:CreateGrant",
               "kms:DescribeKey"
             ],
             "Resource": "*",
             "Effect": "Allow"
        },
        {
             "Action": [
               "logs:PutRetentionPolicy"
             ],
             "Resource": "*",
             "Effect": "Allow"
        }
    ]
}
```

- The second one is `IamLimitedAccess` policy.

IamLimitedAccess.json

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "iam:CreateInstanceProfile",
                "iam:DeleteInstanceProfile",
                "iam:GetInstanceProfile",
                "iam:RemoveRoleFromInstanceProfile",
                "iam:GetRole",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy",
                "iam:ListInstanceProfiles",
                "iam:AddRoleToInstanceProfile",
                "iam:ListInstanceProfilesForRole",
                "iam:PassRole",
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:GetRolePolicy",
                "iam:GetOpenIDConnectProvider",
                "iam:CreateOpenIDConnectProvider",
                "iam:DeleteOpenIDConnectProvider",
                "iam:TagOpenIDConnectProvider",
                "iam:ListAttachedRolePolicies",
                "iam:TagRole"
            ],
            "Resource": [
                "arn:aws:iam::<account_id>:instance-profile/eksctl-*",
                "arn:aws:iam::<account_id>:role/eksctl-*",
                "arn:aws:iam::<account_id>:oidc-provider/*",
                "arn:aws:iam::<account_id>:role/aws-service-role/eks-
nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup",
                "arn:aws:iam::<account_id>:role/eksctl-managed-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole"
            ],
            "Resource": [
                "arn:aws:iam::<account_id>:role/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": [
                        "eks.amazonaws.com",
                        "eks-nodegroup.amazonaws.com",
                        "eks-fargate.amazonaws.com"
```

```
                    ]
                }
            }
        }
    ]
}
```

We can attach IAM policies directly to the IAM user or follow the best practices and create an IAM group first.

- Create `EKS` IAM group and attach the following policies:
  - `EksAllAccess` (Customer Managed Policy)
  - `IamLimitedAccess` (Customer Managed Policy)
  - `AmazonEC2FullAccess` (AWS Managed Policy)
  - `AWSCloudFormationFullAccess` (AWS Managed Policy)
- After that, we can create an IAM user. Give it a name `eksctl` and place it in the `EKS` group. Don't forget to download credentials; we will use them to create an AWS profile.

Before proceeding to the next step, make sure that you have AWS CLI installed on your working machine. Depending on the operating system, you can follow one of those instructions to install the tool.

To run `eksctl` locally, you can export environment variables with your access key and secret or create a profile which is more convenient, in my opinion. Example with environment variables:

```
export AWS_ACCESS_KEY_ID=<access-key>
export AWS_SECRET_ACCESS_KEY=<secret-key>
```

If you just run `aws configure`, it will create a default profile. It's fine, but I don't really want to use that `eksctl` user for anything else rather than creating and managing EKS clusters. You can add a `--profile` option; this will create a named profile. Also, select the region that you want to use, in my case `us-east-1`.

```
aws configure --profile eksctl
```

To verify that everything is configured correctly, run the following command:

```
aws sts get-caller-identity --profile eksctl
```

You should get a json object with your user (example).

```
{
    "UserId": "AIDAWFUREJA54O5C55JKB",
    "Account": "<account_id>",
    "Arn": "arn:aws:iam::<account_id>:user/eksctl"
}
```

Every time you want to execute any `aws` or `eksctl` command, you need to add `--profile eksctl`.

Create Simple Public EKS Cluster Using eksctl

In this section, we will create a simple public EKS cluster using eksctl. If you don't have eksctl installed, follow one of those instructions. By public, I mean a Kubernetes cluster with nodes that have **public** IP addresses. You don't need a NAT gateway for that setup.

The simplest way is to run `create` command. It will create VPC as well as an EKS cluster for you. It may be the easiest way but rarely what you want.

```
eksctl create cluster --profile eksctl
```

`eksctl` will create `CloudFormation` stacks to create EKS cluster and instance groups. In case of an error, you can always inspect the stack itself.

You can customize your cluster by providing the flags to the eksctl tool. However, the best approach would be to create a config file. In that case, it's much easier to reproduce the same infrastructure and track the changes in the git.

When the creation is completed, eksctl will automatically configure the kubectl context. You can immediately access the Kubernetes cluster. Run `kubectl get svc` to get Kubernetes api service from default namespace. You can also run `kubectl get nodes` to get all available Kubernetes workers.

To delete the cluster, run the delete command and provide the name of the cluster.

```
eksctl delete cluster --name <name> --profile eksctl
```

Create Private EKS Cluster Using eksctl

For this example, we will create an EKS cluster with only private nodes. You still will be able to expose services to the internet using a load balancer and ingress. It's just Kubernetes nodes will

not be exposed to the internet, which is rarely allowed by the companies.

- Let's create an eksctl config to define the cluster properties. You can find all the possible options here.

1-example.yaml

```yaml
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: private-eks
  region: us-east-1
  version: "1.21"
availabilityZones:
- us-east-1a
- us-east-1b
managedNodeGroups:
- name: general
  privateNetworking: true
  instanceType: t3.small
  desiredCapacity: 1
- name: spot
  privateNetworking: true
  instanceType: t3.small
  spot: true
  desiredCapacity: 1
  labels:
    role: spot
  taints:
  - key: spot
    value: "true"
    effect: NoSchedule
```

- To create this cluster, provide config to the `eksctl`.

```
eksctl create cluster -f 1-example.yaml --profile eksctl
```

- Similar command to delete the cluster.

```
eksctl delete cluster -f 1-example.yaml --profile eksctl
```

Create EKS Cluster in Existing VPC Using eksctl

Often you already have a VPC, and you want to create an EKS cluster in the same network. You can do it with eksctl; you just need to provide a few additional variables.

- Create `main` VPC with the following IPv4 CIDR block `10.0.0.0/16`
- Create `igw` Internet Gateway and attach it to the `main` VPC.
- Create 4 subnets in 2 different availability zones.
  - `private-us-east-1a`, CIDR: `10.0.0.0/18`
  - `private-us-east-1b`, CIDR: `10.0.64.0/18`
  - `public-us-east-1a`, CIDR: `10.0.128.0/18`
  - `public-us-east-1b`, CIDR: `10.0.192.0/18`
- Allocate public IP address for NAT Gateway. Give it a name `nat`.
- Create NAT gateway with `nat` name and place it in one of the public subnets.
- Create 2 routing tables. One for private subnets with a default route (`0.0.0.0/0`) to NAT Gateway, and one for public subnets with a default route(`0.0.0.0/0`) to Internet Gateway.
- Associate subnets with proper routing tables.
- Create a new eksctl config with existing VPC and subnets.

2-example.yaml

```yaml
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: existing-vpc
  region: us-east-1
  version: "1.21"
vpc:
  id: vpc-0e88c3c17fb0c7cca
  subnets:
    private:
      us-east-1a:
        id: subnet-0699d559ea175cac2
      us-east-1b:
        id: subnet-049a9d986dc0d9aa5
    public:
      us-east-1a:
        id: subnet-0e4f8a8dda322fac4
      us-east-1b:
        id: subnet-0068c08ea23be99dc
managedNodeGroups:
- name: general
  privateNetworking: true
  instanceType: t3.small
  desiredCapacity: 1
```

- Create a new EKS cluster with a existing VPC.

```
eksctl create cluster -f 2-example.yaml --profile eksctl
```

IAM Roles for Service Accounts

Amazon EKS supports IAM Roles for Service Accounts (IRSA) that allows cluster operators to map AWS IAM Roles to Kubernetes Service Accounts.

This provides fine-grained permission management for apps that run on EKS and use other AWS services. These could be apps that use S3, any other data services (RDS, MQ, STS, DynamoDB), or Kubernetes components like AWS Load Balancer controller or ExternalDNS.

- Let's create an `AllowListAllMyBuckets` IAM policy to allow list all the buckets in this AWS account.

AllowListAllMyBuckets.json

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action": "s3:ListAllMyBuckets",
         "Resource":"*"
      }
   ]
}
```

- Next, create an eksctl config with open ID connect enabled.

3-example.yaml

```
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: cluster-irsa
  region: us-east-1
availabilityZones:
- us-east-1a
- us-east-1b
iam:
  withOIDC: true
  serviceAccounts:
```

```
  - metadata:
      name: foo
      namespace: staging
    attachPolicyARNs:
    - arn:aws:iam::<account_id>:policy/AllowListAllMyBuckets
    # role has to start with eksctl-*
    roleName: eksctl-list-s3-buckets
    roleOnly: true
  - metadata:
      name: cluster-autoscaler
      namespace: kube-system
    wellKnownPolicies:
      autoScaler: true
    roleName: eksctl-cluster-autoscaler
    roleOnly: true
managedNodeGroups:
- name: general
  tags:
    # EC2 tags required for cluster-autoscaler auto-discovery
    k8s.io/cluster-autoscaler/enabled: "true"
    k8s.io/cluster-autoscaler/<cluster-name>: "owned"
  desiredCapacity: 1
  minSize: 1
  maxSize: 10
```

- Create a new EKS cluster with a single autoscaling node group.

```
eksctl create cluster -f 3-example.yaml --profile eksctl
```

When the cluster is created, you can go to the AWS IAM service and find additional IAM roles that were created by eksctl.

- First, let's test if we can list S3 buckets from the pod. Create and apply deployment with aws sdk.

s3.yaml

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: staging
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: foo
  namespace: staging
  annotations:
```

```
    eks.amazonaws.com/role-arn: arn:aws:iam::<account_id>:role/eksctl-list-
s3-buckets
---
apiVersion: v1
kind: Pod
metadata:
  name: aws-cli
  namespace: staging
spec:
  serviceAccountName: foo
  containers:
  - name: aws-cli
    image: amazon/aws-cli
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "while true; do sleep 30; done;" ]
  tolerations:
  - operator: Exists
    effect: NoSchedule
```

- Create a Kubernetes service account and a pod.

```
kubectl apply -f k8s/s3.yaml
```

- Then you need to ssh to the pod by using `exec` command.

```
kubectl exec -it aws-cli -n staging -- bash
```

- You can use aws cli to list S3 buckets in your account.

```
aws s3api list-buckets --query "Buckets[].Name"
```

Deploy Cluster Autoscaler to EKS

Create autoscaller deployment and apply it to the cluster.

cluster-autoscaler.yaml

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cluster-autoscaler
  namespace: kube-system
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::<account_id>:role/eksctl-
cluster-autoscaler
```

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-autoscaler
rules:
  - apiGroups: [""]
    resources: ["events", "endpoints"]
    verbs: ["create", "patch"]
  - apiGroups: [""]
    resources: ["pods/eviction"]
    verbs: ["create"]
  - apiGroups: [""]
    resources: ["pods/status"]
    verbs: ["update"]
  - apiGroups: [""]
    resources: ["endpoints"]
    resourceNames: ["cluster-autoscaler"]
    verbs: ["get", "update"]
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["watch", "list", "get", "update"]
  - apiGroups: [""]
    resources: ["namespaces", "pods", "services", "replicationcontrollers",
"persistentvolumeclaims", "persistentvolumes"]
    verbs: ["watch", "list", "get"]
  - apiGroups: ["extensions"]
    resources: ["replicasets", "daemonsets"]
    verbs: ["watch", "list", "get"]
  - apiGroups: ["policy"]
    resources: ["poddisruptionbudgets"]
    verbs: ["watch", "list"]
  - apiGroups: ["apps"]
    resources: ["statefulsets", "replicasets", "daemonsets"]
    verbs: ["watch", "list", "get"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses", "csinodes", "csidrivers",
"csistoragecapacities"]
    verbs: ["watch", "list", "get"]
  - apiGroups: ["batch", "extensions"]
    resources: ["jobs"]
    verbs: ["get", "list", "watch", "patch"]
  - apiGroups: ["coordination.k8s.io"]
    resources: ["leases"]
    verbs: ["create"]
  - apiGroups: ["coordination.k8s.io"]
    resourceNames: ["cluster-autoscaler"]
    resources: ["leases"]
    verbs: ["get", "update"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
```

```
  name: cluster-autoscaler
  namespace: kube-system
rules:
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["create","list","watch"]
  - apiGroups: [""]
    resources: ["configmaps"]
    resourceNames: ["cluster-autoscaler-status", "cluster-autoscaler-
priority-expander"]
    verbs: ["delete", "get", "update", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-autoscaler
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-autoscaler
subjects:
  - kind: ServiceAccount
    name: cluster-autoscaler
    namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: cluster-autoscaler
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: cluster-autoscaler
subjects:
  - kind: ServiceAccount
    name: cluster-autoscaler
    namespace: kube-system
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cluster-autoscaler
  namespace: kube-system
  labels:
    app: cluster-autoscaler
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cluster-autoscaler
  template:
    metadata:
```

```
      labels:
        app: cluster-autoscaler
    spec:
      serviceAccountName: cluster-autoscaler
      containers:
        - image: k8s.gcr.io/autoscaling/cluster-autoscaler:v1.21.0
          name: cluster-autoscaler
          resources:
            limits:
              cpu: 100m
              memory: 600Mi
            requests:
              cpu: 100m
              memory: 600Mi
          # https://github.com/kubernetes/autoscaler/blob/master/cluster-
autoscaler/cloudprovider/aws/README.md
          command:
            - ./cluster-autoscaler
            - --v=4
            - --stderrthreshold=info
            - --cloud-provider=aws
            - --skip-nodes-with-local-storage=false
            - --expander=least-waste
            - --node-group-auto-discovery=asg:tag=k8s.io/cluster-
autoscaler/enabled,k8s.io/cluster-autoscaler/<cluster-name> # Update cluster
            - --balance-similar-node-groups
            - --skip-nodes-with-system-pods=false
          volumeMounts:
            - name: ssl-certs
              mountPath: /etc/ssl/certs/ca-certificates.crt
              readOnly: true
          imagePullPolicy: "Always"
      volumes:
        - name: ssl-certs
          hostPath:
            path: "/etc/ssl/certs/ca-bundle.crt"
```

- After you create all the objects needed to deploy the autoscaller, apply the YAML.

```
kubectl apply -f k8s/cluster-autoscaler.yaml
```

- Check the pod status, and it should be in running state.

```
kubectl get pods -n kube-system
```

- It's a good idea to check logs for any errors.

```
kubectl logs -l app=cluster-autoscaler -n kube-system -f
```

- Let's list all the nodes in the cluster.

```
watch kubectl get nodes
```

- Create a deployment with five replicas to test autoscaling.

cluster-autoscaler.yaml

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
        resources:
          requests:
            memory: 2Gi
            cpu: 250m
```

```
kubectl apply -f k8s/deployment.yaml
```

Clean

- Delete `eksctl` IAM user
- Delete `EKS` IAM group
- Delete `EksAllAccess` IAM policy
- Delete `IamLimitedAccess` IAM policy
- Delete `AllowListAllMyBuckets` IAM policy
- Delete EKS `eksctl delete cluster -f 3-example.yaml --profile eksctl`