**What is kubeconfig in Kubernetes cluster?**

A kubeconfig file is a configuration file used by kubectl (or other Kubernetes clients) to connect to a Kubernetes cluster.

Basically, kubeconfig file is YAML file that tells kubectl which cluster, how to authenticate, and which context to use. Default path: ~/.kube/config. For EKS, aws eks update-kubeconfig builds it for you.

It contains:

* Cluster details (API server endpoint, certificate authority)
* User authentication details (access token, IAM role, client certificate)
* Context information (which cluster + which user to use by default)

By default, kubectl looks for this file at:

>> ~/.kube/config

But you can specify a different file like this:

>> kubectl --kubeconfig /path/to/your/kubeconfig get pods

So, kubeconfig is your "credentials + connection info" for a k8s cluster.

**Connecting to a Private Endpoint EKS Cluster**

When an EKS cluster endpoint is private, it cannot be accessed from the public internet. You must be within a network that can reach the VPC hosting the cluster.

1. **Prerequisites (do these once)**

* Tools
* AWS CLI v2 (aws --version)
* kubectl (same major/minor as your cluster or ±1 is usually fine)
* AWS permissions for your IAM principal
* eks:DescribeCluster at minimum to generate kubeconfig
* (Optional) rights to assume an admin role if you’ll edit RBAC
* Know your: CLUSTER\_NAME, REGION, and VPC access method (bastion/VPN/etc.)

1. **Understand private-endpoint EKS access (what must be true)**

When the EKS API server is **private**, kubectl can only reach it **from inside the VPC or connected networks**. So you must use **one** of these paths:

**A. Bastion/Jump EC2** inside the VPC (SSH or Session Manager)

**B. VPN into the VPC** (AWS Client VPN, Site-to-Site, TGW, or Direct Connect)

**C. From a peered VPC** or via Transit Gateway (ensure routing + SG rules)

Your laptop on the public internet **cannot** hit a private endpoint directly.

1. **Generate kubeconfig (safe to run anywhere)**

This only talks to the EKS control plane API (public AWS API), not your cluster endpoint.

# If you use named profile:

export AWS\_PROFILE=my-profile

export REGION=ap-south-1

export CLUSTER=your-cluster-name

aws eks --region $REGION update-kubeconfig --name $CLUSTER

# Optional: if you must assume a role to access the cluster:

# aws eks --region $REGION update-kubeconfig --name $CLUSTER --role-arn arn:aws:iam::123456789012:role/EKSAdminRole

This writes a context like arn:aws:eks:REGION:ACCOUNT:cluster/CLUSTER into ~/.kube/config.

Note: kubectl commands will **still fail** from your laptop until you’re on a network that can reach the **private** endpoint.

1. **Choose a connectivity path and follow the steps**

**A) Bastion/Jump EC2 (fastest to set up)**

* Launch an EC2 in the same VPC (any subnet that can route to the EKS endpoint).

AMI: Amazon Linux 2 or 2023 is fine. Size: t3.micro is enough for kubectl.

* **Security Groups**

Attach a client SG to your EC2 (allow outbound).

Find the EKS cluster security group (controls API access):

>> aws eks describe-cluster --name $CLUSTER --region $REGION \

--query "cluster.resourcesVpcConfig.clusterSecurityGroupId" --output text

Allow inbound HTTPS (443) on that cluster SG from your EC2’s SG.

This is critical; otherwise the API rejects connections.

* Access the EC2

Prefer AWS Systems Manager Session Manager (no SSH keys, no public IP).

Attach IAM role AmazonSSMManagedInstanceCore to the EC2 and open SSM Session.

Or use SSH to a public bastion that can reach the private subnets.

* **Install tools on the EC2**

>> sudo yum install -y unzip

# kubectl (example for v1.29; choose your cluster's minor)

>> curl -LO https://dl.k8s.io/release/v1.29.6/bin/linux/amd64/kubectl

>> chmod +x kubectl && sudo mv kubectl /usr/local/bin/

# AWS CLI v2 is preinstalled on AL2/AL2023; if not, install it

* **Put kubeconfig on the EC2**

Either re-run:

>> aws eks --region $REGION update-kubeconfig --name $CLUSTER

Or copy your local ~/.kube/config to the EC2 (if you already generated it).

* **Test**

>> kubectl cluster-info

>> kubectl get nodes

**B) VPN into the VPC (no bastion, you use your own laptop)**

* **Create an AWS Client VPN endpoint (or use an existing Site-to-Site/VPN).**

Attach to at least one target subnet in the EKS VPC.

Add a route for the VPC CIDR to the Client VPN.

Add authorization rules for your users.

* **Security Groups**

Ensure the cluster security group allows TCP 443 from the subnets/CIDRs used by the VPN clients (or from a dedicated client SG if you hairpin through an ENI).

* **Connect the VPN from your laptop.**
* **Use kubectl locally (kubeconfig already generated in step 3):**

>> kubectl get ns

**C) From a peered VPC / Transit Gateway / Direct Connect**

* Ensure routing is in place (your source network must know the EKS VPC CIDR, and vice-versa).
* Security Groups: allow 443 on the cluster SG from the source (peer VPC SG or on-prem CIDR via TGW/DX).
* Test with:

# From a host in the connected network

>> curl -k https://<private-eks-endpoint-dns>

# should return a TLS error or 403, not timeout

>> kubectl get pods -A

**5) Make sure your IAM ↔ RBAC mapping exists (common “Unauthorized” fix)**

Even with network access, you’ll get:

You must be logged in to the server (Unauthorized)

unless your IAM principal is mapped in **aws-auth** ConfigMap.

**If you have cluster admin already** (e.g., the role that created the cluster):

# Map an IAM role to cluster-admin (example)

>> kubectl edit configmap aws-auth -n kube-system

# Add something like:

# mapRoles: |

# - rolearn: arn:aws:iam::123456789012:role/EKSAdminRole

# username: admin

# groups:

# - system:masters

**If you don’t have admin yet**, ask an existing admin to add your IAM user/role to aws-auth.

Tip: With eksctl, you can do:

eksctl create iamidentitymapping \

--cluster $CLUSTER \

--region $REGION \

--arn arn:aws:iam::123456789012:role/EKSAdminRole \

--group system:masters \

--username admin

**6) Troubleshooting cheat-sheet**

* **Unable to connect to the server: dial tcp … timeout**  
  Network path missing. Confirm you’re on VPC/VPN; check routes; check that **cluster SG** allows 443 from your client SG/CIDR.
* **certificate signed by unknown authority**  
  Regenerate kubeconfig: aws eks update-kubeconfig --name $CLUSTER --region $REGION.
* **No Auth Provider found for name "aws"**  
  Update kubectl to a modern version (uses exec plugin) and ensure AWS CLI v2 is present.
* **Unauthorized**  
  Your IAM principal isn’t mapped in aws-auth. Add it (step 5).
* **From laptop it fails, from EC2 it works**  
  That’s expected if the endpoint is private and your laptop isn’t on VPN.

**7) Minimal working example (bastion route)**

# On your laptop (public internet is fine for this step)

>> aws eks --region ap-south-1 update-kubeconfig --name my-private-eks

# Start SSM Session to a small EC2 in same VPC (recommended)

# (In AWS Console: Session Manager -> Start session -> select instance)

# Inside the EC2 shell:

>> aws eks --region ap-south-1 update-kubeconfig --name my-private-eks

>> kubectl get nodes