**Kubeconfig in K8s cluster**

kubeconfig is a configuration file that stores information required by the kubectl command-line tool (or other Kubernetes clients) to communicate with a cluster.

It contains details like:

* **Clusters** → API server addresses and certificate information.
* **Users (Credentials)** → Authentication details (certificates, tokens, etc.).
* **Contexts** → A combination of a cluster, a user, and a namespace to define the working environment.
* **Current-context** → The default context used when you run kubectl commands without explicitly specifying a context.
* the kubeconfig file is located at:

~/.kube/config

Kubeconfig acts like a "login + environment setup" file for Kubernetes, telling kubectl **which cluster to connect to, which user credentials to use, and which namespace to default to.**

Use kubeconfig files to organize information about clusters, users, namespaces, and authentication mechanisms.

The kubectl command-line tool uses kubeconfig files to find the information it needs to choose a cluster and communicate with the API server of a cluster.

A file that is used to configure access to clusters is called a *kubeconfig file*.

kubectl looks for a file named config in the $HOME/.kube directory. You can specify other

kubeconfig files by setting the KUBECONFIG environment variable or by setting the [-- kubeconfig](https://kubernetes.io/docs/reference/generated/kubectl/kubectl/) flag.

In a **kubeconfig**, a **context** is simply a shortcut that defines which cluster, which user credentials, and which namespace kubectl should use by default.

**Ways to Connect to a Private EKS Cluster**

**Step 1: Get your kubeconfig file**

On your **local machine** (not inside VPC yet):

>>aws eks update-kubeconfig --region <region> --name <cluster-name>

Example:

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

This writes cluster details into ~/.kube/config.

If it’s **private endpoint**, your laptop won’t be able to connect yet. That’s why we need the next steps.

**Step 2: Launch a Bastion EC2 in the same VPC**

1. Create an EC2 inside the **same VPC & subnet** as the EKS cluster.
2. Install kubectl & awscli inside that EC2.

Commands (run inside EC2 after launch):

# Update packages

>>sudo yum update -y

# Install AWS CLI v2

>>curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

>>unzip awscliv2.zip

>>sudo ./aws/install

# Install kubectl (replace version with your cluster's version, e.g., 1.28)

>>curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.28.0/2023-11-14/bin/linux/amd64/kubectl

>>chmod +x ./kubectl

>>sudo mv ./kubectl /usr/local/bin/kubectl

# Verify

>>kubectl version --client

>>aws --version

**Step 3: Copy kubeconfig to Bastion EC2**

>>scp -i mykey.pem ~/.kube/config [ec2-user@<BASTION\_PUBLIC\_IP>:~/.kube/config](mailto:ec2-user@%3cBASTION_PUBLIC_IP%3e:~/.kube/config)

Step4: **IAM Authentication**

* EKS doesn’t use certificates by default; instead, it relies on **IAM authentication** with aws-iam- authenticator (built into AWS CLI v2).
* Ensure your AWS CLI has credentials for a user/role with access to EKS.
* Your IAM entity must be listed in the **aws-auth ConfigMap** in the cluster to have Kubernetes RBAC permissions.

**Step 4: Run kubectl inside Bastion**

SSH into Bastion:

>>ssh -i mykey.pem ec2-user@<BASTION\_PUBLIC\_IP>

Then test:

>>kubectl get nodes

>>kubectl get pods -A