Kubeconfig in Kubernetes and Connecting to Private Endpoint EKS Cluster

# 1. What is meant by kubeconfig in Kubernetes cluster?

* A kubeconfig file is a configuration file used by kubectl and other Kubernetes tools to access and manage Kubernetes clusters. It is usually stored at ~/.kube/config by default.  
    
  The kubeconfig file contains the following key information:  
  - Clusters: Information about Kubernetes clusters (API server endpoint, certificate authority data).  
  - Users: Authentication details for accessing the clusters (credentials, exec plugin for token fetching).  
  - Contexts: A combination of cluster and user that specifies which cluster a user is interacting with.  
  - Current-context: The default context used by kubectl if not explicitly specified.  
    
  In short, kubeconfig acts as a map between users, clusters, and authentication methods, allowing developers to seamlessly switch between multiple Kubernetes clusters.

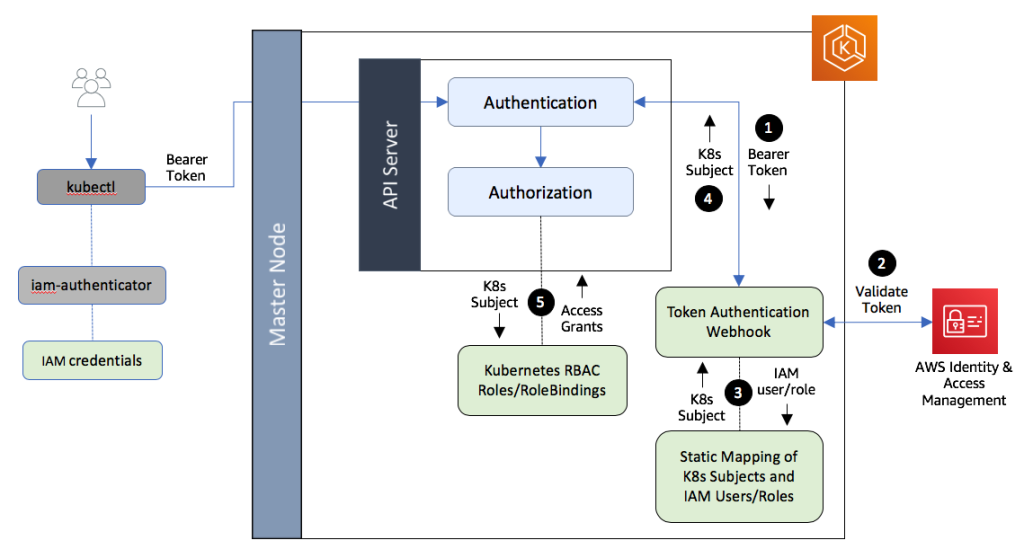
# 2. How to connect to a Private Endpoint EKS Cluster?

When an Amazon EKS (Elastic Kubernetes Service) cluster is configured with a private endpoint, its Kubernetes API server is accessible only within the VPC or through connected networks such as VPN, VPC Peering, Transit Gateway, or AWS Direct Connect. You cannot connect directly over the internet.  
  
Step-by-step process to connect:

* 1. Ensure network access: The machine (or environment) where you run kubectl must have network connectivity to the EKS VPC. This can be achieved via:  
   - Launching an EC2 instance in the same VPC (bastion host).  
   - Connecting via AWS VPN or Direct Connect.  
   - Using AWS Systems Manager (SSM) Session Manager for tunneling.
* 2. Configure AWS CLI: Install AWS CLI v2 and configure it with proper IAM credentials that have permissions like eks:DescribeCluster.
* 3. Install kubectl: Ensure kubectl is installed on the system.
* 4. Update kubeconfig: Run the command:  
   aws eks update-kubeconfig --region <region> --name <cluster-name>  
   This command fetches cluster details and updates your kubeconfig file.
* 5. (If using a bastion/SSM tunnel): Modify the kubeconfig to point the cluster server field to localhost:   
   Example: https://localhost:8443
* 6. Verify connection:  
   kubectl get nodes  
   kubectl get pods -A  
    
   If the connection works, you will see the list of nodes or pods in the cluster.

In summary, kubeconfig provides the authentication and cluster information needed by kubectl, while accessing a private EKS endpoint requires network reachability into the cluster's VPC through methods such as EC2 bastion, VPN, or SSM.

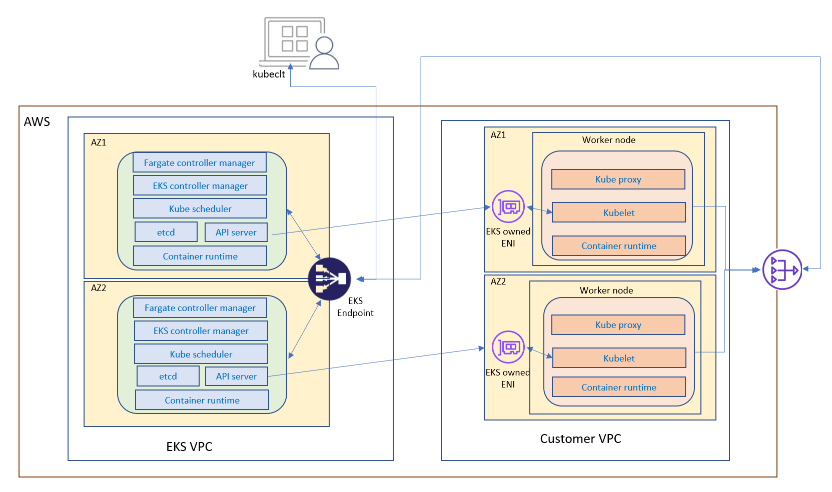
Workflow & Architecture



**Diagram 1 (Cluster Networking & Components)**

* This diagram shows the **infrastructure and network-level view** of an EKS cluster.
* It explains that:
  + The **control plane** (API server, etcd, scheduler, controller manager) runs inside the AWS-managed VPC.
  + The **worker nodes** (EC2 or Fargate) run inside the **customer’s VPC**.
  + Communication between control plane and worker nodes happens through **ENIs (Elastic Network Interfaces)**.
  + A kubectl user connects to the **EKS API endpoint** (public or private).

👉 Focus: **Infrastructure & communication flow**.



**Diagram 2 (Authentication & Authorization)**

* This diagram shows the **security and authentication workflow** when you interact with the EKS API using kubectl.
* When you run a kubectl command:
  + AWS IAM credentials are used to get a **Bearer Token**.
  + The **API Server** validates this token with **AWS IAM**.
  + IAM role/user is mapped to a **Kubernetes subject** through the aws-auth ConfigMap.
  + **RBAC (Role-Based Access Control)** in Kubernetes decides whether the action is allowed.

👉 Focus: **Security (IAM + RBAC) & API request workflow**.

👉 Diagram Flow:  
**User → Kubeconfig → AWS IAM → EKS API Server → RBAC → Worker Nodes (via ENI) → Response back to User**