Module 1: Creating User for RBAC

We need to create a user with the following specification:

username: john

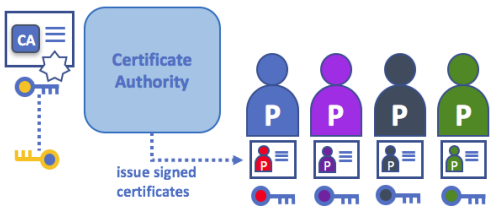
User needs to authenticate via certificates to the K8s cluster.

John

There are two approaches to deal with this:

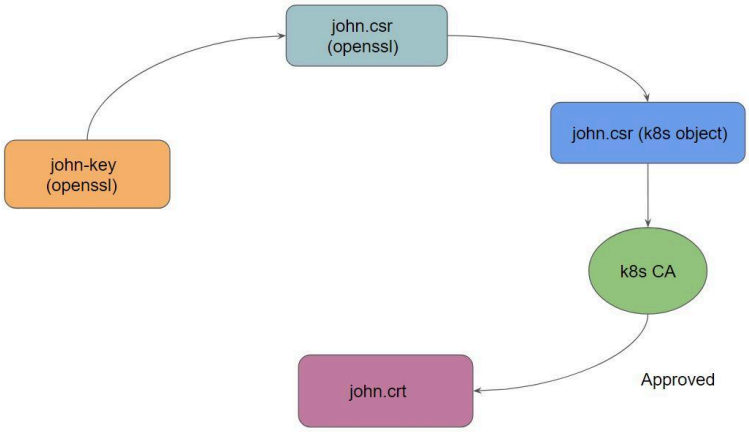
Approach 1:

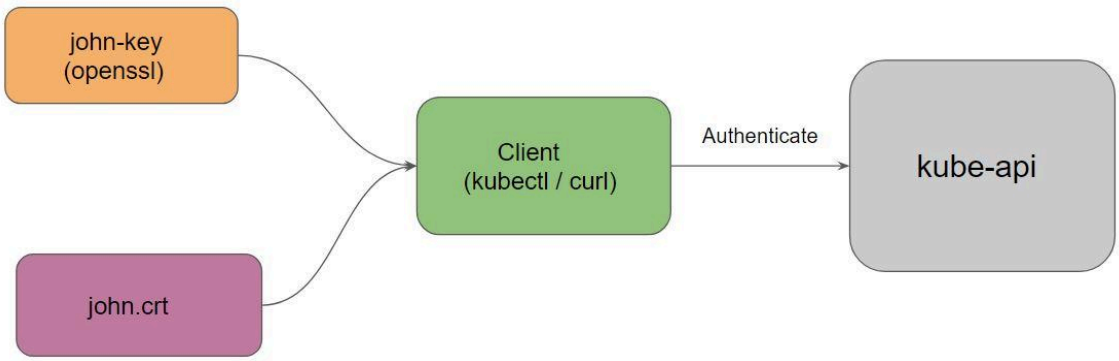
- Create a private key for user john.
- Create the CSR for user john.
- Sign the CSR with CA Certificate + CA Key available at a specific location.



Approach 2:

- Create private key for user john.
- Create the CSR for user john.
- Create a CertificateSigningRequest
- Approve the CSR
- Get the certificate





Module 2: Role-Based Access Control

2.1 Use-Case:

User John has joined the organization as a developer.

He needs basic permissions to create PODS and read POD related information.

John

2.2 Basics of Role and Role Binding

A role contains rules that represent a set of permissions A role binding grants the permissions defined in a role to a user or set of users.

John

Role Binding:

ReadSome
- zeal
- john
- alice

Role ReadSome

Allow:
Read pods
Read services
Read secrets

2.3 API Group

API groups make it easier to extend the Kubernetes API.

There are several API groups in Kubernetes:

API Groups	Description
Core API Group (Legacy)	It is found at REST path /api/v1.
Named Groups	Available at REST path /apis/$GROUP_NAME/$VERSION and use apiVersion: $GROUP_NAME/$VERSION (for example, apiVersion: batch/v1).

2.4 Checking API Access

kubectl provides the auth can-i subcommand for quickly querying the API authorization layer.

It allows users to determine if the current user can perform a given action

```
john@kubeadm-master:~$ kubectl auth can-i create deployments
no
```

Module 3: ClusterRole and ClusterRoleBinding

A Role can only be used to grant access to resources within a single namespace.

A ClusterRole can be used to grant the same permissions as a Role, but because they are cluster-scoped, they can also be used to grant access to:

cluster-scoped resources (like nodes)
namespaced resources (like pods) across all namespaces

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list","create"]
```

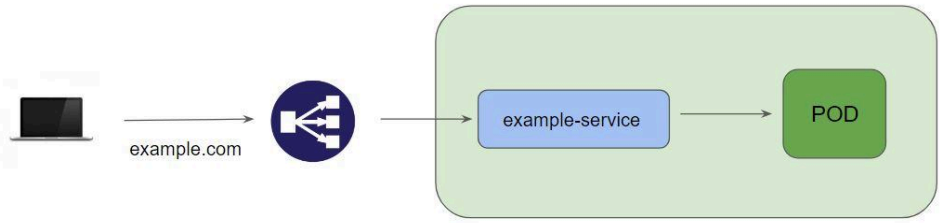
Important Pointer

A RoleBinding may also reference a ClusterRole to grant the permissions to namespaced resources defined in the ClusterRole within the RoleBinding's namespace

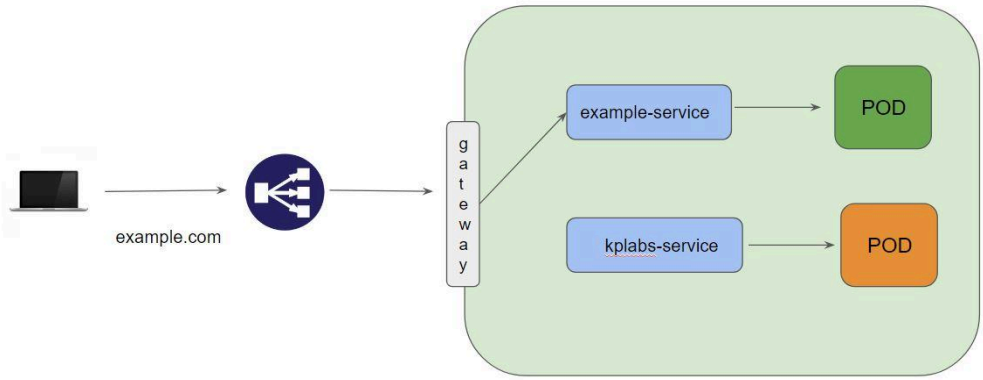
This allows the administrator to have a set of central policy which can be attached via RoleBinding so it is applicable at a per-namespace level.

Module 4: Ingress

Whenever making use of the LoadBalancer Service Type, out of the box, you can make use of a single website.



With ingress, we can set up multiple rules through which traffic can be routed to an appropriate service depending on the URL of the website that is requested.



Kubernetes Ingress is a collection of routing rules which governs how external users access the services running within the Kubernetes cluster.

Ingress can provides various features which includes:

- Load Balancing
- SSL Termination
- Named-based virtual hosting

Module 5: Ingress Resource and Ingress Controllers

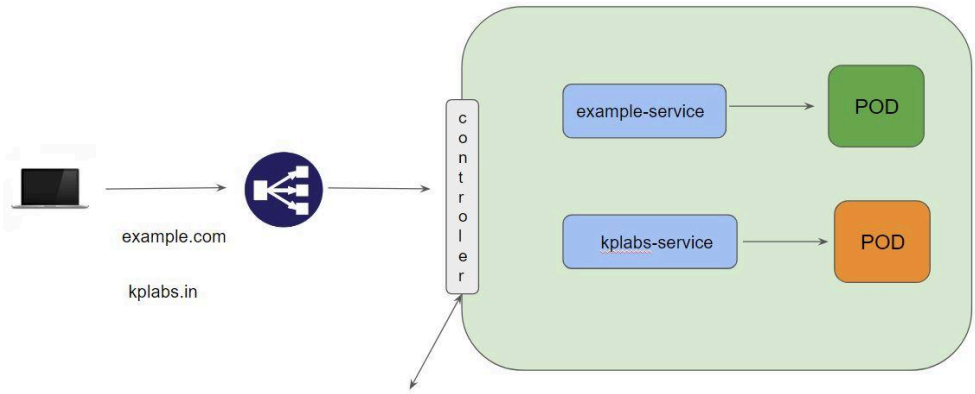
There are two sub-components when we discuss about Ingress:

- Ingress Resource
- Ingress Controllers

Ingress Resource contains set of routing rules based on which traffic is routed to a service.

Ingress Controller takes care of the Layer 7 proxy to implement ingress rules.

You must have an ingress controller to satisfy an Ingress. Only creating an Ingress resource has no effect

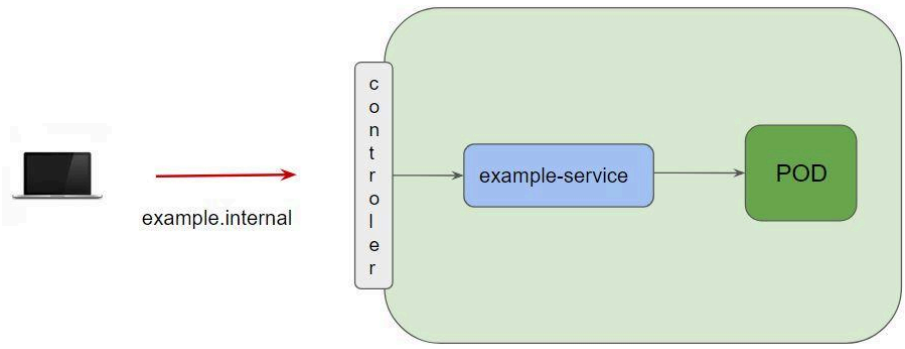


Rule	Host	Service
1	example.com	example-service
2	kplabs.in	kplabs-service

Module 6: Ingress Resource and Ingress Controllers

A HTTP based connection to the ingress controller is not a secure.

A HTTP based connection to the ingress controller is not a secure.



Following is a sample code snippet

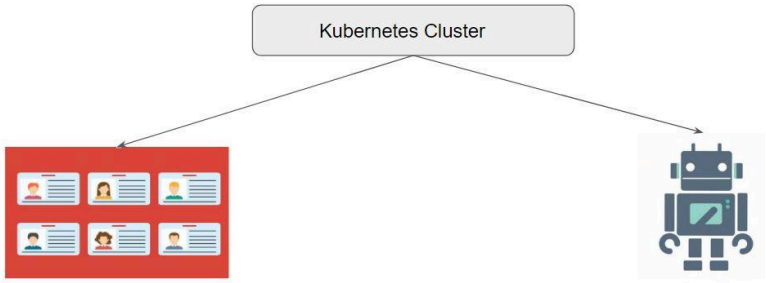
```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: demo-ingress
  annotations:
     nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  tls:
  - hosts:
      - example.internal
    secretName: tls-cert
  rules:
  - host: example.internal
```

Module 7: Service Account

13.1 Understanding Authentication

Kubernetes Clusters have two categories of users:

- Normal Users.
- Service Accounts

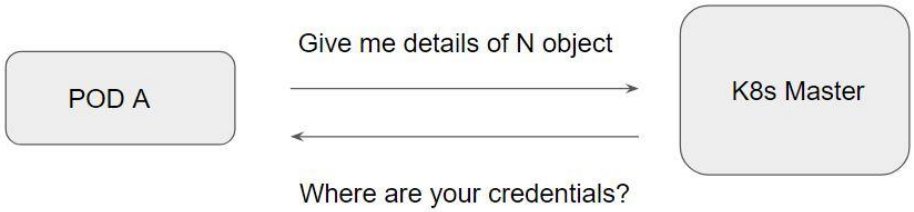


13.2 Understanding Service Accounts

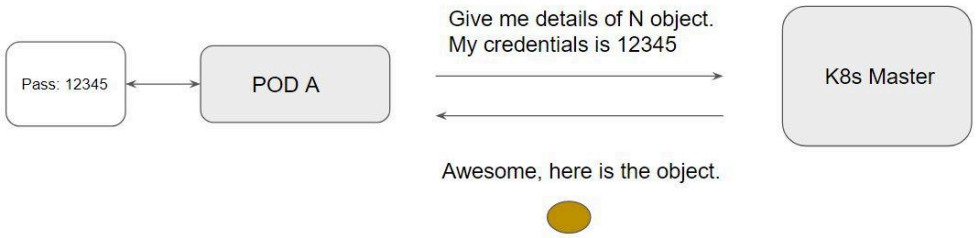
Service Accounts allows the Pods to communicate with the API Server

Let's understand with a use-case:

Application within Pod A wants to retrieve an object within your K8S cluster.



Following diagram indicates the working of service account:



13.3 Important Pointer

Service Accounts are namespaced.

Default service account gets automatically created when you create a namespace

PODS are automatically mounted with the default service accounts.

Module 8: Upgrading kubeadm

Following are the high-level steps required to perform kubeadm upgrade.

- Find the latest version to upgrade to.
- Unhold the kubeadm binary
- Download the latest kubeadm binary (apt-get install)
- Drain the control plane node
- Perform the Upgrade Plan
- Choose version to upgrade and run appropriate command
- Upgrade the kubelet and kubectl binaries

Join Our Discord Community

We invite you to join our Discord community, where you can interact with our support team for any course-based technical queries and connect with other students who are doing the same course.

Joining URL:

http://kplabs.in/chat