Security

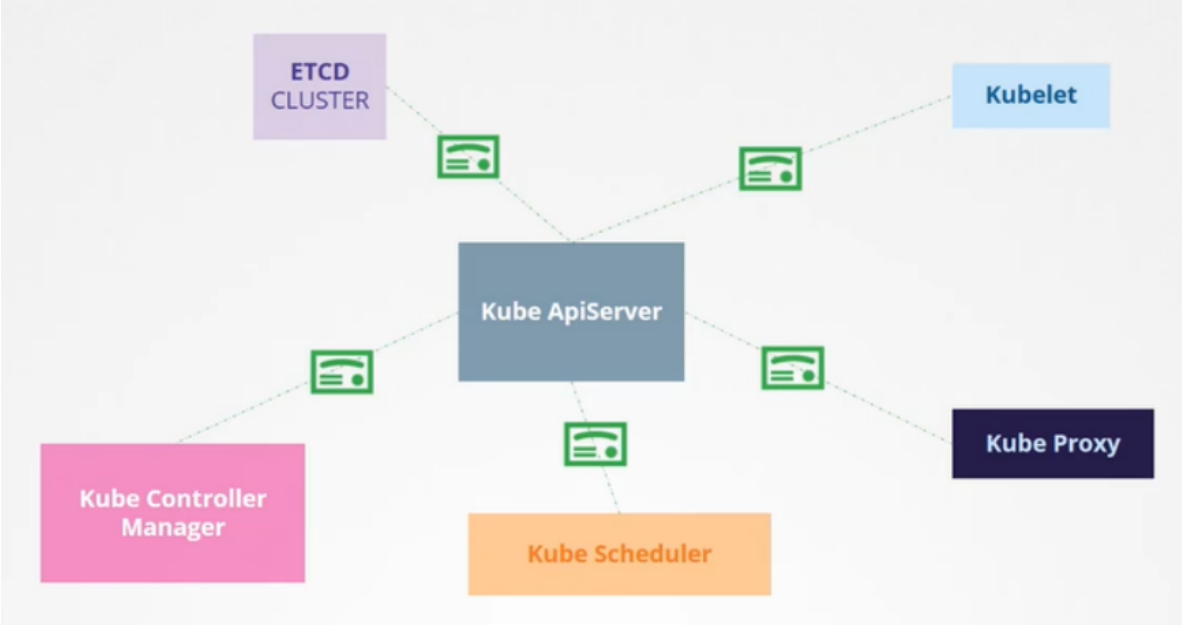
Of course all access to these hosts must be secured, root access disabled, password based authentication disabled, and only SSH key based authentication to be made available.

What are the risks and what measures do you need to take to secure the cluster? As we have seen already, the kube-api server is at the center of all operations within kubernetes. We interact with it through the kubectl utility or by accessing the API directly and through that you can perform almost any operation on the cluster. So that's the first line of defense. **Controlling access to the API server itself.**
We need to make two types of decisions.

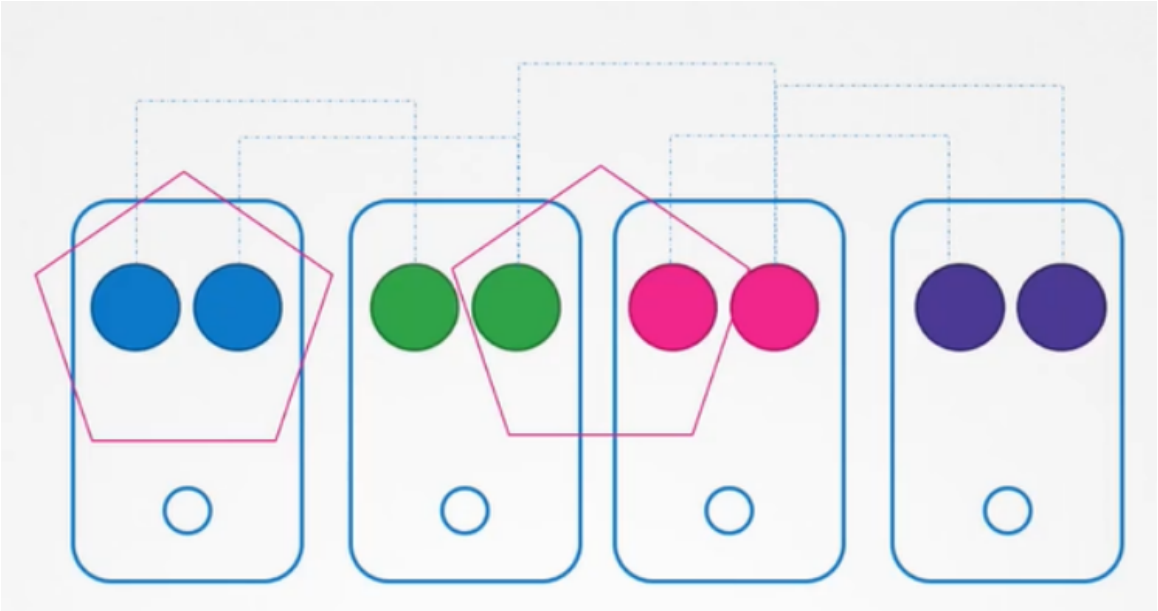
1. **Who can access?**
    - Who can access the API server is defined by the **Authentication** mechanisms.
2. **What can they do?**
    - Once they gain access to the cluster, what they can do is defined by **Authorization** mechanisms.

TLS Certificates

- All communication with the cluster, between the various components such as the ETCD Cluster, kube-controller-manager, scheduler, api server, as well as those running on the working nodes such as the kubelet and kubeproxy is secured using TLS encryption.



What about communication between applications within the cluster?
By default all PODs can access all other PODs within the cluster. You can restrict access between them using **Network Policies**.



.

.

.

Authentication

So, let's talk about the different users that may be accessing the cluster security of end users who access the applications deployed on the cluster is managed by the applications themselves internally.

Our focus is on users access to the Kubernetes cluster for administrative purposes.

So we are left with two types of users humans, such as the administrators and developers and robots such as other processes or services or applications that require access to the cluster.

Kubernetes does not manage user accounts natively. It relies on an external source, like a file with user details or certificates or a third party identity service like LDAP to manage these users.

You cannot create users in a Kubernetes cluster or view the list of users.

However, in case of service accounts, Kubernetes can manage them. You can create and manage service accounts using the Kubernetes API.

```
kubectl create serviceaccount sa1
```

We have a section on service accounts exclusively where we discuss and practice more about service accounts.

```
kubectl list serviceaccont
```

We will focus on users in Kubernetes. **All user access is managed by the API server.**

There are different authentication mechanisms that can be configured:

- You can have a list of username and passwords in a static password file
- Usernames and tokens in a static token file.
- Or you can authenticate using certificates.

- And another option is to connect to third party authentication protocols like LDAP, Kerberos, etc.

1. Let's start with the simplest form of authentication. You can create a list of users and their passwords in a CSV file and use that as the source for user information.

The file has three columns password, username and user ID.

We then pass the file name as an option to the API server.

```
--basic-auth-file=user-details.csv
```

You must then restart the API server for these options to take effect!

kube-apiserver configuration

- If you set up via kubeadm then update kube-apiserver.yaml manifest file with the option.

/etc/kubernetes/manifests/kube-apiserver.yaml

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --authorization-mode=Node,RBAC
    - --advertise-address=172.17.0.107
    - --allow-privileged=true
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --basic-auth-file=user-details.csv
    image: k8s.gcr.io/kube-apiserver-amd64:v1.11.3
    name: kube-apiserver
```

- To authenticate using the basic credentials while accessing the API server. Specify the user and password in a core command like this.

```
curl -v -k https://master-node-ip:6443/api/v1/pods -u "user1:password123"
```

- In the CSV file with the user details that we saw, we can optionally have a fourth column with the group details to assign users to specific groups:

user-details.csv

```
password123,user1,u0001,group1
password123,user2,u0002,group1
password123,user3,u0003,group2
password123,user4,u0004,group2
password123,user5,u0005,group2
```

Similarly, **instead of a static password file, you can have a static token file.**

user-token-details.csv

```
KpjCVbI7rCFAHYPkByTIzRb7gu1cUc4B,user10,u0010,group1
rJjncHmvtXHc6MlWQddhtvNyyhgTdxSC,user11,u0011,group1
mjpOFIEiFOkL9toikaRNtt59ePtczZSq,user12,u0012,group2
PG41IXhs7QjqwWkmBkvgGT9glOyUqZij,user13,u0013,group2
```

- Pass the token file as an option token auth file to the cube API server while authenticating:

```
curl -v -k https://master-node-ip:6443/api/v1/pods --header "Authorization: Bearer
KpjCVbI7rCFAHYPkBzBb7gu1cUc48"
```

Note

- This is not a recommended authentication mechanism
- Consider volume mount while providing the auth file in a kubeadm setup
- Setup Role Based Authorization for the new users

LINKS:

https://kubernetes.io/docs/reference/access-authn-authz/authentication/

.

.

.

Authorization

So we will be creating accounts for them to access the cluster by creating usernames and passwords or tokens or signed certificates or service accounts, as we saw in the previous part.

But we don't want all of them to have the same level of access as us, for example.
We don't want the developers to have access to modify our cluster configuration, like adding or deleting nodes or the storage or networking configurations. We can allow them to view, but not modify. But they could have access to deploying applications. The same goes with service accounts. We only want to provide the external application, the minimum level of access to perform its required operations.
When we share our cluster between different organizations or teams by logically partitioning it using namespaces, we want to restrict access to the users to their namespaces alone. That is what authorization can help you within the cluster.

Authorization Mechanisms

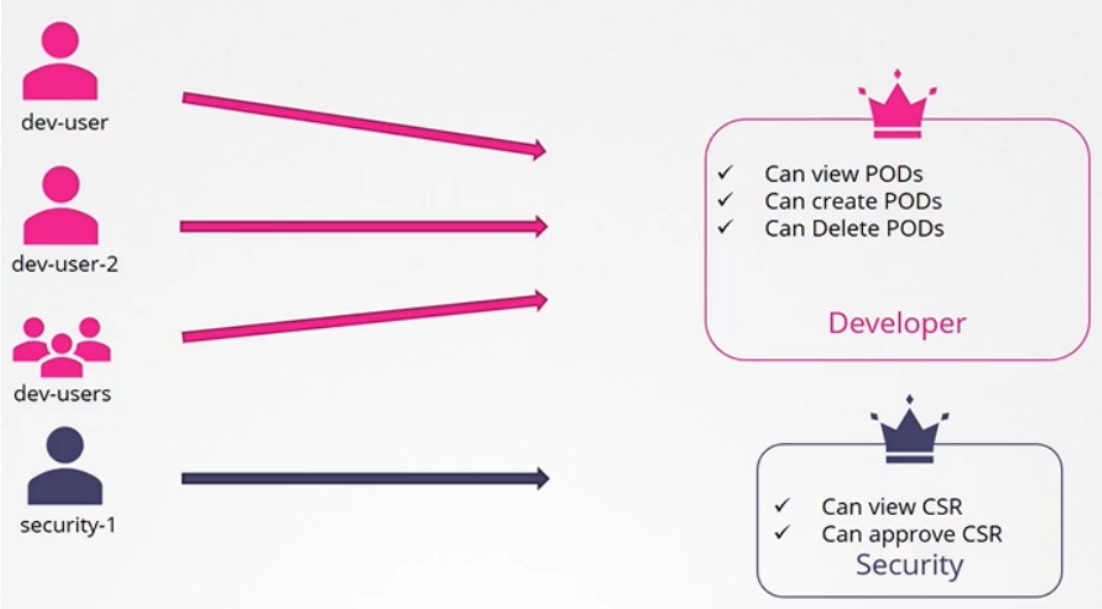
- There are different authorization mechanisms supported by kubernetes:
  - Node Authorization
  - Attribute-based Authorization (ABAC) - is where you associate a user or a group of users with a set of permissions (creating a policy file with a set of policies defined in a JSON format).
    You pass this file into the API server.
    Similarly, we create a policy definition file for each user or group in this file.
    Now, every time you need to add or make a change in the security, you must edit this policy file manually and restart the API server. **Difficult to manage!**
  - Role-Based Authorization (RBAC)
  - Webhook

RBAC

Role based access controls make this much easier with roll based access controls instead of directly associating a user or a group with a set of permissions.

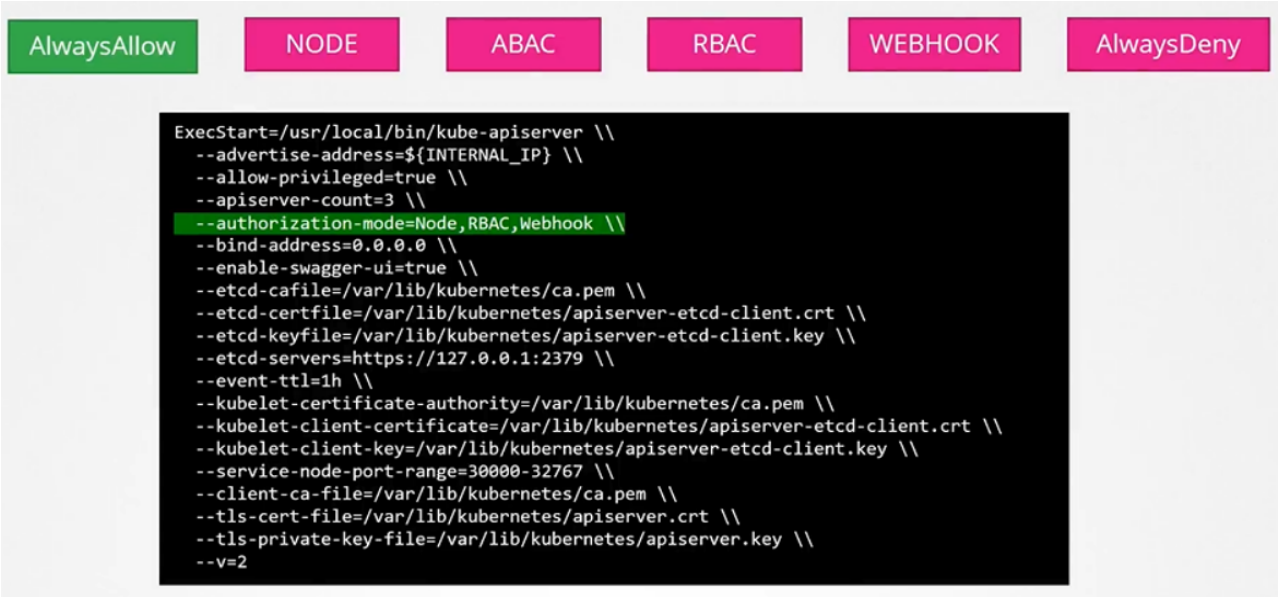
We define a rule in this case for developers. We create a role with the set of permissions required for developers. Then we associate all the developers to that role.

Similarly, create a role for security users with the right set of permissions required for them, then associate the user to that role going forward. Whenever a change needs to be made to the users access, we simply modify the role and it reflects on all developers immediately.



Authorization Modes

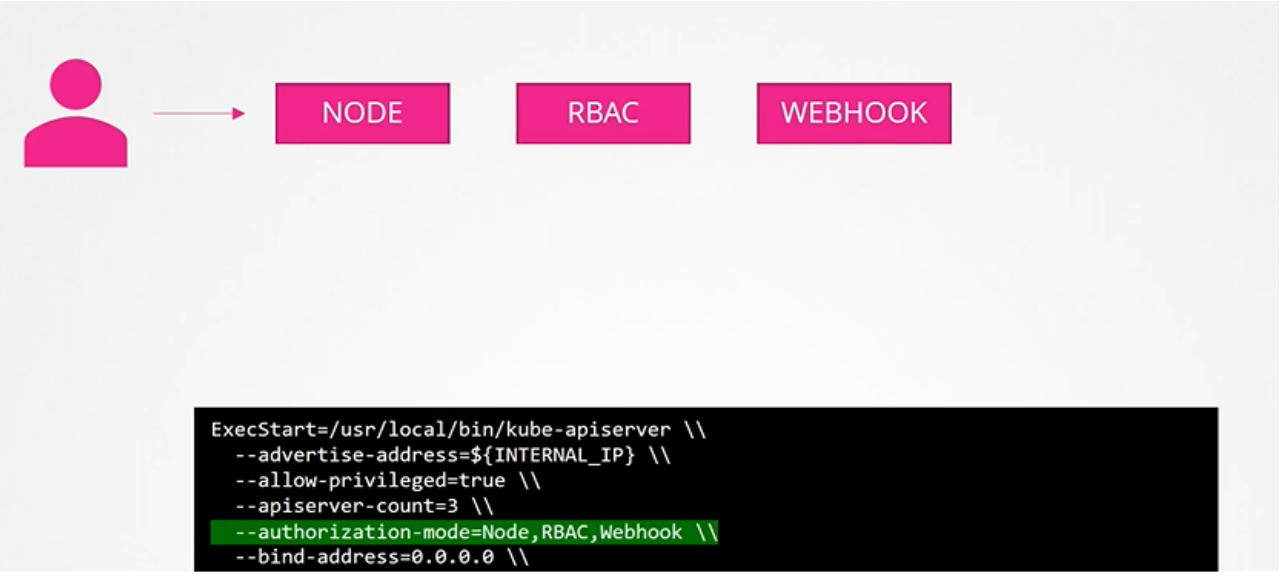
The modes are set using the authorization mode option on the API server. If you don't specify this option, it is set to always allow. By default you may provide a comma separated list of multiple modes that you wish to use. In this case, I want to set it to Node or RBAC or Webhook.



When you have multiple modes configured, your request is authorized using each one in the order it is specified(*в указанном порядке).

For example, when a user sends a request, it's first handled by the Node authorizer. The node authorizer handles only no requests. So it denies the request.

Whenever a module denies a request, it is forwarded to the next one in the chain.



The Role based Access Control module performs its tests and grants. The user permission authorization is complete and user is given access to the requested object!

.

.

.

Role Based Access Controls (RBAC)

How do we create a role?

We do that by creating a role object. So we create a role definition file with the API version set to **rbac.authorization.K8s.io/v1** and kind said to role. We need the role developer as we are creating this role for developers and then we specify rules.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: developer
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "list", "update", "delete", "create"]
- apiGroups: [""]
  resources: ["ConfigMap"]
  verbs: ["create"]
```

- Each role has 3 sections:
  - apiGroups
  - resources
  - verbs
    - We can add multiple rules for a single role like this.

- Create the role with kubectl command:

```
kubectl create -f developer-role.yaml
```

The next step is to link the user to that role.

- For this we create another object called **RoleBinding**:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: devuser-developer-binding
subjects:
- kind: User
  name: dev-user # "name" is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: developer
  apiGroup: rbac.authorization.k8s.io
```

The role binding object links a user object to a role.

- Create the role binding using kubectl command:

```
kubectl create -f devuser-developer-binding.yaml
```

Also note that the roles and role bindings fall under the scope of namespace.

So here the dev-user gets access to pods and configmaps within the default namespace. If you want to limit the dev user's access within a different namespace then specify the namespace within the metadata of the definition file while creating them.

View RBAC

- To list roles:

```
kubectl get roles
```

- To list rolebindings:

```
kubectl get rolebindings
```

- To describe role:

```
kubectl describe role developer
```

- To describe rolebinding:

```
kubectl describe rolebinding devuser-developer-binding
```

What if you being a user would like to see if you have access to a particular resource in the cluster?

Check Access

- You can use the kubectl auth command:

```
kubectl auth can-i create deployments
```

```
kubectl auth can-i delete nodes
```

```
kubectl auth can-i create deployments --as dev-user
kubectl auth can-i create pods --as dev-user
```

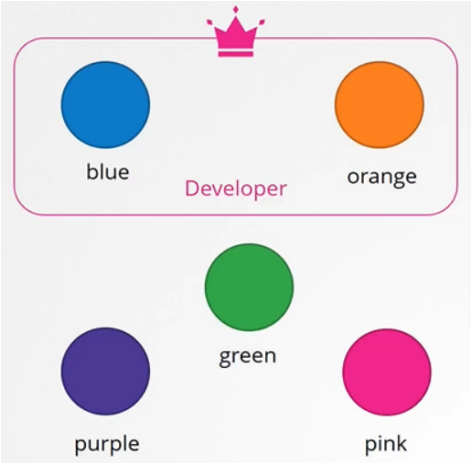
```
kubectl auth can-i create pods --as dev-user --namespace test
```

Resource Names

We just saw how you can provide access to users for resources like pods within the namespace. You can go one level down and allow access to specific resources alone.

For example say you have five pods in namespace. You want to give access to a user to pods, but not all pods. You can restrict access to the blue and orange pod alone by adding a **resourceNames** field to the rule.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: developer
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "update", "create"]
  resourceNames: ["blue", "orange"]
```



LINKS:

https://kubernetes.io/docs/reference/access-authn-authz/rbac/

https://kubernetes.io/docs/reference/access-authn-authz/rbac/#command-line-utilities

.

.

.

Cluster Roles

We said that roles and role bindings are namespaced meaning they are created within namespaces. If you don't specify in namespace they're created in the default namespace and control access within that namespace alone in one of the previous part we discussed about namespaces and how it helps in grouping or isolating resources like pods, deployments and services. But what about other resources like nodes?

Can you group or isolate nodes within a namespace?

Like can you say node 01 is part of the dev namespace. **No!**

These are cluster-wide resources. They cannot be associated with any particular namespace.

Кластерные ресурсы - это ресурсы, для которых не указывается пространство имен, когда вы их создаете. Например, nodes, persistent volumes, persistent clusterroles and clusterrolebinding, которые мы собираемся рассмотреть.

- To see **namespaced** resources:

```
kubectl api-resources --namespaced=true
```

- To see **non-namespaced** resources:

```
kubectl api-resources --namespaced=false
```

Cluster Roles and Cluster Role Bindings

We authorize users to cluster wide resources like nodes or persistent volumes that is where you use cluster roles and cluster role bindings.

They are for a cluster scoped resources for example a cluster admin role can be created to provide a cluster administrator permissions to view create or delete nodes in a cluster.

- Cluster Roles are roles except they are for a cluster scoped resources. Kind as **CLusterRole**:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-administrator
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["nodes"]
  verbs: ["get", "list", "delete", "create"]
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin-role-binding
subjects:
- kind: User
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-administrator
  apiGroup: rbac.authorization.k8s.io
```

```
kubectl create -f cluster-admin-role.yaml
kubectl create -f cluster-admin-role-binding.yaml
```

You can create a cluster role for namespace resources as well. When you do that user will have access to these resources across all namespaces.

LINKS:

https://kubernetes.io/docs/reference/access-authn-authz/rbac/#role-and-clusterrole

https://kubernetes.io/docs/reference/access-authn-authz/rbac/#command-line-utilities

.

.
