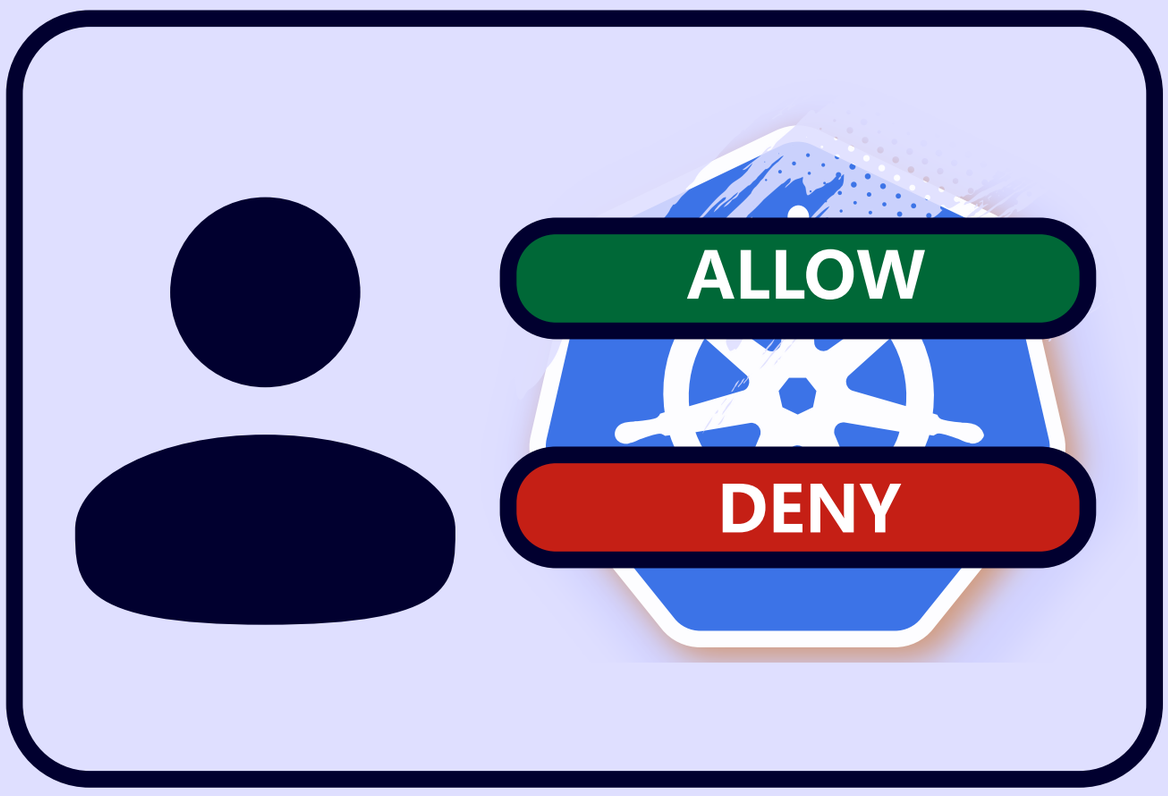
Authorization in Kubernetes

Authorization

- access management



Illustrated Handbook

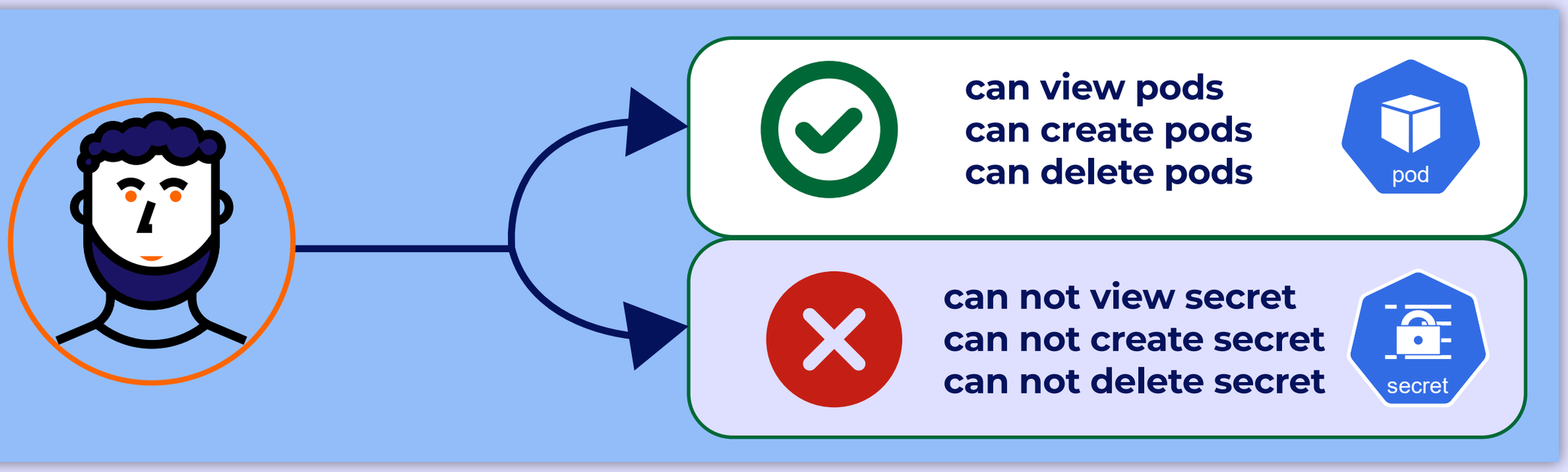
@deoluoyinlola

Content

1.0 What, Why & When Authorization

1.1 What is Authorization in Kubernetes?

Authorization, determines what the user **can and cannot do or access** in the system.



1.2 Why is Authorization in Kubernetes Necessary?

It is a security best practice, where access is giving to users or services just enough to perform their respective tasks in the system; **Least Privilege Rule.**

Admin Access

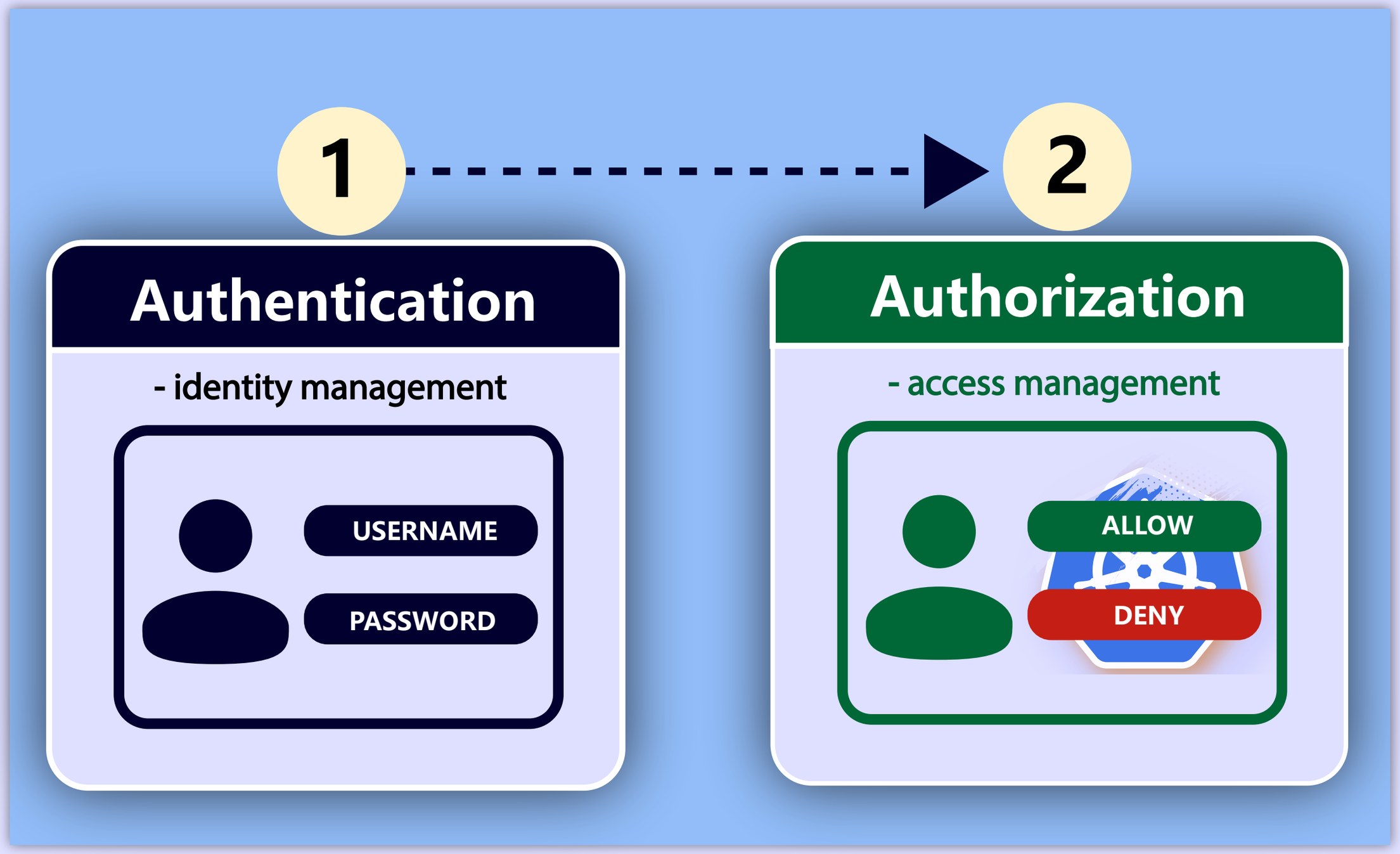
Cluster-wide access to administer the kubernete cluster.

Developer Access

Limited access to deploying, debugging applications in specified **namespace.**

1.3 When is Authorization in Kubernetes Needed?

Authentication is the first step and authorization must always follow authentication. Authorization is the next step when verifying if the user can perform the action they want to

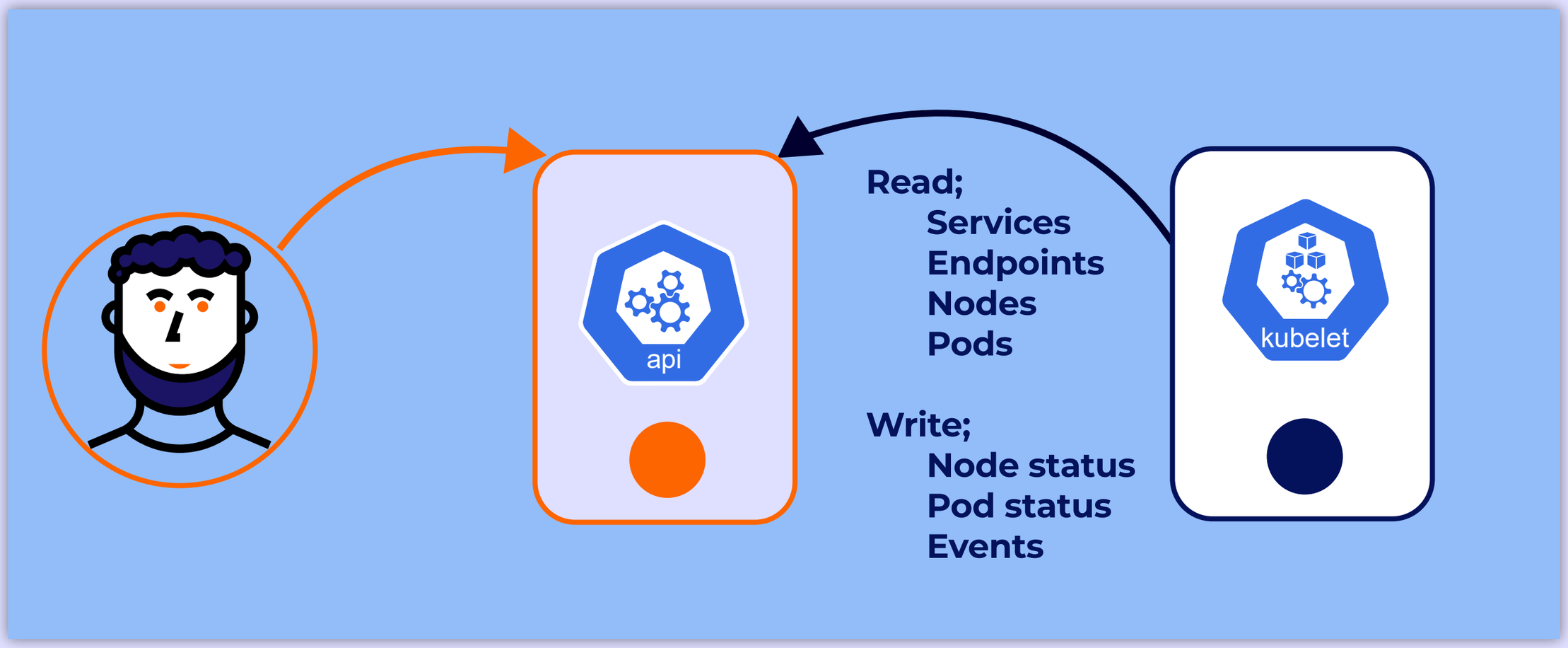


2.0 Kubernetes Authorization Modes

Kubernetes has a few authorization modes available that can be enabled by using the authorization-mode argument when starting the Kubernetes API server:

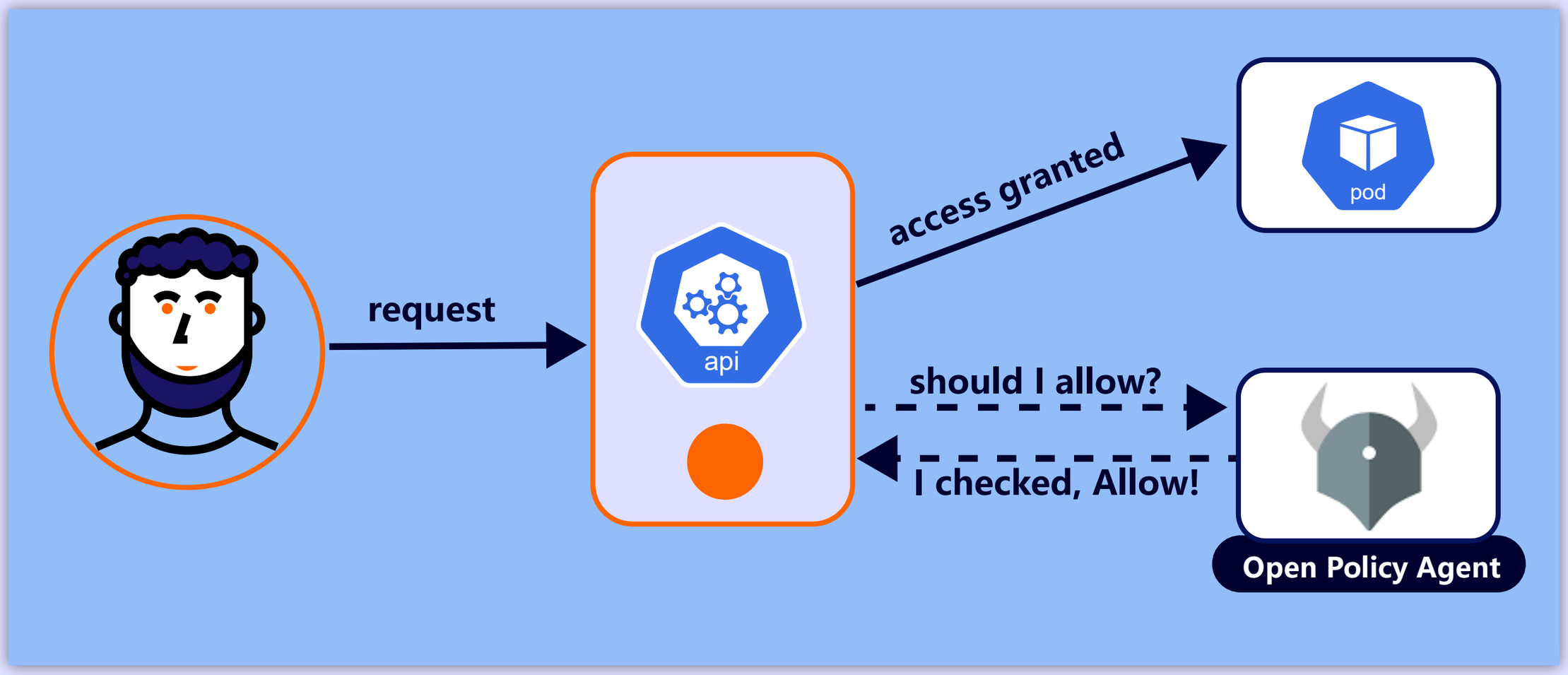
2.1 Node

A special-purpose authorization mode used for authorizing API requests made by kubelet in the cluster.



2.2 Webhook

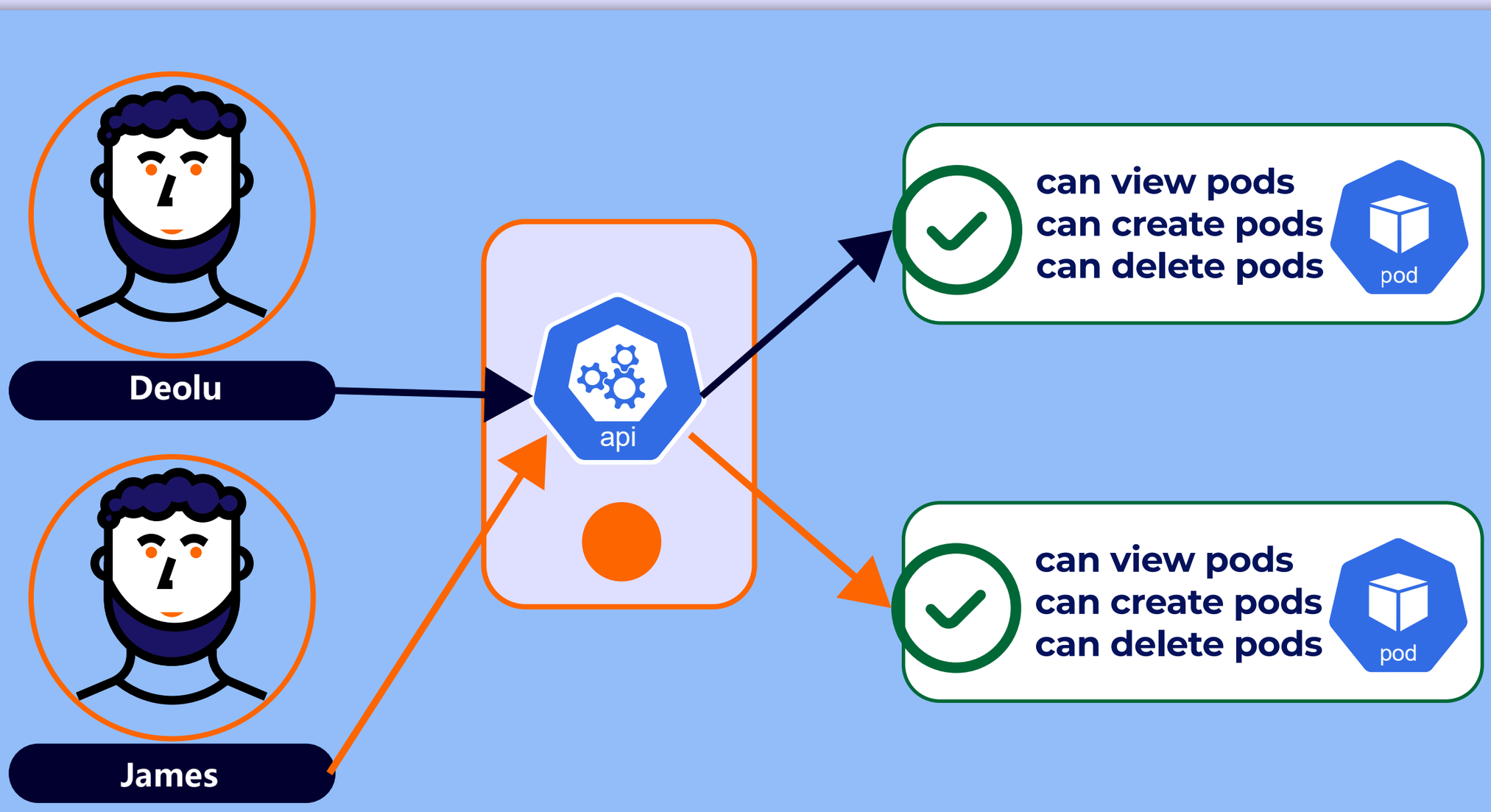
You can define an external service that needs to handle HTTP POST requests with an SubjectAccessReview object that's sent by the Kubernetes API server. This service must process the request and determine if the request should be allowed or denied. Based on that, the Kubernetes API server will either proceed with the request or reject it with an HTTP status code of 403.



2.0 Kubernetes Authorization Modes cont.

2.3 ABAC - Attribute Base Access Control

This uses policies based on the attributes of the user, resource, and environment. In Kubernetes, this is modeled using Policy objects. Policies are sets of attributes that must be present together for the action to be performed.
For example, you can define that the authenticated user, Deolu, can read any Pods in the default Namespace. If you want to give the same access to user James, then you need to create a new Policy for user James.




```
{"kind": "Policy", "spec": {"user": “Deolu", "namespace": "*", "resource": “pods", "apiGroup": "*"}}
{"kind": "Policy", "spec": {"user": “James", "namespace": "*", "resource": “pods", "apiGroup": "*"}}
```

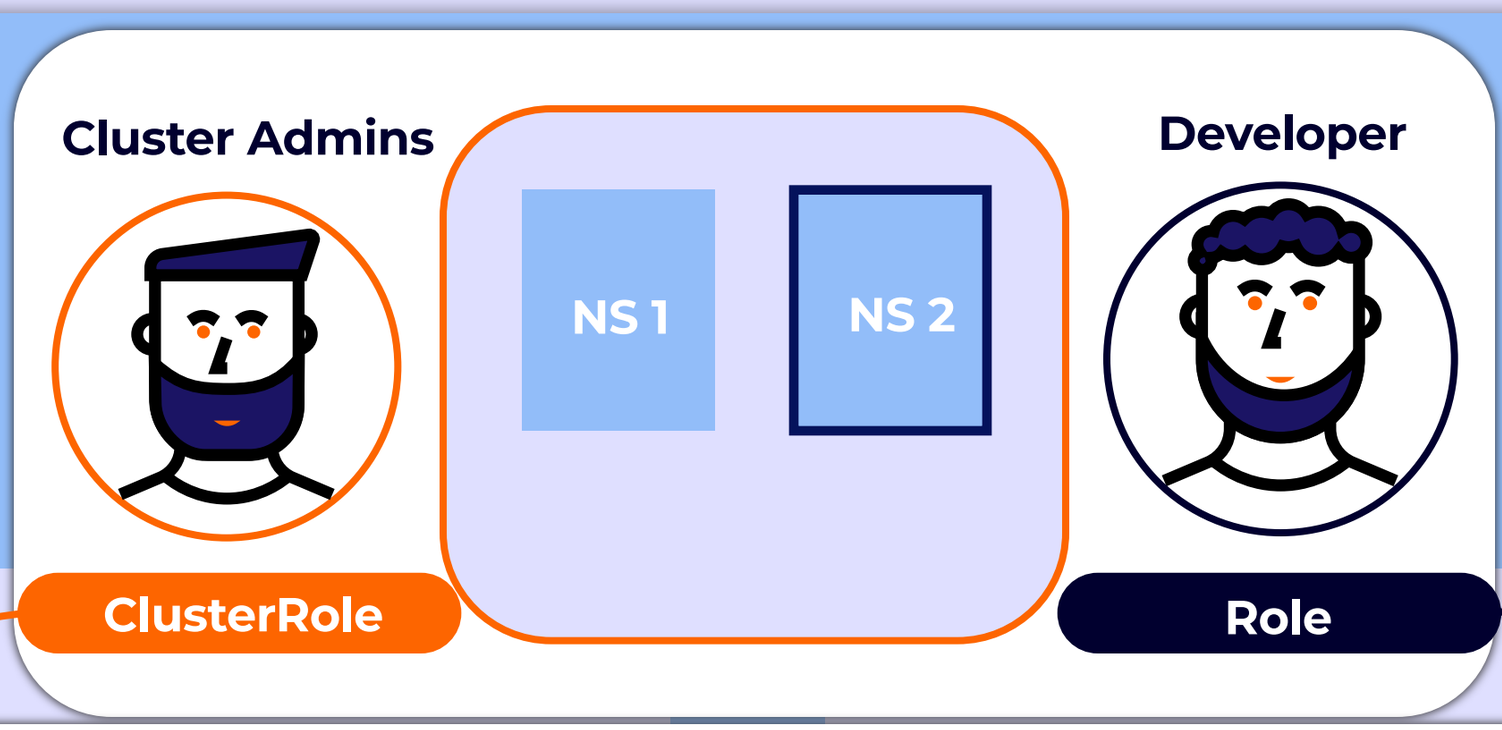
2.4 RBAC - Role Base Access Control

Roles can be assigned to users in the system, which gives them certain privileges and access.
This can be done by associating the user with a role – in Kubernetes, you model this using the RoleBinding and ClusterRoleBinding objects. In this way, multiple users can be assigned a role and a single user can have multiple roles assigned. Please note that in Kubernetes, RBAC is permissive, which means that there are no deny rules. Everything is denied by default, and you have to define allow rules instead.

2.0 Kubernetes Authorization Modes cont.

2.4 RBAC - Role Base Access Control cont.

Using RBAC in Kubernetes involves two types of API resources that belong to the rbac.authorization.k8s.io **API group: Role and ClusterRole & RoleBinding and ClusterRoleBinding**



Both define a set of permissions. Each rule in Role says which verb(s) are allowed for which API resource(s).

- define permissions cluster wide; global

cluster-admin-role.yaml

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

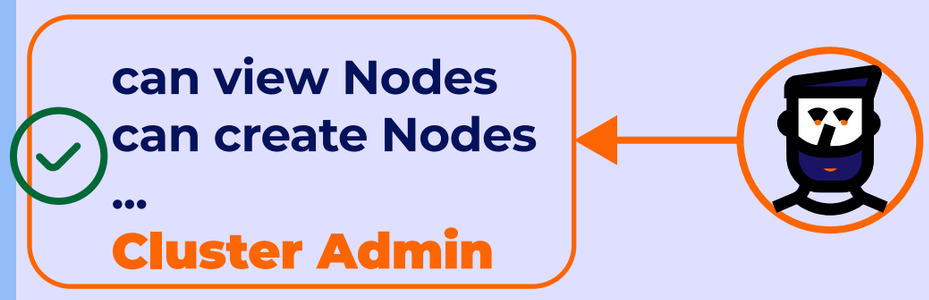
- define namespaced permissions through Role

developer-role.yaml

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

ClusterRoleBinding

They associate users or a set of users (alternatively, groups or ServiceAccounts) with a given Role. Link(”Bind”) to a user or Group



RoleBinding

3.0 Basic RBAC Implementation

3.1 RBAC demostration

We will now demonstrate this in practice by using RBAC in Kubernetes to be able to read the Pods in the cluster for a Pod running under ServiceAccount.
Please follow these steps to configure this example:

1. **Create a dedicated ServiceAccount named pod-logger by creating a YAML manifest named** pod-logger-serviceaccount.yaml**:**

pod-logger-serviceaccount.yaml

```
apiVersion: v1
 kind: ServiceAccount
metadata:
 name: pod-logger
 namespace: default
```

2. **Apply the manifest to the cluster using the following command:**

```
$ kubectl apply -f ./pod-logger-serviceaccount.yaml
```

3. **Create a role named pod-reader. This role will only allow the get, watch, and list verbs on pods resources in the Kubernetes RESTful API. The structure of the** pod-reader-role.yaml **YAML manifest file is as follows**

pod-reader-role.yaml

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