# End user Authentication with Auth0

## Prerequisities

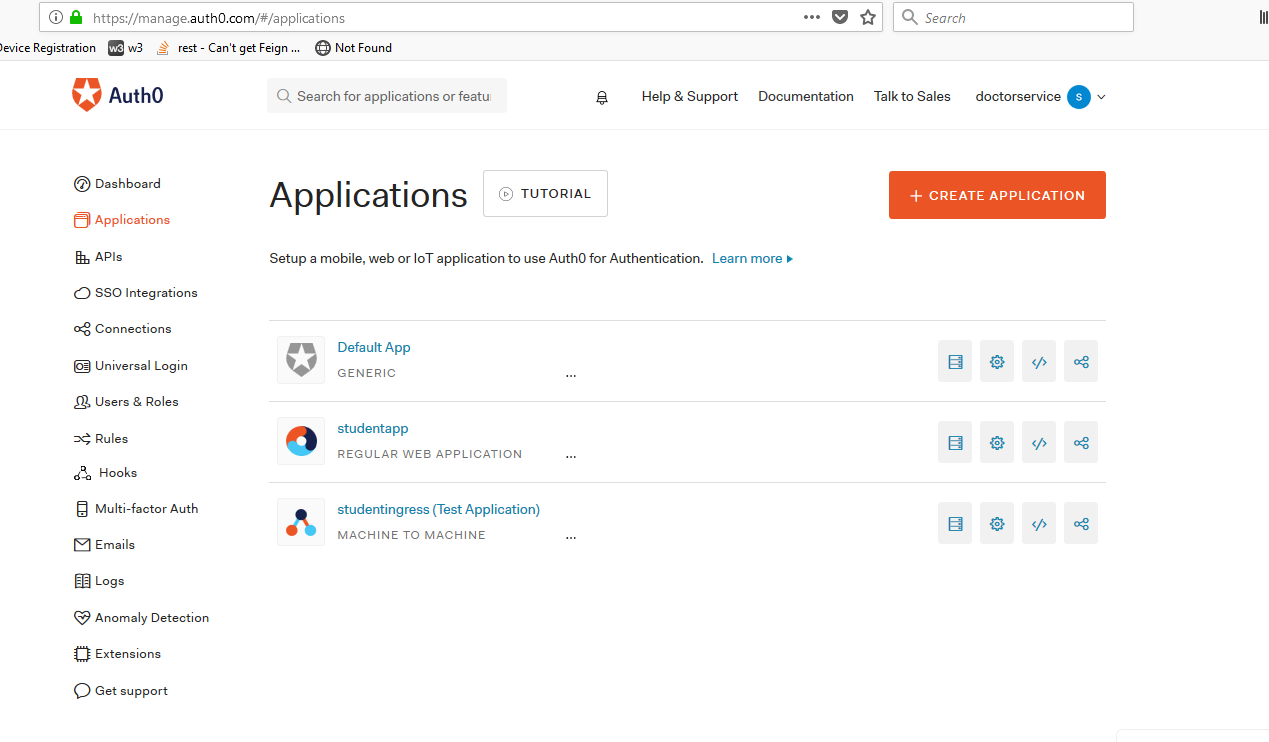
1. Have a running k8 cluster on one of the clouds (not on minikube)
2. An account with Auth0
3. Services deployed in K8 cluster
4. Istio installed on K8 cluster
5. Istio Ingress gateway ready to receive request from external world
6. Virtual services for all the services
7. Destination rules for all the services
8. Reference doc - <https://auth0.com/blog/securing-kubernetes-clusters-with-istio-and-auth0/>

## Implementation

## Setting up Auth0

1. Follow the reference doc to set up Auth0 authentication
2. That includes create an Application and an API

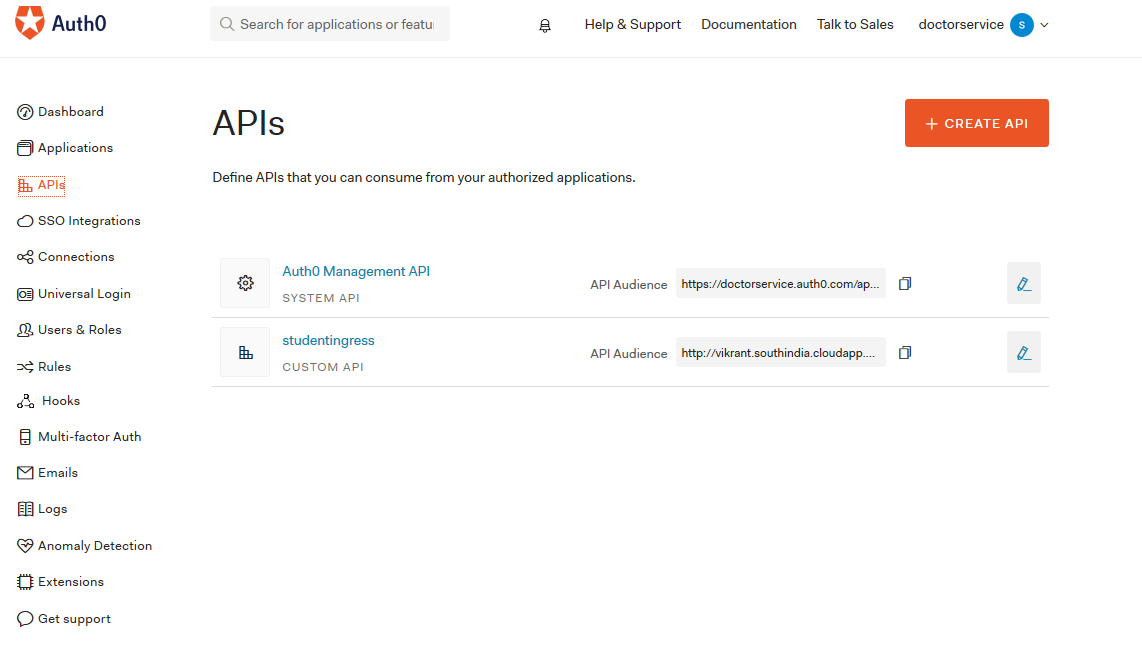
### Application



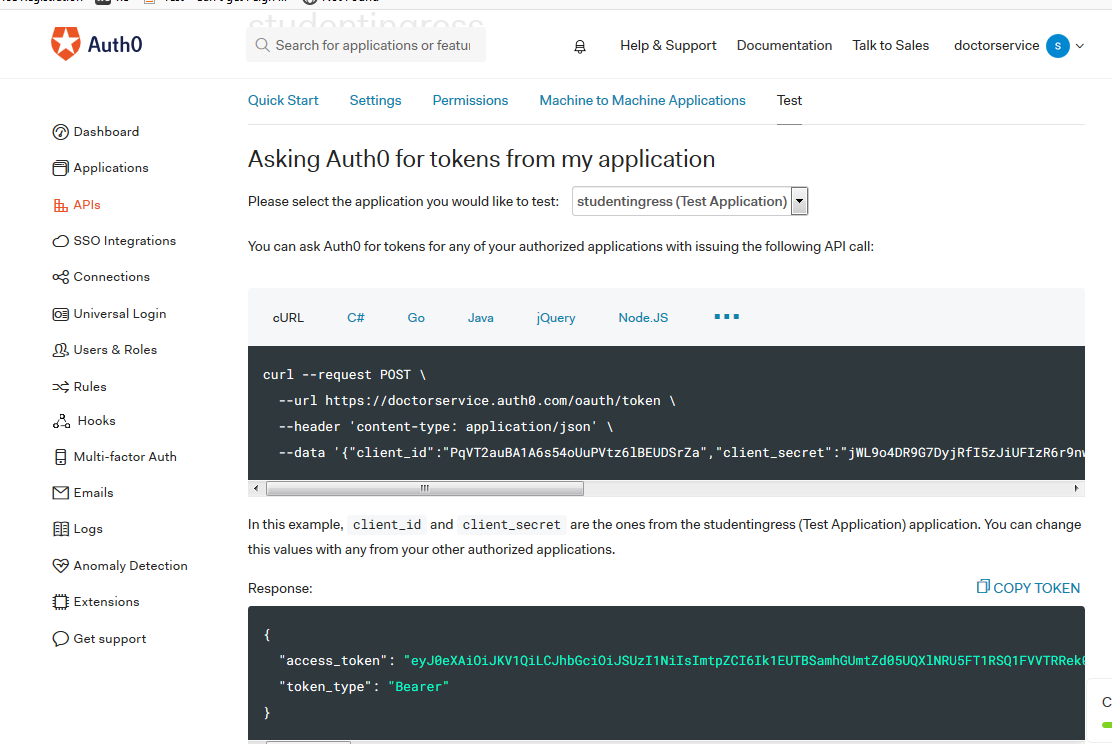
* I have created a studentApp application. Click on create Application to create your own app – should be a regular app – if you are using spring boot app, a different type of app if you are using some frontend tech.
* If you click on the application, it will show you the client id, client secret and domain
* My account name is doctorservice
* You will need to include callback url – should be – http://{Public ip/domain of your cluster}/callback
* Save your application

### API

* Create an API as well in Auth0



* I have create studentingress api.
* While creating your api, please provide a valid identifier name – should be – http://{Public ip/domain of your cluster}/
* Once created click on test, this will generate a sample test app, which you can see in the application section,



* The token that you see will be used to authenticate the endusers.

## Setting up Istio

1. Think – do you want to secure all the microservices within your service mesh ?

* If the answer is yes, MeshPolicy is our implementation
* Below is the code

#Mutual TLS authentication enabled on the entire mesh. This policy specifies that all workloads in the mesh will only accept encrypted request using tLS.  
#this authentication policy has the kind: MeshPolicy. the name of the policy must be default, and it contains no targets specification (as it is intended to aplly to all services in the mesh).  
  
apiVersion: "authentication.istio.io/v1alpha1"  
kind: "MeshPolicy"  
metadata:  
name: "default"  
spec:  
peers:  
- mtls: {}  
origins:  
- jwt:  
issuer: "[**https://doctorservice.auth0.com/"**](https://doctorservice.auth0.com/')  
jwksUri: "[**https://doctorservice.auth0.com/.well-known/jwks.json"**](https://doctorservice.auth0.com/.well-known/jwks.json')  
principalBinding: USE\_ORIGIN

* See that we have included origins and Jwt related info , if you apply this, and hit the endpoints of your microservice, you will see origin authentication failed

#### Problem with this implementation

* All the microservices are now authenticated with auth0 JWT
* What happens when a microservice has to talk to other microservice?
* If your code in the client microservice does not pass JWT token to the server microservice, you will get origin authentication failed from the other microservice , why? Because mesh wide policy is applied on all the services within the mesh
* We are still looking for a possible solution to this
* However you will have rare scenarios, where microservices are exposed to external world and hence need JWT authentication and they also communicate with each other , you will face the above mentioned problem.

**Possible solutions tried** – Created two services for the same Pod , Ingress gateway calls one service while during service-service communication , other service is invoked. This solution did not work because the policy is distributed to the sidecar

**Possible Solutions –** Create a wrapper which is the only entry point from external world and then routes requests based on url to different microservices, then the request will only be authenticated at the wrapper microservice and internal service-service communication would be just on mtls. But we can not use Meshwide policy to enforce authentication , else it will apply to all the microservices. We will need to create a separate policy which just applies to the wrapper microservice and includes policy enforcement of JWT authentication

**Possible workarounds :** If you are in dire need and this needs to work immediately for some client demo and you have time later to fix it, deploy two different apps of your microservice, two different services for those deployments, which means there will be 2 different sidecars, and then you apply policy to only one service. Not a prod solution though.

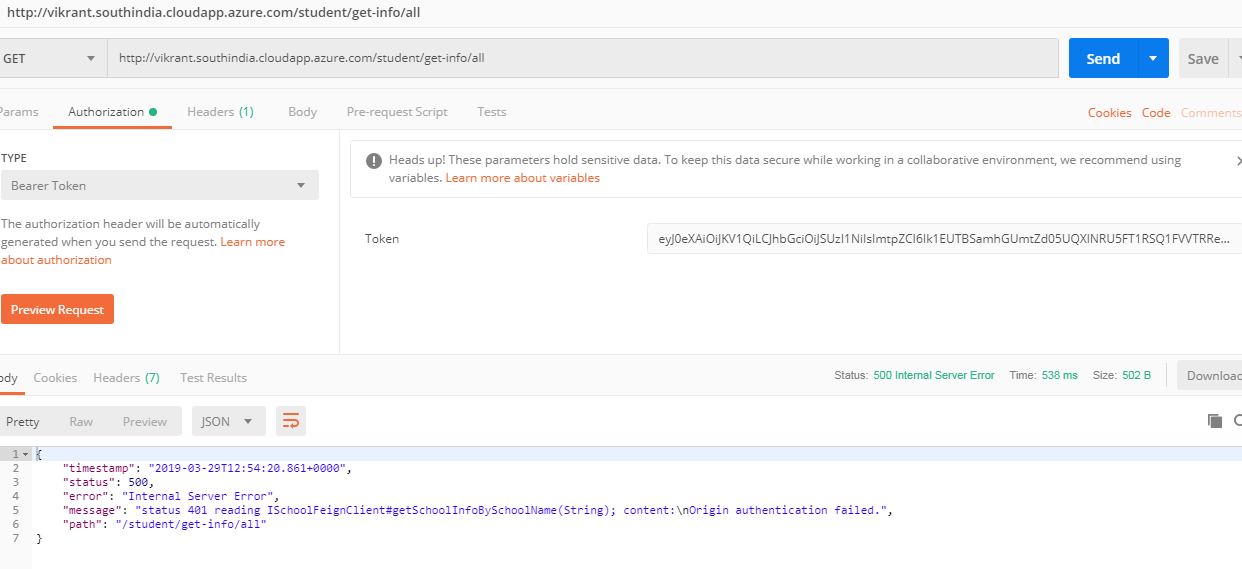
* We spoke about creating policy not at mesh level but at Service level
* Here is a sample

apiVersion: "authentication.istio.io/v1alpha1"  
kind: "Policy"  
metadata:  
name: "jwt-policy"  
namespace: "student"  
spec:  
peers:  
- mtls: {}  
targets:  
- name: studentservice  
- name: schoolservicev1   
origins:  
- jwt:  
issuer: "[**https://doctorservice.auth0.com/"**](https://doctorservice.auth0.com/')  
jwksUri: "[**https://doctorservice.auth0.com/.well-known/jwks.json"**](https://doctorservice.auth0.com/.well-known/jwks.json')  
principalBinding: USE\_ORIGIN

* Check out this policy this now applies only to two services not on the entire mesh.
* And you will need to change your Meshwide policy to just enforce service-service communication over mtls.

apiVersion: "authentication.istio.io/v1alpha1"  
kind: "MeshPolicy"  
metadata:  
name: "default"  
spec:  
peers:  
- mtls: {}

* If you apply these , your apps are authenticated using Auth0 and if you pass the bearer token in the request you should be able to see the response



* Remember to get the bearer token from Autho -> APIs-> Your api -> Test section
* You see we still get the origin authentication failure , because my policy applies auth0 to both the microservices that I have, and internally one ms calls other ms while the first ms gets authenticated because we have passed the bearer token , but when it makes call to other ms , it does not pass the bearer token and hence an origin auth error

## Authorization

1. Create service role in your namespace

apiVersion: "rbac.istio.io/v1alpha1"  
kind: ServiceRole  
metadata:  
name: studentadmin  
namespace: student  
spec:  
rules:  
- services: ["\*"]  
methods: ["\*"]

* I have created a service role called studentadmin – and it have access to all the services and all the methods within those services

1. Bind this role to a user at namespace level using servicerolebinding

apiVersion: "rbac.istio.io/v1alpha1"  
kind: ServiceRoleBinding  
metadata:  
name: student-admin  
namespace: student  
spec:  
subjects:  
- properties:  
source.namespace: "istio-system"  
- properties:  
source.namespace: "student"   
  
roleRef:  
kind: ServiceRole  
name: "studentadmin"

* The above config reads that the role student admin is applied to all the services within “istio-system” namespace as will as “student” namespace
* You can have more fine grained control by binding this role to a specific serviceaccount.
* ServiceAccounts are users in Kubernetes, there is no concept of end user in K8
* Each deployment in Kuberenetes that you create gets assigned a service account called default,if you do not specifically say which serviceaccount should this deployment be associated to
* Create a service account – kubectl create serviceaccount abc -n student
* Assign this service account to your deployment config under spec.spec section – spec.spec. serviceAccountName: abc



* Now bind this service account to the service role,

apiVersion: "rbac.istio.io/v1alpha1"  
kind: ServiceRoleBinding  
metadata:  
 name: student-admin  
 namespace: student  
spec:  
 subjects:  
 -user: “cluster.local/ns/student/sa/abc   
  
roleRef:  
kind: ServiceRole  
name: "studentadmin"

* Now service role admin only applies to the abc service account , which means the your app which got deployed with abc service account is assigned admin role

1. Create a rbac config, which will enable rbac on your entire mesh

#rbac - role backed access control  
#Metadata.name value should be default and this obj is mesh wide singleton

apiVersion: "rbac.istio.io/v1alpha1"

kind: RbacConfig

metadata:

name: default

spec:

mode: 'ON\_WITH\_INCLUSION'

inclusion:

namespaces: ["student"]

1. Authorization is enabled.