Kong Gateway Operations for Kubernetes

Securing Kong Gateway

Course Agenda

1. Kong Gateway Installation on K8s
2. Intro to Kong Ingress Controller
3. **Securing Kong Gateway**
4. Securing Services on Kong
5. OIDC Plugin
6. Kong Vitals
7. Advanced Plugins Review
8. Troubleshooting
9. Test your Knowledge

Learning Objectives

1. Describe the differences between Authentication (authn) and Authorization (authz)
2. Describe RBAC within the context of Kong
3. Understand how to secure Kong Manager and Kong Admin API
4. Configure Workspace permissions
5. Secure the Admin API
6. Identify some general Kong control plane hardening measures

Reset Environment

Before we start this lesson, be sure to reset your environment:

```
$ cd /home/labuser
$ git clone https://github.com/Kong/edu-kgac-202.git
$ source ./edu-kgac-202/base/reset-lab.sh
```

Security within Kong

Securing Kong

Kong admins with sufficient privileges can create/update/delete different Services, Routes, Plugins, Consumers and more, so you need to implement rigorous security on the control plane.

There are a number of aspects of your Kong Gateway you need to secure:

- Kong Manager
- Kong Admin API
- The Developer Portal

This security is based on the two basic principles of **Authentication** & **Authorization.**

Authentication & Authorization sound similar but they are very different security principles, and used for different purposes.

What are Authentication & Authorization?

Authentication means identifying users and validating who they claim to be.

Common authentication techniques include:

- Simple password
- One-time passcodes (OTP) via SMS/email
- Single Sign-On (SSO)
- Multi-factor authentication (MFA)
- Biometrics

Authorization means determining the level of access the user has on the system, once authenticated.

A common authorization technique is Role Based Access Control (RBAC).

We will look at Authentication within Kong shortly, but first we'll look closer at RBAC.

Authorization within Kong

Admins vs RBAC Users

There are two types of Kong administrative users:

- **Admins** - Users that have access to Kong Manager and the Admin API
- **RBAC Users** - Users that have programmatic access to Kong's Admin API

Please note that RBAC has not yet been enabled on our lab system.

Therefore we need to set the super users with Admin API and Kong Manager access first, so we can have access once we have enabled RBAC.

RBAC in Kong Manager & Admin API

An admin's level of authorization via Kong Admin API and Kong Manager is determined using Role-Based Access Control (RBAC).

The fundamentals of Kong's built-in RBAC system are:

- A user has a role
- A role has a collection of permissions
- An Admin belongs to a Workspace and has at least one Role

Since Kong is built entirely on top of APIs, the same account can be restricted in both the Kong Manager UI and the Admin API, and can be configured through either.

Out of the Box Roles

Kong ships with the following roles out of the box, each with a number of permissions

Role	Description
Workspace-admin	Full access to all endpoints in the workspace - except RBAC Admin API (e.g. can not create new admins)
Workspace-read-only	Read access to all endpoints in the workspace
Workspace-super-admin	Full access to all entities, including RBAC accounts (e.g. can create new admins)

What are permissions?

Each role may have a number of permissions that determine CRUD (Create, Read, Update, or Delete) access to resources in the API.

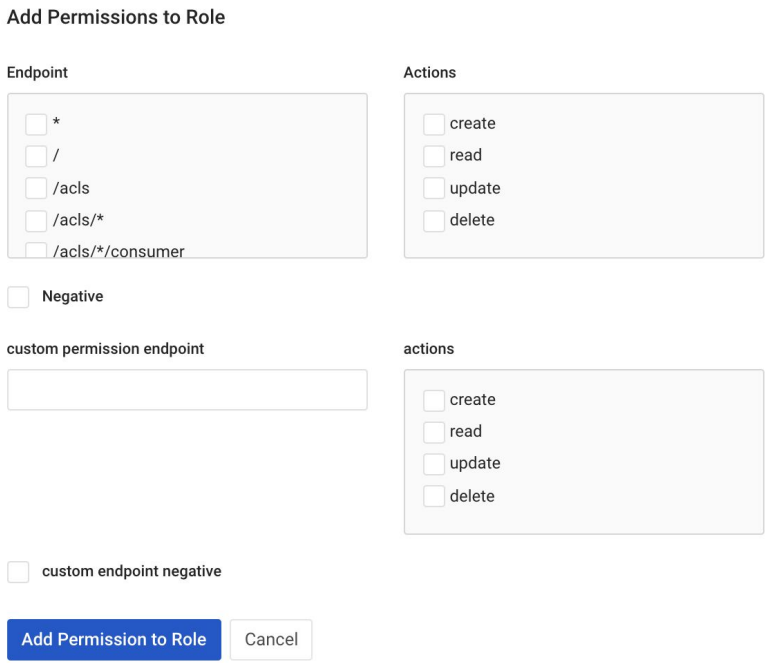
Kong's granular RBAC system allows permissions to be applied to sub resources under the main resource .

For example, you could have:

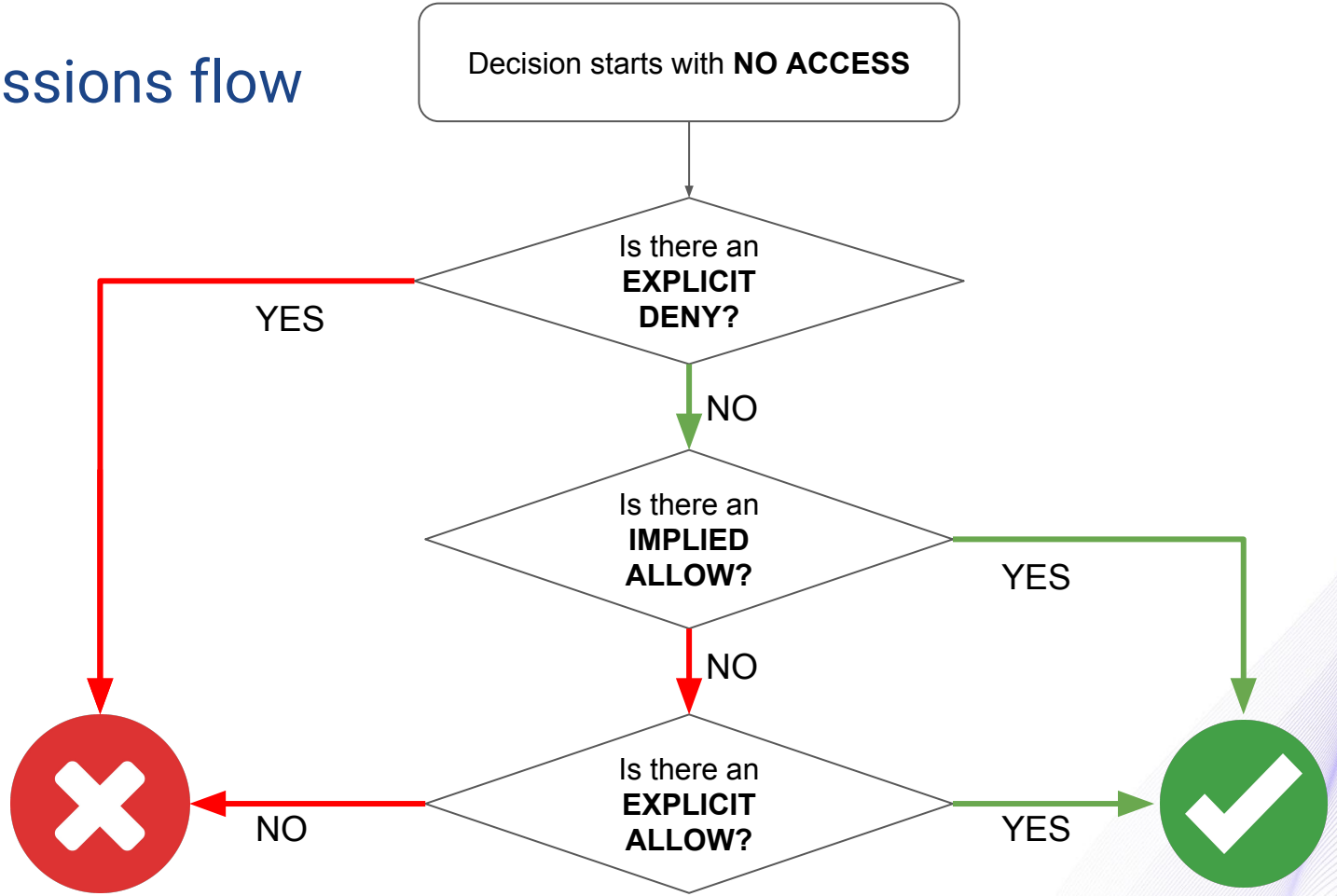
- `read` permission to `/foo/bar`, and
- `write` permission to `/foo/bar/far`

RBAC uses the principle of least privileged, whereby the smallest scoped role is available to particular personas.

Later we will see how to assign roles and permissions to users, but first let's create our own role called my_role.



Permissions flow



Lab: Configure RBAC User

In this learning lab you will:

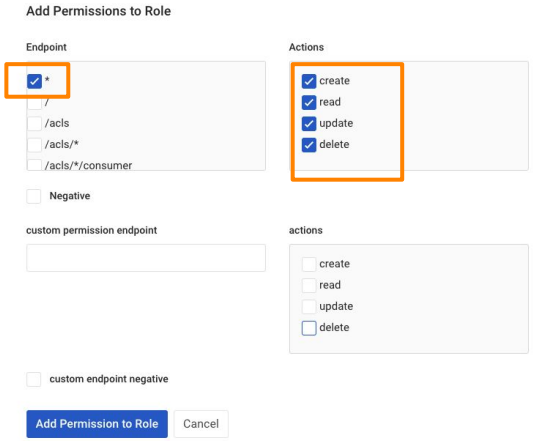
1. **Create an RBAC user called 'my-super-admin'**
2. Verify user and assign roles
3. Verify my-super-admin Role
4. Automatically Assign Roles to RBAC user

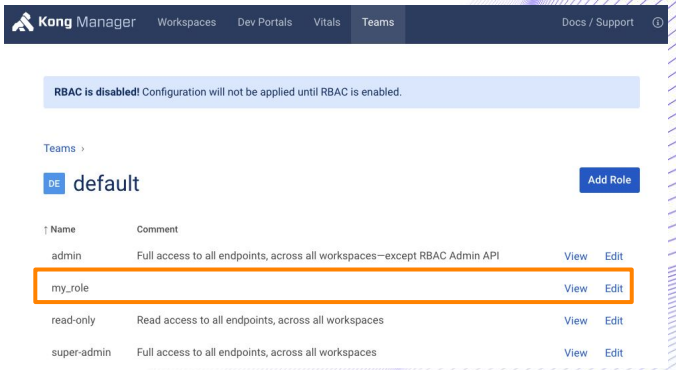
Task: Create New Role `my_role` and add Permissions

To create new roles and permissions

1. Open Kong Manager
2. Navigate to `Teams` > `Roles`
3. Click `View` next to `Default` workspace
   You will see existing Roles
4. Click `Add Role` and name it `my_role`
5. Click `Add Permission`
   Notice you can have very granular CRUD permissions for **any** endpoint
6. Select permissions as illustrated
7. Click `Add Permissions to Role`
8. Click `Create`

Note, you will see a message RBAC is disabled - we will enable it later.





Creating RBAC Users with Kong Admin API

An **rbac user** is added by posting to the '`/rbac/users`' endpoint on the Admin API

```
$ http post kongcluster:30001/rbac/users \
name=<admin-name> \
user_token="<rbac-token>"
```

RBAC user name must be globally unique

Kong uses **password authentication** to log in to Kong Manager, but uses an RBAC **user_token** to make requests to the Kong Admin API.

Task: Create an RBAC user called 'my-super-admin'

Lets create a new super-user called `my-super-admin` with token `my_token`

```
$ http post kongcluster:30001/rbac/users name=my-super-admin user_token="my_token"
```

```
HTTP/1.1 201 Created
Access-Control-Allow-Credentials: true
...
{
    "comment": null,
    "created_at": 1546940826000,
    "enabled": true,
    "id": "0bd27848-52de-445f-9213-76ecae19d68d",
    "name": "my-super-admin",
    "user_token": "RlpG8483KwDHPXZVL7lw5PhLBEnBJ0Bl",
    "User_token_ident": "93748"
}
```

Lab: Configure RBAC User

In this learning lab you will:

1. Create an RBAC user called 'my-super-admin'
2. **Verify user and assign roles**
3. Verify my-super-admin Role
4. Automatically Assign Roles to RBAC user

Task: Verify User and Assign Role

Even though the user now exists, they have no roles attached:

```
$ http get kongcluster:30001/rbac/users/my-super-admin/roles
```

```
{
    "roles": [],
    "user": {
        "comment": null,
...
}
```

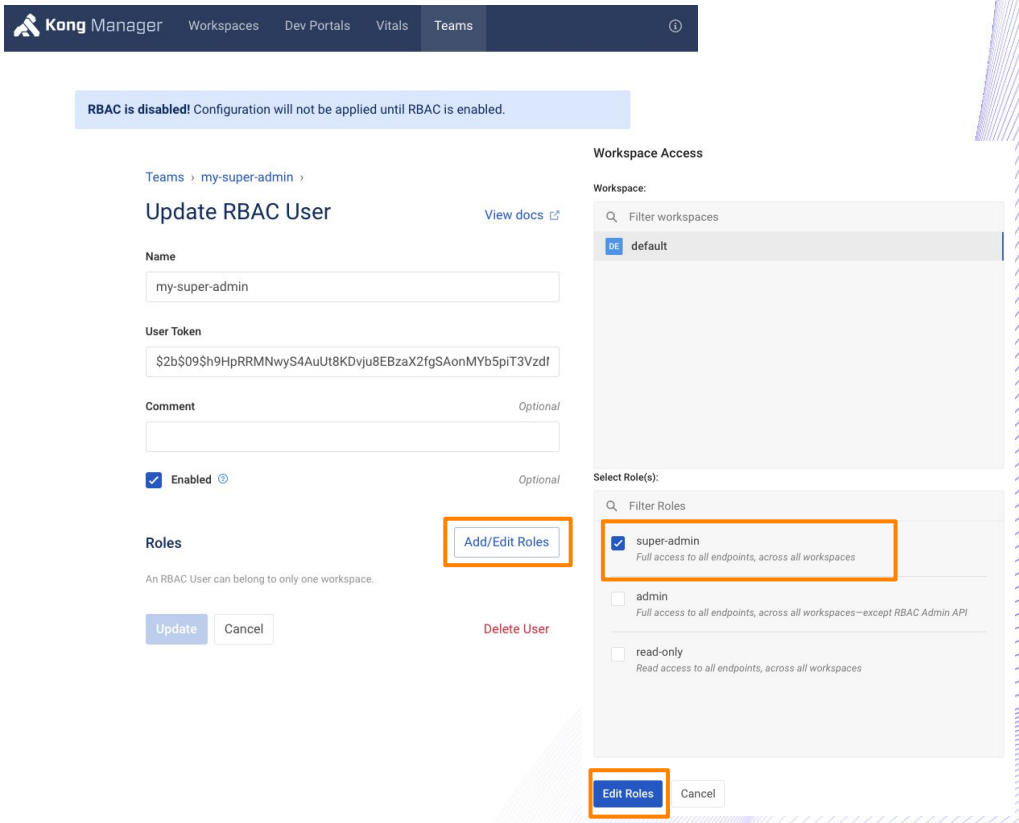
You can add the role via Kong Manager, but we will add it using Admin API

```
$ http post kongcluster:30001/rbac/users/my-super-admin/roles roles='my_role'
```

Task: Assign super-admin Role to my-super-admin

To assign super-admin role to `my-super-admin`:

1. Open Kong Manager
2. Navigate to `Teams` > `RBAC Users` > `my-super-admin` > `View` > `Edit User`
3. Add/edit roles
4. Assign roles '`super-admin`' & 'my_role'
5. Edit Roles > Update



Lab: Configure RBAC User

In this learning lab you will:

1. Create an RBAC user called 'my-super-admin'
2. Verify user and assign roles
3. **Verify my-super-admin Role**
4. Automatically Assign Roles to RBAC user

Task: Verify my-super-admin Role

Notice the super-admin user you created now has the `super-admin` role

```
$ http get kongcluster:30001/rbac/users/my-super-admin/roles
```


```
{
    "roles": [
        {
            "created_at": 1629840150,
            "id": "19088a8c-a0a1-475c-9632-7d0ca74d8aa7",
            "name": "my_role",
            "ws_id": "5a32cf60-2ea4-42f8-b60f-f159debc232b"
        },
        {
            "comment": "Full access to all endpoints, across all workspaces",
            "created_at": 1629816578,
            "id": "c354cac8-f5c9-4e9c-a1d9-5b5be11f16f5",
            "name": "super-admin",
            "ws_id": "5a32cf60-2ea4-42f8-b60f-f159debc232b"
        }
...
```

Lab: Configure RBAC User

In this learning lab you will:

1. Create an RBAC user called 'my-super-admin'
2. Verify user and assign roles
3. Verify my-super-admin Role
4. **Automatically Assign Roles to RBAC user**

Task: Automatically Assign Roles to RBAC user

It's worth noting that RBAC users that have the same name as a **predefined role** are automatically given that role

To illustrate, create a new user with name super-admin

```
$ http post kongcluster:30001/rbac/users \
    name=super-admin \
    user_token="super-admin"
```

```
HTTP/1.1 201 Created
```

If you verify this user you'll see they automatically have the super-admin role

```
$ http get kongcluster:30001/rbac/users/super-admin/roles
```

Lab: Configure RBAC User

In this lab we:

- ✓ Created an RBAC user called 'my-super-admin'
- ✓ Verified user and assign roles
- ✓ Verified my-super-admin Role
- ✓ Automatically Assigned Roles to RBAC user

Authentication within Kong

Enabling Authentication

Up to this point we have created permissions, roles, and RBAC users, and have assigned the roles to the user, however to test these we must enable RBAC on our Kong Gateway.

So now we can enable Authentication on Kong Gateway and test our credentials.

Kong supports three methods of authentication for the control plane:

- Basic Authentication
- OIDC
- LDAP

In this lesson we will use Basic Authentication.

Configuring Authentication

Generally, authentication is enabled by setting the following values in the configuration file `/etc/kong/kong.conf`

```
enforce_rbac = on
admin_gui_auth = basic-auth <or oidc or ldap>
admin_gui_session_conf = { "secret":"set-your-string-here" }
```

Refer to the Kong docs more optional configuration settings, e.g.

- `admin_gui_auth_password_complexity`
- `admin_gui_auth_header`
- etc

However, since we are using Helm, we will enable it via our values yaml.

Sessions and Cookies

- When RBAC is enabled, you must configure session cookies.
- Session cookies are used to authenticate client requests issued during a particular session.
- The session has a limited duration and renews at a configurable interval.

Let's configure Basic Authentication, including this session cookie, on our Kong Gateway.

Lab: Enable and Verify RBAC

In this learning lab you will:

1. Enable RBAC, reducing default cookie_lifetime
2. Verify Authentication with Kong Manager
3. Verify Authentication with Admin API

Task: Enable RBAC, reducing default cookie_lifetime

Run the following command to create a new admin_gui_session_conf file with a reduced session timeout so we can see it expire. Apply the settings to the existing secret.

```
$ cd /home/labuser/edu-kgac-202
$ cat << EOF > admin_gui_session_conf
{
    "cookie_name":"admin_session",
    "cookie_samesite":"off",
    "secret":"kong",
    "cookie_secure":true,
    "cookie_lifetime":60,
    "storage":"kong"
}
EOF
$ kubectl create secret generic kong-session-config -n kong \
--save-config \
--dry-run=client \
--from-file=admin_gui_session_conf \
-o yaml | \
kubectl apply -f -
$
```

Task: Enable RBAC, reducing default cookie_lifetime

Use sed to inject the correct values of this lab so that the CORS will be set correctly by helm and make the gateway aware of its endpoints.

```
$ yq -i '.env.admin_gui_url = env(KONG_MANAGER_URL)'
./exercises/rbac/cp-values-rbac.yaml
$ yq -i '.env.admin_api_url = env(KONG_ADMIN_API_URL)'
./exercises/rbac/cp-values-rbac.yaml
$ yq -i '.env.admin_api_uri = env(KONG_ADMIN_API_URI)'
./exercises/rbac/cp-values-rbac.yaml
$ yq -i '.env.proxy_url = env(KONG_PROXY_URL)' ./exercises/rbac/cp-values-rbac.yaml
$ yq -i '.env.portal_api_url = env(KONG_PORTAL_API_URL)'
./exercises/rbac/cp-values-rbac.yaml
$ yq -i '.env.portal_gui_host = env(KONG_PORTAL_GUI_HOST)'
./exercises/rbac/cp-values-rbac.yaml
$
```

Task: Enable RBAC, reducing default cookie_lifetime

Run the following command to apply the upgrade with the new cp-values-rbac.yaml Helm values file.

```
$ helm upgrade -f ./exercises/rbac/cp-values-rbac.yaml kong kong/kong -n kong \
--set manager.ingress.hostname=$KONG_MANAGER_URI \
--set portal.ingress.hostname=$KONG_PORTAL_GUI_HOST \
--set admin.ingress.hostname=$KONG_ADMIN_API_URI \
--set portalapi.ingress.hostname=$KONG_PORTAL_API_URI
$ watch "kubectl get pods -n kong"
```

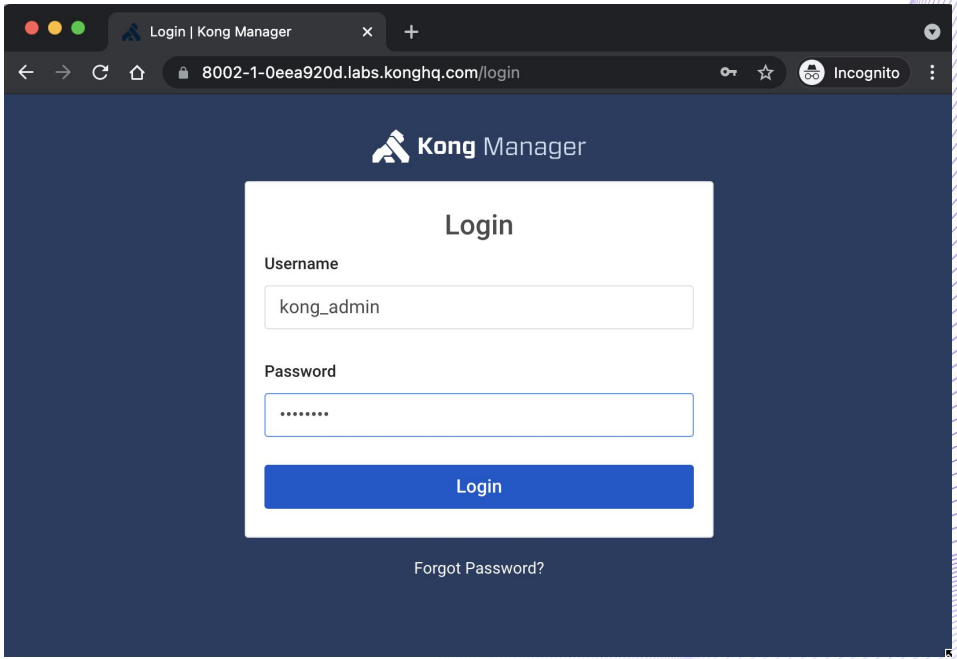
Press CTRL-C to quit watching the pods

Task: Verify Authentication with Kong Manager

Refresh Kong Manager - you will now be prompted for a username and password - you can login with `kong_admin/password`

After `cookie_lifetime` seconds the cookie will expire and you will need to reauthenticate.

Next, we will revert the cookie_lifetime.



Task: Revert the cookie_lifetime to 36000 seconds

Run the following command to modify the admin_gui_session_conf file and apply the new settings to the existing secret.

```
$ sed -i "s/\"cookie_lifetime\":60/\"cookie_lifetime\":36000/g" admin_gui_session_conf
$ kubectl create secret generic kong-session-config -n kong \
--save-config \
--dry-run=client \
--from-file=admin_gui_session_conf \
-o yaml | \
kubectl apply -f -
secret/kong-session-config configured
$ DELETE_POD=`kubectl get pods --selector=app=kong-kong -n kong -o
jsonpath='{.items[*].metadata.name}'`
$ kubectl delete pod $DELETE_POD -n kong
$ watch "kubectl get pods -n kong"
$
```

Press CTRL-C to quit watching the pods

Task: Verify Authentication with Admin API

We now must pass a token when access thing the Admin API

```
$ http --headers get kongcluster:30001/services
```

```
HTTP/1.1 401 Unauthorized
```

You can pass the token `my_token` created earlier in the `Kong-Admin-Token` HTTP Header as follows

```
$ http --headers get kongcluster:30001/services Kong-Admin-Token:my_token
```

```
HTTP/1.1 200 OK
```

Workspaces and Teams

What are Workspaces and Teams?

Workspaces enable traffic segmentation and allows teams of Admins to share the same Kong cluster, but only interact with entities from their groups.

For example, **TeamA** may be responsible for managing **ServiceA**, whereas **TeamB** may be responsible for managing **ServiceB**, and each one only have the required roles to perform the administrative tasks within their corresponding workspaces.

Delegated Administration of the Gateway can be achieved by adding administrators to 'Teams', then assigning these Teams to 'workspaces'.

We will illustrate workspaces and admins by way of a lab.

Lab: Create Workspaces

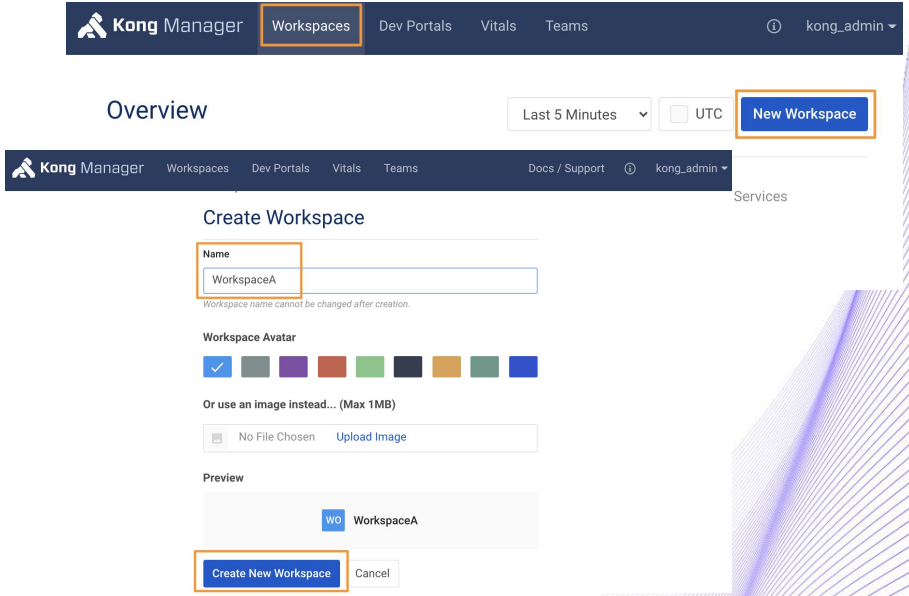
In order to help understand workspaces, we will perform the following tasks

1. Create and verify WorkspaceA and WorkspaceB
2. Create and verify AdminA and AdminB
3. Create an `admin` role with permissions for `WorkspaceA` and `WorkspaceB`
4. Assign `admin` role to admin user in each workspace
5. Verify `AdminA` has access to `WorkspaceA`, but not `WorkspaceB`
6. Use `AdminB` to add service to `WorkspaceA` - **FAILURE**
7. Use `AdminA` to add service to `WorkspaceA` - **SUCCESS**

Task: Create & verify WorkspaceA & WorkspaceB

Let's create WorkspaceA & WorkspaceB

1. Log into Kong Manager
2. Navigate to `Workspaces` > `New Workspace`
3. Name it `WorkspaceA`
4. Assign a color if you wish, and click `Create New Workspace`
5. Repeat steps 1-4 for `WorkspaceB`



If you wish you can verify them as follows

```
$ http get kongcluster:30001/workspaces Kong-Admin-Token:my_token | jq '.data[].name'
```

```
"WorkspaceA"
"WorkspaceB"
"default"
```

Naming Workspaces

Note: Do not give a Workspace the same name as any of these paths in Admin API:

- APIs
- Admins
- Certificates
- Consumers
- Plugins
- Portal
- Routes
- SNIs
- Services
- Upstreams
- Vitals

Task: Create AdminA & AdminB

Now create AdminA & AdminB. Notice that we are using `super-admin` role to create admins in each workspace. Note also the Workspace name is specified in the request

```
$ http post kongcluster:30001/WorkspaceA/rbac/users \
    name=AdminA \
    user_token=AdminA_token \
    Kong-Admin-Token:super-admin
```

```
HTTP/1.1 201 Created
```

```
$ http post kongcluster:30001/WorkspaceB/rbac/users \
    name=AdminB \
    user_token=AdminB_token \
    Kong-Admin-Token:super-admin
```

```
HTTP/1.1 201 Created
```

Task: Verify AdminA & AdminB

Now verify AdminA is assigned to WorkspaceA. You can do the same for AdminB & WorkspaceB

```
$ http get kongcluster:30001/WorkspaceA/rbac/users Kong-Admin-Token:super-admin
```

```
...
        {
            "comment": null,
            "created_at": 1629969213,
            "enabled": true,
            "id": "c2a95c3a-2e08-4100-9101-f41eb514e4c7",
            "name": "AdminA",
            "user_token": "$2b$09$.FDD6jUM1rUTV7jQvFVgHO8u7tXkaYQbesbMlhz8pml6FdsE9dS46",
            "user_token_ident": "500ac"
        }
...
```

```
$ http get kongcluster:30001/WorkspaceB/rbac/users Kong-Admin-Token:super-admin
```

Task: Create an admin role & permissions for WorkspaceA

We now need to create an admin role that grants permissions to admin users

```
$ http post kongcluster:30001/WorkspaceA/rbac/roles name=admin Kong-Admin-Token:super-admin
```

```
{
    "comment": null,
    "created_at": 1629970196,
    "id": "3703d0a4-2191-4133-8fbb-2487bf23d81f",
    "is_default": false,
    "name": "admin"
}
```

To create the actual permissions

```
$ http post kongcluster:30001/WorkspaceA/rbac/roles/admin/endpoints/ \
     endpoint=* \
     workspace=WorkspaceA \
     actions=* \
     Kong-Admin-Token:super-admin
```

```
HTTP/1.1 201 Created
```

Task: Repeat admin role & permissions for WorkspaceB

```
$ http post kongcluster:30001/WorkspaceB/rbac/roles name=admin Kong-Admin-Token:super-admin
```

```
HTTP/1.1 201 Created
```

```
$ http post kongcluster:30001/WorkspaceB/rbac/roles/admin/endpoints/ \
     endpoint=* \
     workspace=WorkspaceB \
     actions=* \
     Kong-Admin-Token:super-admin
```

```
HTTP/1.1 201 Created
```

Task: Assign admin role to admin user in each workspace

The next step is to add user AdminA to the admin role in WorkspaceA, and AdminB to the admin role in WorkspaceB

```
$ http post kongcluster:30001/WorkspaceA/rbac/users/AdminA/roles/ \
    roles=admin \
    Kong-Admin-Token:super-admin
```

```
HTTP/1.1 201 Created
```

```
$ http post kongcluster:30001/WorkspaceB/rbac/users/AdminB/roles/ \
    roles=admin \
    Kong-Admin-Token:super-admin
```

```
HTTP/1.1 201 Created
```

Task: Verify AdminA/AdminB access to corresponding Workspaces

Now AdminA should have access to WorkspaceA, but not WorkspaceB and vice versa

```
$ http get kongcluster:30001/WorkspaceA/rbac/users Kong-Admin-Token:AdminA_token
```

```
HTTP/1.1 200 OK
```

```
$ http get kongcluster:30001/WorkspaceB/rbac/users Kong-Admin-Token:AdminA_token
```

```
HTTP/1.1 403 Forbidden
```

and vice versa

```
$ http get kongcluster:30001/WorkspaceB/rbac/users Kong-Admin-Token:AdminB_token
```

```
$ http get kongcluster:30001/WorkspaceA/rbac/users Kong-Admin-Token:AdminB_token
```

Task: Deploy a service to WorkspaceA with correct Admin

Now let's deploy a service to WorkspaceA using AdminB (fail!) then AdminA (win!)

```
$ http post kongcluster:30001/WorkspaceA/services name=httpbin_service \
    url=http://httpbin:80 Kong-Admin-Token:AdminB_token
```

```
HTTP/1.1 403 Forbidden
```

```
$ http post kongcluster:30001/WorkspaceA/services name=httpbin_service \
    url=http://httpbin:80 Kong-Admin-Token:AdminA_token
```

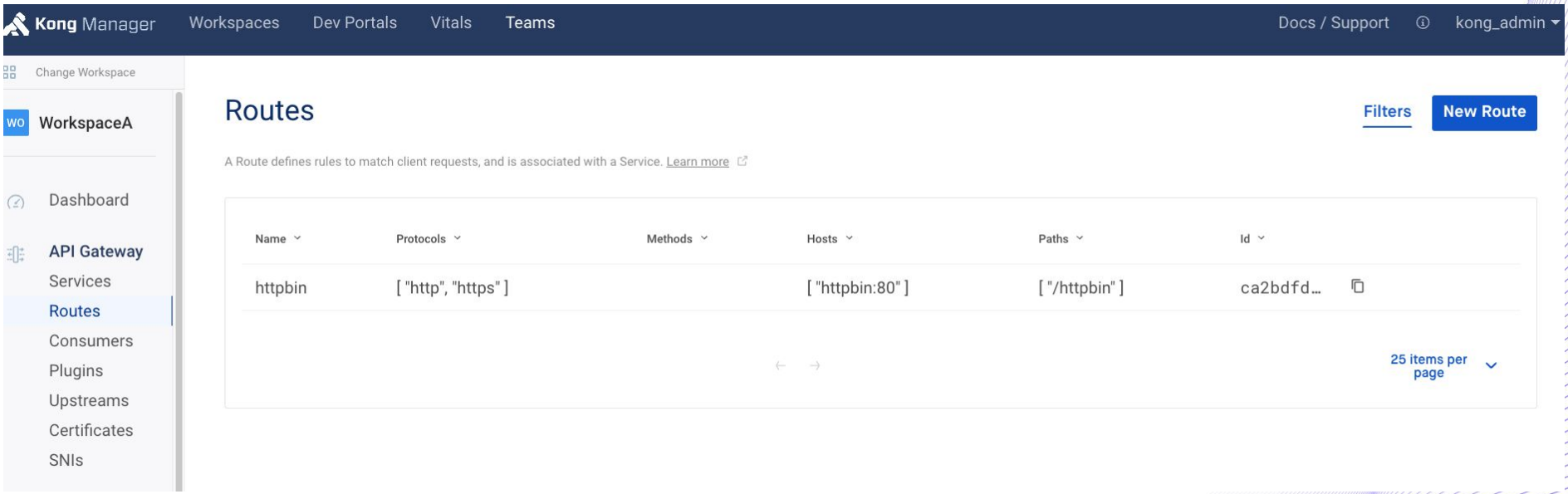
```
HTTP/1.1 201 Created
```

```
$ http -f post kongcluster:30001/WorkspaceA/services/httpbin_service/routes name=httpbin \
    hosts=httpbin:80 paths=/httpbin Kong-Admin-Token:AdminA_token
```

```
HTTP/1.1 201 Created
```

Task: Verify service in WorkspaceA

Let's verify services in new workspace in Kong Manager



Lab: Create Workspaces and Admins

In order to help understand workspaces, we will perform the following tasks

- ✓ Create & verify WorkspaceA & WorkspaceB
- ✓ Create & verify AdminA & AdminB
- ✓ Create an `admin` role & permissions for `WorkspaceA` & `WorkspaceB`
- ✓ Assign `admin` role to admin user in each workspace
- ✓ Verify `AdminA` has access to `WorkspaceA`, but not `WorkspaceB`
- ✓ Use `AdminB` to Add service to `WorkspaceA` - FAIL!!
- ✓ Use `AdminA` to Add service to `WorkspaceA` - WIN!!

Working with Teams

The concept of workspaces not only relates to admins, like we have just seen, but also for teams of users with lesser permissions.

To illustrate this we will create engineers with read only for the workspace they are in, and no access to the other workspace at all.

So the tasks are

1. Add `TeamA_engineer` & `TeamB_engineer` to the workspace teams
2. Create read-only roles and permissions for 'Team_engineers'
3. Assign roles and permissions for 'Team_engineers'
4. Test access for the engineers for their workspace and ensure that TeamA_engineer
    a. should be able to read from WorkspaceA
    b. cannot create in WorkspaceA
    c. should have no access to WorkspaceB

Task: Add TeamA_engineer & TeamB_engineer to the workspace teams

By their nature Admins have privileged access, but RBAC allows multiple levels of access.

So let's use the correct Admin to create a user 'Team_engineer' with read only access for their workspace

```
$ http post kongcluster:30001/WorkspaceA/rbac/users name=TeamA_engineer
user_token=teama_engineer_user_token Kong-Admin-Token:AdminB_token
```

```
HTTP/1.1 403 Forbidden
```

```
$ http post kongcluster:30001/WorkspaceA/rbac/users name=TeamA_engineer
user_token=teama_engineer_user_token Kong-Admin-Token:AdminA_token
```

```
HTTP/1.1 201 Created
```

```
$ http post kongcluster:30001/WorkspaceB/rbac/users name=TeamB_engineer
user_token=teamb_engineer_user_token Kong-Admin-Token:AdminB_token
```

Task: Create read-only roles and permissions for 'Team_engineer'

Similar to what we done before for the admins, we'll now create a specific '`engineer-role`' role, and assign only read-only permissions to bother workspaces

```
$ http post kongcluster:30001/WorkspaceA/rbac/roles name=engineer-role
Kong-Admin-Token:super-admin
```

```
HTTP/1.1 201 Created
```

```
$ http post kongcluster:30001/WorkspaceA/rbac/roles/engineer-role/endpoints/ \
      endpoint=* \
      workspace=WorkspaceA \
      actions="read" \
      Kong-Admin-Token:AdminA_token
```

```
HTTP/1.1 201 Created
```

Repeat for WorkspaceB and AdminB.

Task: Assign roles and permissions for 'Team_engineer'

Now that we have created the roles and permission, we'll now assign these to the 'Team_engineer' user - this is similar to what we did for Admins earlier

```
$ http post kongcluster:30001/WorkspaceA/rbac/users/TeamA_engineer/roles \
    roles=engineer-role \
    Kong-Admin-Token:AdminA_token
```

```
HTTP/1.1 201 Created
```

And for the 'B' cohort

```
$ http post kongcluster:30001/WorkspaceB/rbac/users/TeamB_engineer/roles \
    roles=engineer-role \
    Kong-Admin-Token:AdminB_token
```

```
HTTP/1.1 201 Created
```

Verifying Teams Access

Now if we have set things up correctly, TeamA_engineer

- should be able to read from WorkspaceA
- and cannot create in WorkspaceA
- and should have no access to WorkspaceB

Similarly for TeamB_engineer

Task: Test read only access for the engineers for their workspace only

Now if we have set things up correctly, TeamA_engineer should be able to list consumers from WorkspaceA but cannot create them

```
$ http get kongcluster:30001/WorkspaceA/consumers Kong-Admin-Token:teama_engineer_user_token
```

```
HTTP/1.1 200 OK
```

```
$ http post kongcluster:30001/WorkspaceA/consumers username=Jane
Kong-Admin-Token:teama_engineer_user_token
```

```
    "message": "TeamA_engineer, you do not have permissions to create this resource"
```

TeamA_engineer should have no read access to WorkspaceB

```
$ http get kongcluster:30001/WorkspaceB/consumers Kong-Admin-Token:teama_engineer_user_token
```

```
    "message": "TeamA_engineer, you do not have permissions to create this resource"
```

Disable RBAC

Task: Disable RBAC

```
$ cat << EOF > admin_gui_session_conf
{
    "cookie_name":"admin_session",
    "cookie_samesite":"off",
    "secret":"kong",
    "cookie_secure":false,
    "storage":"kong"
}
EOF
$ kubectl create secret generic kong-session-config -n kong \
--save-config \
--dry-run=client \
--from-file=admin_gui_session_conf \
-o yaml | \
kubectl apply -f -
$ helm upgrade -f ./base/cp-values.yaml kong kong/kong -n kong \
--set manager.ingress.hostname=$KONG_MANAGER_URI \
--set portal.ingress.hostname=$KONG_PORTAL_GUI_HOST \
--set admin.ingress.hostname=$KONG_ADMIN_API_URI \
--set portalapi.ingress.hostname=$KONG_PORTAL_API_URI
$ watch "kubectl get pods -n kong"
```

Secure the Admin API

Securing the Admin API

It is important to secure Kong's Admin API against unwanted access as it allows full control of Kong Gateway.

There are a several possible approaches to this

- RBAC
- Network Layer Access Restrictions via Kubernetes Network Policies
- External Proxies or other Network Devices

We have already covered RBAC, so we will now look at restrictions via a k8s network policy

Network Layer Access Restrictions via K8s

If you are using a k8s implementation with a standard Container Network Interface (CNI) such as Calico, you can use the NetworkPolicy type of resources to restrict traffic. Our lab has Calico networking already configured so we will use that and demonstrate the networking restriction.

Lab: Restrict Admin API via NetworkPolicy

In order to help understand workspaces, we will perform the following tasks

- Deploy NetworkPolicy
- Verify IP restrictions
- Delete Network Policy

Task: Deploy NetworkPolicy

```
$ kubectl apply -f exercises/network-policy/admin-api-restrict.yaml
networkpolicy.networking.k8s.io/admin-api-restrict created
$ kubectl describe networkpolicy admin-api-restrict -n kong
Name:         admin-api-restrict
Namespace:    kong
Created on:   2022-06-27 21:43:53 +0000 UTC
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     app=kong-kong
  Allowing ingress traffic:
    To Port: 8001/TCP
    From:
      IPBlock:
        CIDR: 172.16.0.0/12
        Except:
    From:
      IPBlock:
        CIDR: 192.168.0.0/16
        Except:
-------
$
```

Task: Verify restrictions and Delete NetworkPolicy

```
$ http $KONG_ADMIN_API_URL
^C
$ http kongcluster:30001
$ kubectl delete networkpolicy admin-api-restrict -n kong
networkpolicy.networking.k8s.io "admin-api-restrict" deleted
$
```

Lab: Restrict Admin API via NetworkPolicy

In order to help understand workspaces, we will perform the following tasks

- ✓ Deploy NetworkPolicy
- ✓ Verify IP restrictions
- ✓ Delete Network Policy

Hardening the Kong Control Plane

Hardening Kong

There are a few specific security measures you should consider to harden your Kong Deployment - listing here for reference, but we will not go through each one individually

- RBAC is used following the principle of least privileged access
- Database and cache connections should be configured to use TLS
- Untrusted Lua execution should be disabled
- Read-only Database replicas are configured
- Kong Admin API is restricted at a network level
- Nginx access logs are disabled on the Kong data & control planes
- LDAP & database passwords are not stored in clear text
- Kong should have a minimum TLS version of 1.2
- Kong vitals data is published to a monitoring tool.
- Kong database is restricted
- Kong should be configured to use real IP
- Audit logging is enabled
- Appropriate availability configuration
- Health check for the Kong plane
- Kong server fingerprinting is disabled
- Disable anonymous reporting
- Disable HTTP across the Kong deployment

Summary

In this lesson we

- Described the differences between Authentication & Authorization (Authn/Authz)
- Described RBAC within the context of Kong
- Explained how to secure Kong Manager and Kong Admin API
- Configured Teams and Workspaces
- Listed some general Kong control plane hardening measures

Questions?

What's next?

In the next section we will show you how to secure services on Kong

Thank You