

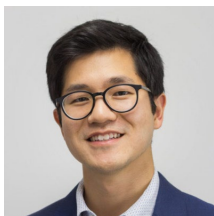


# Dynamic Security Roles in Airflow for Multi-Tenancy

Airflow Summit 2021



# Airflow MCs



Mark Merling -  
Data Engineering  
Manager at Maven  
/e



Sean Lewis -  
Senior Data Engineer  
at Excella Consulting



# Multi Tenancy Need

1. Airflow Adoption
2. Organizational Structure
  - Small contractually separate Agile teams
    - Focus on rapid development
    - Enable shared feature development
1. Technology Landscape
  - Multiple Airflow Instances hosted on AWS EC2s with respective PostgreSQL - RDS Instances
    - Each team should only be able to view their own DAGs in the UI
    - Cost optimization across multiple resources through consolidation
    - Code deployed to each instance through Jenkins with independent deployment cadence



# State of RBAC in Airflow 1.10.14\*

## Limitations

- New roles have to be manually created including assignment of permissions
- New and existing users have to be maintained by an admin
- Roles are not tied to organizational groups, for example Active Directory groups

## Solutions

- Roles are created and named automatically based on the config file
- Roles do not have permissions to see all DAGs by default
- Only when `can_edit`, `can_read` is added to the DAG creation where the role is specified can the DAG be viewed

\* These were true when we started implementation of our multi-tenant approach back in Fall of 2020. Since then Flask AppBuilder has incorporated a few of the key changes which Airflow 2.0+ supports. With some differences.

# Solution: Airflow LDAP/RBAC Implementation

- Enable RBAC in Airflow config
- Webservice config
  - a. Configure LDAP
  - b. Set public role as default.
  - c. Define rule for checking membership.
  - d. Create mapping dictionary where the key is the new role name, and value is the AD group.
- Create custom security class
  - a. Inherit from the security class
  - b. Initialize custom role mappings
  - c. Using the additional config variables, add new functions and overwrite base functions in the FAB package to enable automatic mapping of roles based on group membership.
  - d. Refresh role membership of user at login.

## Webserver.cfg - Role Implementation

```
65 SECURITY_MANAGER_CLASS=AirflowSecurityManagerCustom
66 AUTH_USER_REGISTRATION=True
67 AUTH_LDAP_GROUP_FIELD="memberOf"
68 ROLE_MAPPING = {"role1":"ldap_group1",
69                "role2":"ldap_group2"}
```

# DAG Creation

```
17     dag = airflow.DAG(  
18         dag_id=dag_id,  
19         schedule_interval=schedule_interval,  
20         template_searchpath=template_path,  
21         max_active_runs=max_active_runs,  
22         default_args=default_args,  
23         default_view=default_view,  
24         start_date=start_date,  
25         is_paused_upon_creation=is_paused_upon_creation,  
26         catchup=catchup,  
27         access_control={"grants-analytics-team":{"can_dag_read","can_dag_edit"}},  
28     )
```

# Custom Airflow Security Manager Class - Role Mapping Init

```
19     def __init__(self, appbuilder):
20         super().__init__(appbuilder)
21         self.role_mapping = self.appbuilder.get_app.config["ROLE_MAPPING"]
22         self.auth_ldap_group_field = self.appbuilder.get_app.config["AUTH_LDAP_GROUP_FIELD"]
23         for role_name, role_group in self.role_mapping.items():
24             self.ROLE_CONFIGS.append({'role': role_name, 'perms': self.VIEWER_PERMS | {"can_failed",
25             "can_blocked", "can_dagrun_success", "can_dagrun_failed"},
26             'vms': self.VIEWER_VMS | {"Admin", "XComs", "XComModelView"}}})
```



# Custom Airflow Security Manager Class - Assigning Roles

```
216         # Calculate the user's roles
217         user_role_objects = []
218         if len(self.role_mapping) > 0:
219             user_role_keys = self.ldap_extract_list(
220                 user_info, self.auth_ldap_group_field
221             )
222             user_role_objects += self.get_roles_from_keys(user_role_keys)
223             #example to force someone to be an admin if needed
224             #if username == 'test_admin':
225                 # user_role_objects +=[self.find_role('Admin')]
226         if self.auth_user_registration:
227             user_role_objects += [
228                 self.find_role(self.auth_user_registration_role)
229             ]
```

# Custom Airflow Security Manager Class - Refreshing Roles

```
273     #This will update dag permissions upon login,  
274     #more elegant solution would be after dags are created  
275     args = argparse.Namespace()  
276     Cli.sync_perm(args)
```

## Airflow 2.0+

- Airflow 2.0+ uses later versions of Flask AppBuilder compared to 1.0+
- In 3.2.0 of Flask AppBuilder new config variables were added:
  1. AUTH\_ROLES\_MAPPING
  2. AUTH\_LDAP\_GROUP\_FIELD
  3. AUTH\_ROLES\_SYNC\_AT\_LOGIN

# Summary - RBAC Custom Implementation

- Cost mitigation solution
- Shared resources with privacy enforced
- Code collaboration and shared best practices
- Independent production deployability

# Conclusion/Questions

