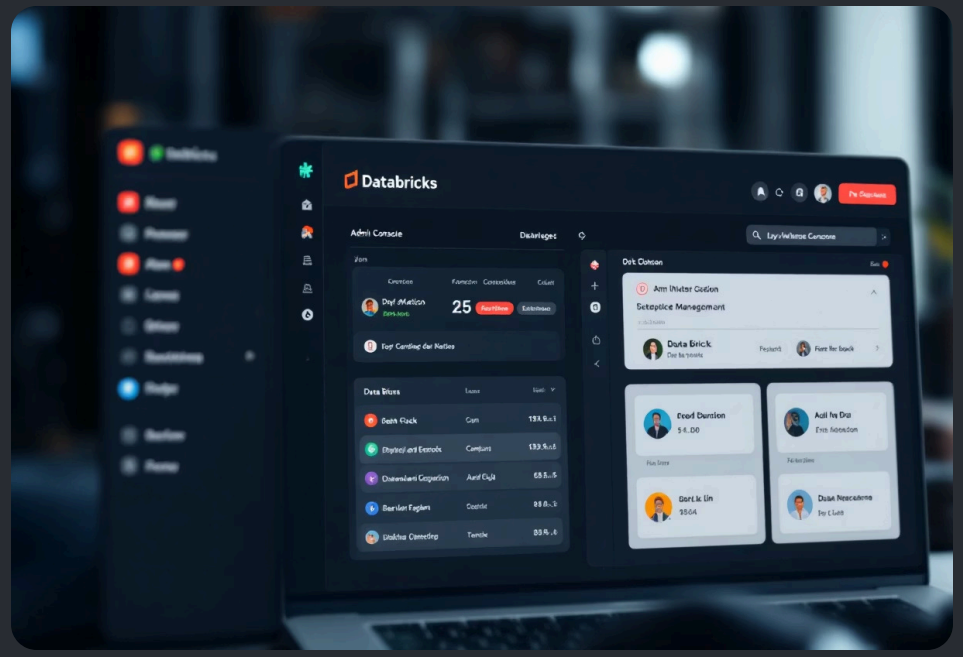
Integrating Zipher with Your Databricks Workspace: A Step-by-Step Guide

Welcome to Zipher! Our mission is to help you optimize your Databricks job clusters, leading to significant cost savings and improved stability. To enable Zipher to analyze your jobs and apply optimization decisions, a secure integration with your Databricks workspace is necessary.

Introduction and Prerequisites

This guide outlines the required permissions and provides a step-by-step manual for generating the necessary credentials and granting access. We are committed to the principle of least privilege, ensuring Zipher only has the access essential for its operations.



Prerequisites

- Access to your Databricks Account Console with account admin privileges (for creating Service Principals and generating OAuth secrets)
- Databricks workspace admin privileges or sufficient permissions to modify job permissions if not an account admin
- A list of the specific Databricks Job IDs that you want Zipher to optimize
- (If applicable) The ability to update network IP Allow Lists for your Databricks workspace

Permissions Required: A Quick Summary

For Zipher to function effectively, it needs specific permissions on the Databricks jobs you wish to optimize:

Permission Level	Scope	Why?
CAN MANAGE	This permission should be granted to Zipher's dedicated Service Principal **only for the specific Databricks Job(s)** you want Zipher to manage. This is as defined within the Databricks **Job Access Control Lists (ACLs)**.	This level allows Zipher to: • **Edit Job Settings:** Modify job definitions, including job cluster configurations (instance types, worker counts, auto-scaling, Spark parameters) via the Databricks Jobs API. • **Manage Job Runs:** Start or cancel job runs. This is vital for stopping inefficient runs to save costs.

In addition, we also need to request **Compute ACL: CAN ATTACH TO** permissions. This is critical as it provides:

1. **"View Spark UI"**
2. **"View compute metrics"**

These metrics are essential for Zipher to analyze cluster performance (CPU/GPU utilization, worker load), and gives us the wanted visibility into the performance of the jobs cluster, executors, workers, and other aspects relevant for our models to reach decisions and actions.

This is in addition to the **can manage** permissions we now have for the job access control lists ACLs.

Zipher does **not** require workspace administrator privileges or broad permissions on other Databricks objects like general compute clusters, notebooks, or dashboards for its core job optimization tasks.

Authentication: Using a Databricks Service Principal with OAuth

For secure, programmatic access, Zipher will interact with your Databricks workspace using a **Databricks Service Principal** authenticated via **OAuth M2M (machine-to-machine)**. This is a robust and recommended method for service-to-service communication.

1

Create a Databricks Service Principal

A Service Principal acts as the dedicated identity for Zipher within your Databricks environment.

1. As a Databricks account admin, log in to your Databricks Account Console.
2. Go to the "Identity and access" section (or similar, e.g., "User management" -> "Service principals").
3. Click "Add service principal" and give it a descriptive name (e.g., zipher-optimization-sp).
4. For detailed, up-to-date instructions, please refer to the official Databricks documentation: **Manage service principals**.

Important: Once created, note down the **Application ID** (also known as Client ID) of the Service Principal. You will need this later.

2

Generate an OAuth Secret for the Service Principal

This secret, along with the Application ID, will be used by Zipher to authenticate.

1. **In the Account Console,** find the Service Principal you just created.
2. **Generate a Secret:**

- Navigate to the "Credentials" or "OAuth secrets" tab for the Service Principal.
- Click "Generate secret."
- **Copy the generated secret immediately and store it securely** (e.g., in a dedicated secrets manager). You will not be able to see this secret again after closing the dialog.
- Ensure this Service Principal is added as a user to any workspace Zipher needs to access.

3

Grant CAN MANAGE Permission on Target Jobs

Grant the Service Principal the CAN MANAGE permission on each Databricks Job you want Zipher to optimize.

Granting Job Permissions

Identify Your Job IDs

- Navigate to your Databricks workspace and go to "Workflows" (Jobs).
- Open each job you want Zipher to manage. The **Job ID** is usually visible in the URL or the job details panel.

Granting Permissions via the Databricks UI

(Recommended for individual jobs)

1. For each target job, go to the job's page.
2. Click on the "Permissions" tab.
3. Click "Add principal" (or "Assign", "Grant").
4. In the "Principal" field, search for and select the Service Principal you created (e.g., zipher-optimization-sp).
5. Assign the CAN MANAGE permission level from the dropdown.
6. Click "Add" or "Save."

Verify Permissions (UI): After adding, confirm the Service Principal is listed with the CAN MANAGE permission.

Granting Permissions via the Databricks REST API

(for automation or multiple jobs)

This requires an Admin Personal Access Token (PAT) or an OAuth token with privileges to modify job permissions. The Databricks CLI can also be used to script these operations.

```
# Endpoint: PUT /api/2.0/permissions/jobs/{job_id}
# Request Body to set/replace permissions:

{
  "access_control_list": [
    {
      "service_principal_name": "YOUR_SERVICE_PRINCIPAL_APPLICATION_ID",
      "permission_level": "CAN_MANAGE"
    }
    // To preserve other existing permissions, list them here too.
  ]
}

# Example using curl:
ADMIN_TOKEN="<your_admin_pat_or_oauth_token>"
WORKSPACE_URL="https://your-workspace.cloud.databricks.com"
JOB_ID_TO_CONFIGURE="<job_id>"
ZIPHER_SP_APP_ID="<zipher_service_principal_application_id>"


curl -X PUT "${WORKSPACE_URL}/api/2.0/permissions/jobs/${JOB_ID_TO_CONFIGURE}" \
  --header "Authorization: Bearer ${ADMIN_TOKEN}" \
  --header "Content-Type: application/json" \
  --data '{
    "access_control_list": [
      {
        "service_principal_name": """${ZIPHER_SP_APP_ID}""",
        "permission_level": "CAN_MANAGE"
      }
    ]
  }'

# Verify Permissions (API):
curl -X GET "${WORKSPACE_URL}/api/2.0/permissions/jobs/${JOB_ID_TO_CONFIGURE}" \
  --header "Authorization: Bearer ${ADMIN_TOKEN}"
```

Network Configuration and Information Sharing

Configure Network Access (IP Allowlisting - Recommended)

For an added layer of security, if your Databricks workspace uses IP Allow Lists, please add Zipher's egress IP addresses. This ensures that API requests from Zipher originate from recognized sources.

- **Zipher Egress IPs:** [Zipher will provide these static IP addresses or ranges]
- **Action:** Add these IPs to your Databricks workspace's IP Allow List. Refer to Databricks documentation: **Configure IP access lists**.

Securely Share Information with Zipher

Once the Service Principal is created, has an OAuth secret, and is granted the necessary job permissions, please provide the following information to the Zipher team through a mutually agreed secure channel:

1. **Databricks Workspace URL** (e.g., https://<your-workspace-id>.cloud.databricks.com)
2. **Zipher Service Principal Application ID** (from Step 1)
3. **Zipher Service Principal OAuth Secret** (generated in Step 2)
4. **List of Databricks Job IDs** that Zipher should optimize (from Step 3)

Verify API Connectivity (Optional)

You can perform a basic check to ensure the workspace API is accessible. This tests general API health and does not verify Zipher's specific job permissions.

Method 1: Using a User's Personal Access Token (PAT) - Recommended for a quick manual test:

```
# Ensure you replace <your-workspace.cloud.databricks.com> with your actual workspace URL
# and <A_VALID_USER_PAT_FOR_TESTING> with an active PAT.
curl --silent --output /dev/null --write-out "%{http_code}\n" \
  https://your-workspace.cloud.databricks.com/api/2.0/workspace/list \
  --header "Authorization: Bearer <A_VALID_USER_PAT_FOR_TESTING>"
# Expected output: 200
```

Method 2: Using the Zipher Service Principal's OAuth Token:

Alternatively, for consistency with Zipher's operational authentication, an OAuth M2M token obtained for the Zipher Service Principal (as configured in Steps 1 & 2) can also be used to perform this connectivity check.

Manually obtaining this token for a one-off `curl` test typically involves:

1. Identifying your Service Principal's Client ID (`<zipher_service_principal_application_id>`).
2. Using its Client Secret (`<zipher_service_principal_oauth_secret>`).
3. Making a POST request to the Databricks token endpoint (e.g., `https://your-workspace.cloud.databricks.com/oidc/v1/token`) with `grant_type: "client_credentials"` and the appropriate `scope` (e.g., `all-apis`) to get an access token.

Once you have the Service Principal's access token (let's say it's stored in a variable `SP_OAUTH_TOKEN`), the `curl` command would be:

```
# Ensure you replace <your-workspace.cloud.databricks.com> with your actual workspace URL
# and <SP_OAUTH_TOKEN> with the actual OAuth token obtained for the Zipher Service Principal.
curl --silent --output /dev/null --write-out "%{http_code}\n" \
  https://your-workspace.cloud.databricks.com/api/2.0/workspace/list \
  --header "Authorization: Bearer <SP_OAUTH_TOKEN>"
# Expected output: 200
```

While this method aligns with how Zipher authenticates, using a PAT (Method 1) is generally more straightforward for a quick, manual API health check performed by an administrator.

Security Best Practices

Least Privilege

Only grant CAN MANAGE on jobs Zipher is actively optimizing.

Secret Management

Treat the Service Principal's OAuth secret with utmost care. Store it securely and use secure methods for sharing it with Zipher. Rotate secrets if a compromise is suspected.

Audit Trails

Databricks audit logs record actions performed by the Zipher Service Principal. Regularly review these logs for monitoring and compliance. Example query:

```
-- Adjust table name and time window as per your audit log setup
SELECT * FROM system.access.audit -- Or your configured audit log table
WHERE principal = '<zipher_service_principal_application_id>'
 -- or servicePrincipalName
AND event_time >= now() - INTERVAL '24 hours'
ORDER BY event_time DESC;
```

Emergency Access Revocation

Using the Databricks UI (Recommended)

1. Navigate to the job in your Databricks workspace.
2. Go to the "Permissions" tab.
3. Find the Zipher Service Principal.
4. Remove the principal or change its permission to "No Permissions."
5. Remove our IP's from the allow list
6. **Verify:** Check the permissions list again to ensure Zipher's SP access is removed or appropriately downgraded.

Using the Databricks REST API

- Update the job's permissions with a PUT request to /api/2.0/permissions/jobs/{job_id}, providing an access_control_list that omits the Zipher Service Principal or explicitly assigns it "NO_PERMISSIONS."
- **Verify:** Use the GET permissions endpoint (as in Step 3) to confirm revocation.

Final Checklist and Support

Final Integration Checklist

1. Service Principal created in Account Console, and Application ID noted
2. OAuth secret generated for the Service Principal and stored securely
3. Service Principal added to the relevant Databricks workspace(s)
4. CAN MANAGE permission granted to the Service Principal on all target Job IDs
5. (Recommended) Zipher's egress IPs added to your workspace IP Allow List
6. Workspace URL, SP Application ID, and SP OAuth Secret ready to be securely shared with Zipher
7. List of Job IDs to be optimized ready to be shared with Zipher

Support

If you have any questions or require assistance during this integration process, please don't hesitate to contact the Zipher support team at [Zipher Support Email/Portal].