Auth0 (Draft)

IRLCA - IQMS - Auth0

Description: This document describes integration of Auth0. The purpose is to document steps and processes involved in the configuration.

Version: 1.0.0

Date: Dec 19, 2023

Author: Vlada Jerkovic

Table of contents

Auth0 Account Creation and Integration with Terraform

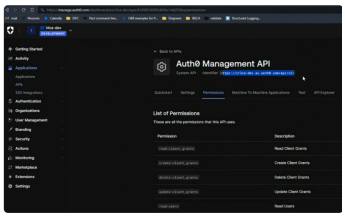
Overview

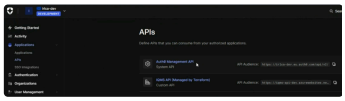
This document outlines the process of integrating Auth0 with Terraform for development purposes. It includes steps for setting up an Auth0 account, configuring Terraform, and creating necessary applications and credentials.

Auth0 Account Setup

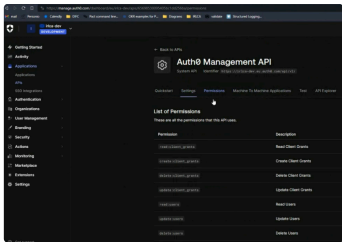
Account Creation: Initiated the creation of an Auth0 account using the '**irlca-dev**' tenant. The account is accessible at Auth0 Management Dashboard.

Purpose of API in Auth0: The Auth0 API, functioning as a resource server, provides essential functionality and data management capabilities. This includes predefined system APIs which are non-deletable and encompass all possible operations within the Auth0 environment. System API can't be deleted.





Operations via API: The API facilitates various operations that can be performed on the UI, such as creating, managing client grants, and handling client keys.

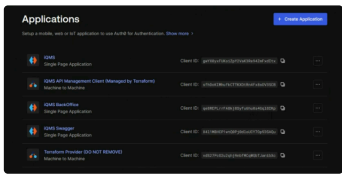


Terraform Access and Application Creation

Access Requirement for Terraform: To configure Auth0 with Terraform, it is crucial for Terraform to have access to the Auth0 API.

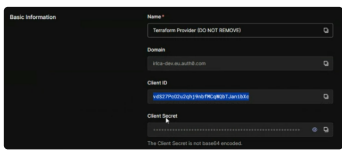
Creating Application for Access: To grant Terraform this access, an application within Auth0 was created.

Initial Cleanup: As a preliminary step, the default applications within the tenant were cleaned, leaving only the essential system API.



Terraform Provider Client: A new Terraform provider client of type 'machine-to-machine' was created.

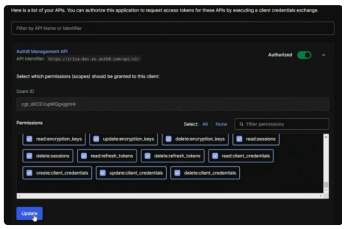
Retrieving Credentials: From this client, the '**Client ID**' and '**Client Secret**' were obtained.



Terraform Variable Configuration: These credentials were then configured as variables in Terraform, authorizing it to manage various aspects of the Auth0 setup. The source code for this project is hosted in the **Azure DevOps Repositories** under the repository named

'irlca.iqms-infrastructure.' For detailed information and access to the codebase, please refer to the corresponding repository in Azure DevOps.

Inclusive Permission Selection: All available permissions were selected and updated in the Auth0 application to ensure comprehensive access for Terraform.



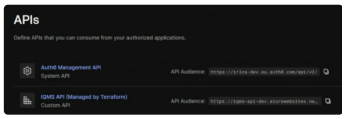
Manual Configuration for Each Tenant

Requirement of Manual Updates: It is important to note that these configurations and permission settings need to be manually replicated for each tenant in Auth0.

Significance of Manual Setup: This manual process ensures that the Terraform integration is tailored specifically to the needs and security requirements of each individual tenant.

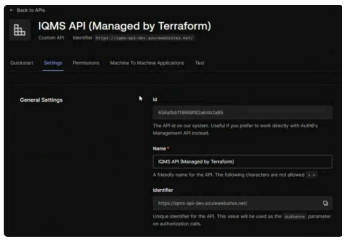
API and Application Configuration

API Definition: The first step involved defining the IQMS API in Auth0, which is now managed by Terraform. This entity in the API was created to facilitate the integration.



Application Access to API: Subsequently, applications were set up with access to this newly defined API.

Application Setup in Auth0



Different Applications: Three distinct applications were configured:

- **IQMS Main Frontend**: Serves as the primary user interface.
- **Back Office**: Acts as the administrative segment of IQMS.
- **Swagger**: Enables login capabilities within Swagger.

Application Function: Each application functions like a unique login form.

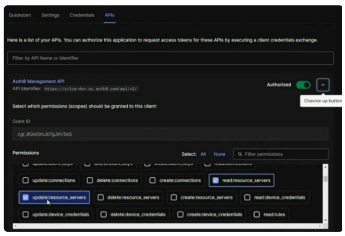
IQMS API Management Client

Backend Integration: The IQMS API Management Client, part of the backend, is responsible for modifying certain aspects within Auth0, such as role creation and permission propagation.

Permission Management: This includes managing organizations, assigning users within these organizations, and handling metadata (although metadata usage is being phased out).

Resource Server and Role Management

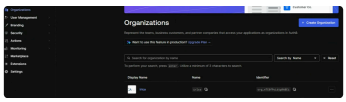
Resource Servers: Access is provided to update and set permissions within resource servers.



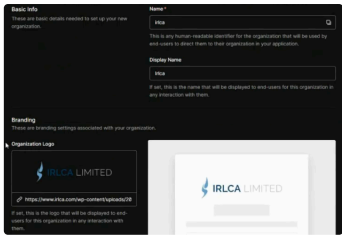
Role Assignment: Roles are managed and assigned to users within organizations, a key feature of Auth0's structure.

Organizations in Auth0

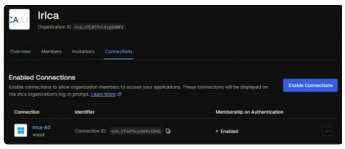
Tenant Representation: Organizations represent external parties or tenants like IRLCA



Login Form Presentation: Although not extensively used, the ability to customize login forms for each organization is available.

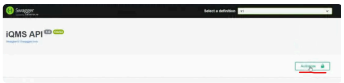


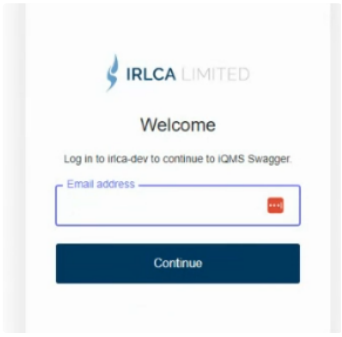
Enterprise Connection: An enterprise connection to Active Directory was established, allowing for differentiated login experiences based on the organization.



Authentication Process

Login Mechanism: Demonstrated how the login process varies based on the organization, with redirections to different authentication services like Active Directory.

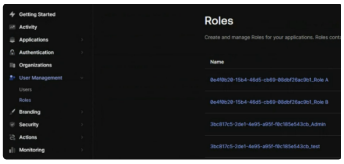




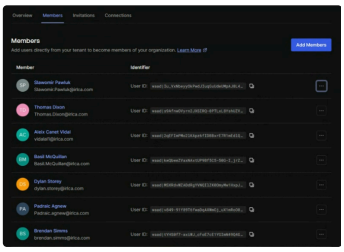
Single Sign-On Functionality: Explained the use of single sign-on tools for seamless authentication across services.

Backend Management

Member and Role Assignment: Members within organizations are assigned from the backend, along with their respective roles.
Role Generation: Roles within the system are generated by the backend. They are not configured directly in Auth0 but are automatically created as part of the backend processes.



User Creation Upon Login: Users are created in the system when they log in for the first time, ensuring that user management aligns with actual system usage.



Terraform Integration

Comprehensive Configuration: The entire setup, including the authentication profile, is configured by Terraform. This includes defining APIs, managing applications, and setting up roles and permissions.

Authentication Profile Configuration and Role Management in Auth0

Authentication Profile Configuration

Purpose of Authentication Profile: The authentication profile is crucial for managing how users are directed to different authentication methods based on their organization.

Discovery of Organizations: To facilitate the discovery of different organizations, users need to enter their email address first. This action determines whether they will be redirected to an external Active Directory or a username/password login.

Organization Discovery Process

Identifier Usage: The process requires the use of a specific identifier to correctly navigate the user to the appropriate authentication method.

Email Address as a Key: The user's email address plays a pivotal role in deciding the authentication pathway – either directing to an external directory or a standard username and password interface.

Terraform Configuration for Auth0 Integration

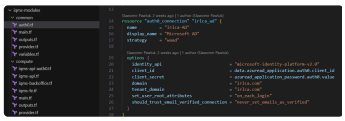
Overview

This section outlines how the Terraform configuration for Auth0 is logically structured, detailing the setup of global resources, organization creation, and application connections.

Terraform Configuration Structure

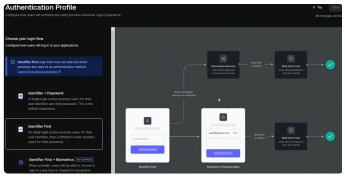
Logical Division: The Terraform configuration for Auth0 is divided into several logical sections to streamline the setup process and enhance understandability.

Global Configuration (Common Auth0): The global resources, such as connections and prompts, form the basis of the Auth0 configuration.



Authentication Profile and Organization Creation

Authentication Profile Configuration: The authentication profile, particularly focusing on the login prompt, is a significant part of the setup.



Organization Creation: An example includes the creation of the 'ircla-dev' organization, which features slight branding, like a logo. However, branding is noted to be less critical in certain redirects, such as when being directed to Microsoft websites.

Organization Connection

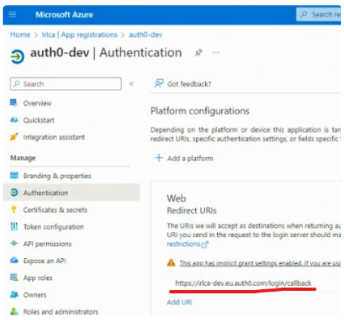
Resource Linking: A specific resource is used to link the 'IRCLA' organization with its respective connection.

Active Directory Integration: The application name from Active Directory (referred to as 'Active directory Entra ID') is integrated into this setup.

Application Registration and Domain Configuration

Auth0-Dev Application: In the Active Directory's application registrations, an application named 'auth0-dev' was created.

Redirect Configuration: The redirect URIs are set based on the documentation, incorporating the domain name of the tenant (e.g., ' Aut h0: Secure access for everyone. But not just anyone. ').



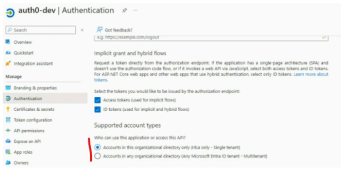
Callback and Redirect Process

Auth0 to Microsoft Redirect: When Auth0 redirects a request to Microsoft, Microsoft, in turn, redirects back to Auth0. This process is crucial for acquiring the access token and ID token.

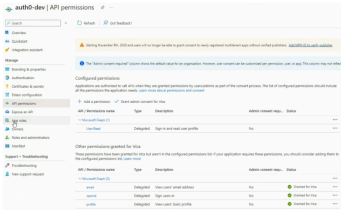
Implicit Grant: The described redirect process is relevant for the implicit grant flow in Auth0.

Application Configuration for Single or Multiple Tenants

Tenant Flexibility: The configuration supports both single and multiple tenant setups.

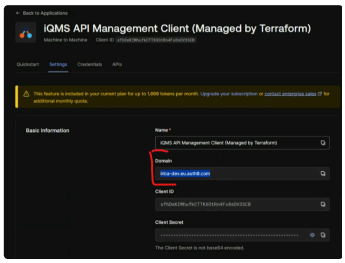


Permission Configuration: Permissions are configured within the application, some of which may be added automatically.



Domain and Tenant Management

Domain Specification: The domain name is crucial for the proper functioning of the applications within Auth0.

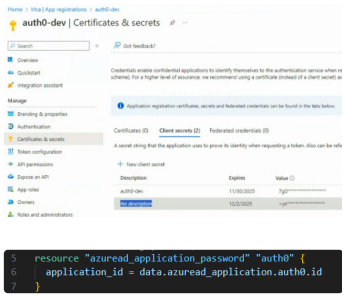


Tenant Identification: The tenant's unique identifier, such as 'ircla-dev', is used across multiple places in the configuration to ensure consistency.

Terraform Data Resource Utilization

Application Resource Retrieval: Terraform uses a data resource to obtain the application resource. This resource retrieval follows a naming convention but can also be derived from variables.

Application Password Secret Management: The setup includes an application password secret, some of which are manually created (but no longer used) and others managed by Terraform without a description.



Secret Management: The secret for the connection is obtained from the password resource managed by Terraform.

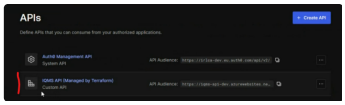
Configuration of Resource Server and API in Auth0 Using Terraform

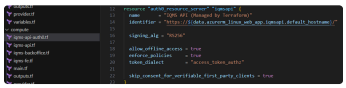
Overview

This section details the steps involved in setting up the resource server and client configurations for API and Swagger using Terraform, focusing on important parameters like the identifier, signing algorithm, and token policies.

Resource Server Configuration

Resource Server Definition: The resource server in Auth0 acts as the definition of the API. The configuration process involves setting various properties of the API.





Key Properties:

- Identifier: The identifier is crucial as it is used in Azure and other applications for integration.
- Signing Algorithm and Offline Access: Configurations like the signing algorithm and offline access are part of the setup.

Policy Enforcement and Token Dialect

Enforce Policies: The configuration includes enabling enforce policies, which are crucial for evaluating permission rules. *enforce_policies* enables **Enable RBAC**.

Token Dialect: The token dialect, though noted as unusual in its functioning, is an essential part of the configuration. It helps in specifying how access tokens are treated and used. *token_dialect* enables **Add Permissions in the Access Token**.

Client Configuration for Swagger

Swagger Client Setup: A client specifically for Swagger is configured, focusing on parameters like allowed URLs and grant types.

Access Restrictions: Currently, login is permitted from development environments and localhost. Swagger will be disabled on test environments.

Management API and API Client

Management API Configuration: The setup includes specific configurations for the management API, aligning it with the overall Auth0 structure.

API Client Configuration: The API client is configured in a way that aligns with the requirements and security standards of the system.

Configuration of Sign-In Actions and Client Settings in Auth0 Using Terraform

Overview

This section provides insights into configuring sign-in actions and client settings in Auth0 using Terraform. The process includes assigning actions, managing triggers, and configuring client settings for different applications.

Sign-In Action Configuration

Accessing Configuration on Auth0 Portal: The initial step involves navigating to the Auth0 portal and accessing the Actions section. Within Actions, the Library is explored to manage custom actions.

Custom Actions: Custom actions generated by Terraform are examined, ensuring that the appropriate action is selected for configuration.

Trigger Assignment: Multiple flows are available, with a focus on the login flow. The specific action is assigned within the login flow configuration.

Client Configuration

Client Definition for Back Office: A client is defined for the back office, managed by Terraform. It is categorized as a single-page application.

Organization-Specific Settings: Configurations related to organizational usage and post-login prompts are emphasized, ensuring consistency across different clients.

JWT Token Configuration: Parameters such as the algorithm for JWT tokens and OpenID conformance are set according to standard practices.

Callback URLs: Callback URLs are specified, with variations for back office and front end, determined by variables. Currently, Azure URLs are used and placeholders are maintained for future Terraform management.

Frontend and Back-Office Client Configuration

Back-Office Client: Specific configurations for the back-office client include organizational usage, post-login prompts, and callback URLs. Variable placeholders are used for future Terraform management.

Front-End Client: Similar configurations are applied to the front-end client, with distinct names and callback URLs.

Output Feature in Terraform

Utilizing Terraform Outputs: The Terraform output feature is utilized to display and copy client IDs conveniently after applying the infrastructure changes.

Configuration of Terraform Credentials Using Azure DevOps Variable Groups

Overview

This section describes an efficient method for configuring credentials in Terraform using Azure DevOps Variable Groups. The approach involves utilizing Variable Groups in Azure DevOps to manage sensitive information such as Auth0 domain, client ID, and Azure client secret.

Variable Group Setup

- **Variable Group Naming**: A Variable Group is created in Azure DevOps, and its name is dynamically built using the group infrastructure dash environment name.
- **Required Variables**: Within the Variable Group, essential variables for Auth0, such as Auth0 domain, Auth0 client ID, and Azure client secret, are defined.

Terraform Provider Configuration

- **Environment Variable Usage**: The Terraform provider configuration for Auth0 references environment variables. The client ID and client secret can be passed either as parameters or environment variables, allowing flexibility.
- **Environment Variable Names**: The environment variables used are AUTH0_CLIENT_ID, AUTH0_CLIENT_SECRET, and AUTH0_DOMAIN. These are set up in the Variable Group.

Advantages of Variable Groups

- **Scalability**: The setup allows for easy scalability. Additional tenants can be accommodated by copying the Variable Group and adjusting the values while keeping the variable names constant.
- **Version Control**: The approach avoids the need to commit another version of the Terraform configuration for each change. The values are stored in the Variable Group, providing a clean and version-agnostic configuration.
- **Secure Storage**: Variable Groups offer a secure way to store sensitive information, and access permissions can be fine-tuned within the Azure DevOps pipeline.

- **Integration with Azure Key Vault**: Variable Groups can seamlessly integrate with Azure Key Vault, enabling the retrieval of values directly from the Key Vault.

Considerations

- **Azure DevOps Permissions**: It's essential to authorize the pipeline to access the Variable Group and manage permissions for editing variables within the group.
- **Secure Credential Handling**: This approach provides a secure way to store credentials, and it is a recommended practice for managing sensitive information.

Authentication Flow Between Frontend and Backend in Auth0

The interaction between the frontend and backend in the Auth0 authentication flow can be explained as follows:

Front-End Interaction:

- **Display and Redirect**: The frontend displays the user interface and initiates the authentication process by redirecting the user to Auth0.
- **User Login**: The user logs in through Auth0, and if the login is successful, Auth0 generates an access token.

Token Usage:

- **Access Token**: The front end receives the access token upon successful authentication.
- **API Request**: The front end utilizes the received JWT (JSON Web Token) access token to make requests to the back-end API.

Back-End API Authorization:

- **Conditions and Endpoints**: The back-end API has a predefined set of conditions for each endpoint.
- **Authorization Check**: When a request is received, the back end checks the access token against the conditions associated with the requested endpoint.
- **Permit or Forbidden**: If the conditions are met, the user is permitted to access the requested resources. If not, a 403 Forbidden error is returned.
- **Unauthorized Access**: If no access token is provided at all, the user receives a 401 Unauthorized error as no authorization information is passed.

Conclusion

This authentication flow ensures that only authenticated and authorized users can access specific resources on the back-end API. The use of JWT access tokens facilitates secure communication between the front end and back end, and the defined conditions determine whether access is granted or denied.

Extra content:

Video explanation between Desmond Farrell, Slawomir Pawluk and Carlos Fuentes: Video