Auth0 Training

Dan Cohen/Matt Bacalakis
Auth0 Solutions Architects

High-level agenda

1. Introduction
2. Core Identity Concepts
3. Protocols
4. Auth0 Introduction
5. Features
6. Extensibility
7. Architecture Scenarios
8. Day-to-day Operations
9. Hands-on Labs

Auth0 Training

1. Introduction

Quick intro to the training course structure

Auth0 5-day training

- 5 days of training, 10x 3h sessions
- 6 instructor sessions (3h each)
- 4 lab sessions

Instructor sessions

- Discussion about Auth0 features and related specs
- Demos of Auth0 platform
- Implementation scenarios
- Q&A

Lab sessions

- Hand-on scenario implementation
- Ok to use any tenant - trial or existing one (non production!)
- Requires development environment
    - Labs were optimised for Node.js and iOS mobile, but can work with any environment

Ready?

Auth0 Training

2. Core Identity Concepts

Introduction to IAM

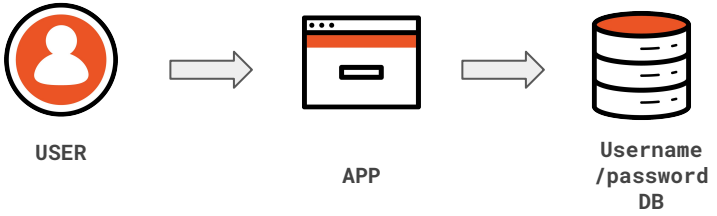
The problem

Key concept in identity and access management (IAM)

- **Authentication** - The process of verifying that a user is who they say they are. This can be done by verifying something the user knows (e.g. a password), something the user has (e.g. a mobile phone number), or something the user is (e.g. biometrics).

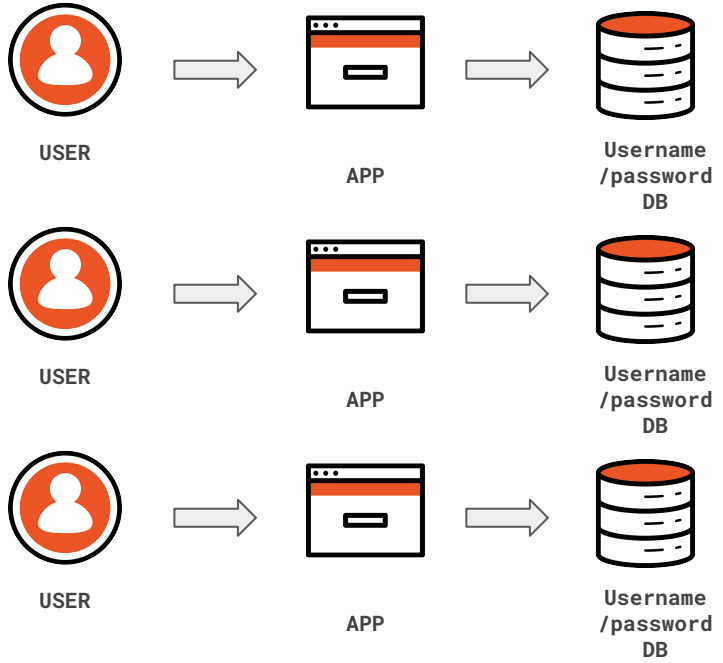
Basic authentication in web apps

- User authenticates with username and password
- The credentials are stored and validated in a database



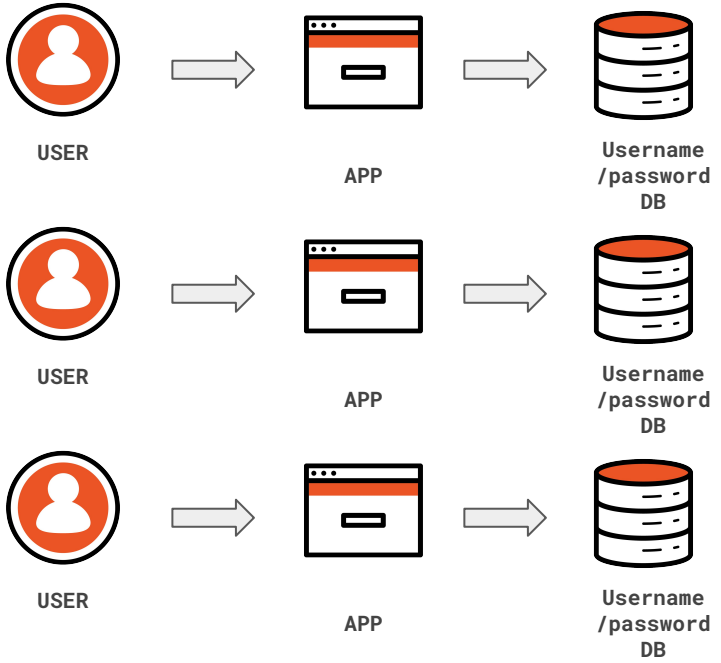
Multiple apps

- Multiple apps = multiple user databases



Problems

- Users must remember multiple passwords
- Passwords will be re-used
- Weak passwords will be invented that are easy to guess
- Passwords stored in cleartext can be stolen
- Forgotten passwords are restored on in one app confusing users



Passwords - encryption

- Encryption is a **reversible** transformation of data.
- Decryption without the key should be computationally expensive.
- **Symmetric** - one key for encryption and decryption.
- **Asymmetric** - one key for encryption and another one for decryption.
- Good to prevent unauthorized access and detect tampering.
- **Not so good to keep passwords secret** - a leaked cipher and key will allow for easy decryption.

Passwords - hashing

- Hashing - mapping of arbitrary data to a fixed size value.
- Password hashing functions should be **computationally expensive to reverse**.
- Leaked password hashes should be very expensive to reverse to plain text passwords.
- What if users use same password? Or attacker precomputes hash values? Append a per-user random value to the password - aka **salt**.
- To verify a password - hash the input and compare with the stored hash. No need for original plain text value!

Passwords - complexity

- Passwords should be difficult to guess and easy to remember. Humans tend to use simple passwords.
- Forcing password expiration leads to predictable passwords (users change a few chars to satisfy the requirements).
- Password complexity rules lead to predictable passwords - users tend to use the simplest password to satisfy the requirements.
- Most reliable complexity metric - password length. Prefer minimum 8 chars, but the more - the better.
- **Do not reuse passwords across systems!**
- Use reliable password managers whenever possible.

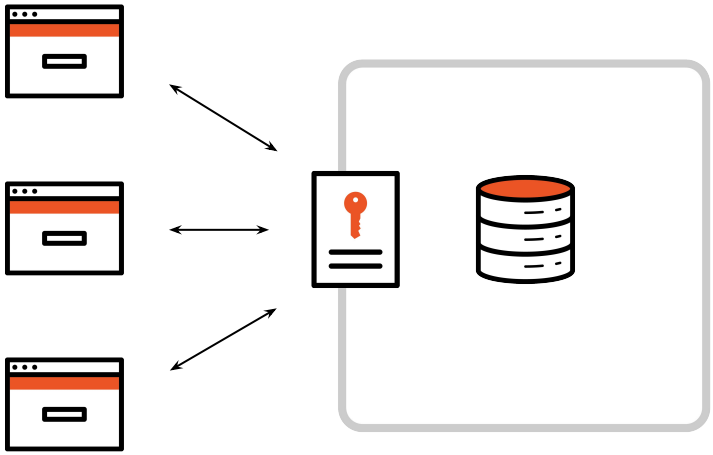
Mitigations

- Data encryption and password hashing
- Password complexity enforcement
- Attack prevention and mitigation - brute force attack detection, account lockout
- Secure infrastructure
- Self-service password reset
- **Still a problem:** multiple passwords

Identity federation and single sign-on (SSO)

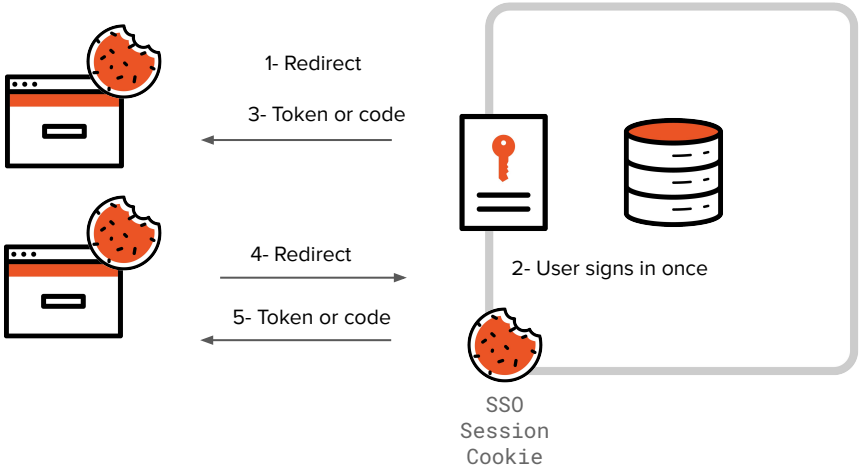
- **Federation** - linking and using the digital identities a user has across multiple systems.

- **SSO** - User logs in with a single ID and password, and gains access to multiple applications and systems in different domains, without needing to enter their credentials more than once



SSO for web apps

- Apps take a dependency (AKA create a trust relationship) with an external authentication server
- Instead of collecting a username and password, the app redirects to the auth servers page for sign-in
- User signs in to auth server and get an SSO session cookie
- Auth server redirects back to the app with a Token that is digitally signed (or better, a code that can be exchanged for a token via a back channel)
- App validates the token, logs the user in, and creates its own session
- Upon accessing a second app, the user is redirected to the auth server, but automatically redirected back with a token due to the SSO session cookie
- Second app validates the token, logs user in, and creates its own session

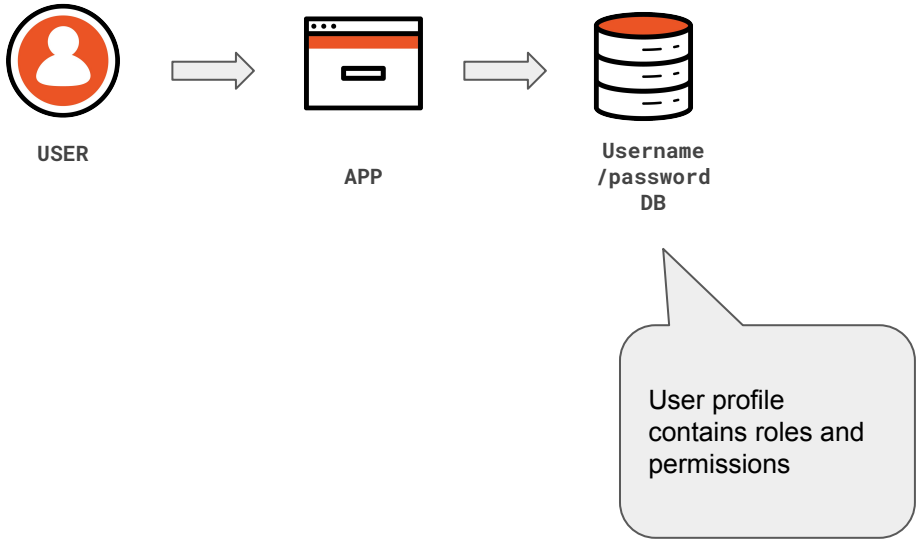


Key concept in identity and access management (IAM)

- **Authorization** - The process of verifying what a user (or client) is allowed to do, or what privileges a client/user has within a given system.

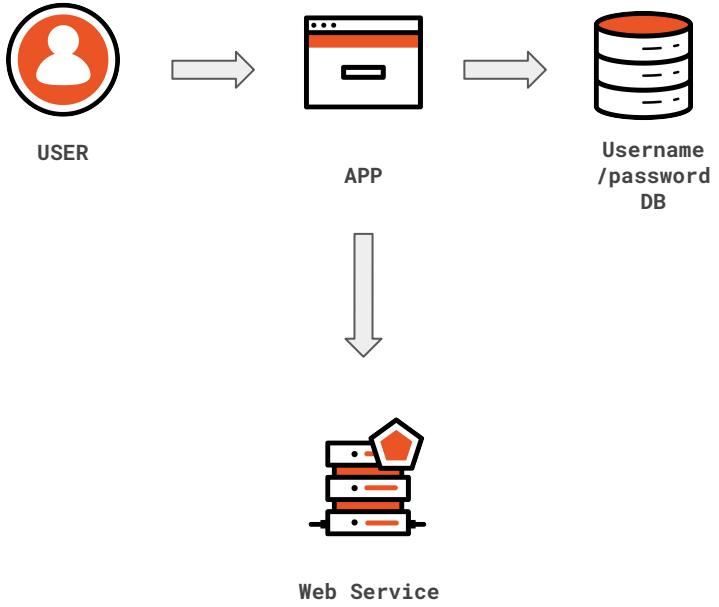
Basic authorization in web apps

- User authenticates with username and password
- The credentials are stored and validated in a database
- App reads role and/or permission information from the use profile record
- Typically, roles and permissions are set by a higher authority and cannot be modified directly by the user



Problem with external services

- What happens when an app needs authorization to an external web service that contains additional user data?



Mitigations

- Need a central authority trusted by all parties
- The authority needs the ability to delegate access to resources
  - On behalf of the user
  - On behalf of the app
- The delegated permission needs to be flexible and, preferably, self contained

End of Chapter 2

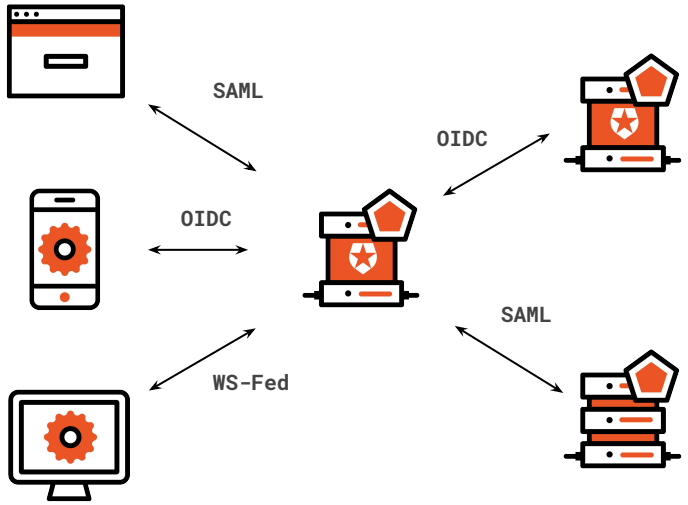
Auth0 Training

3. Protocols

Introduction to supported protocols and terminology

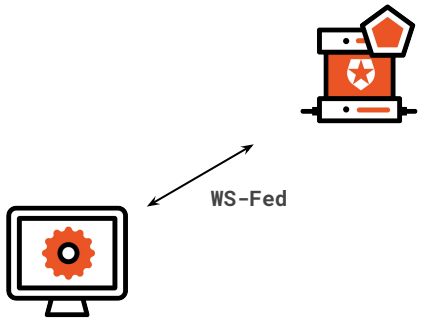
Supported protocols

- WS-Federation
- SAML
- OpenID Connect (OIDC) - with Json Web Tokens (JWT)



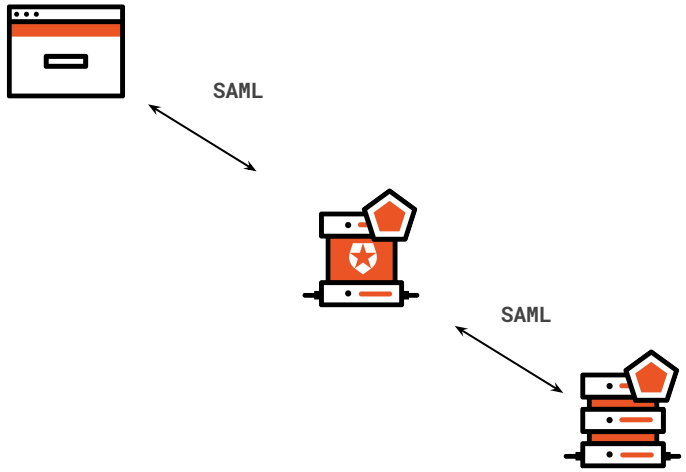
WS-Federation

- Provides a basic model of federation between different Identity Providers and Relying Parties
- Less used due to complexity
- XML based



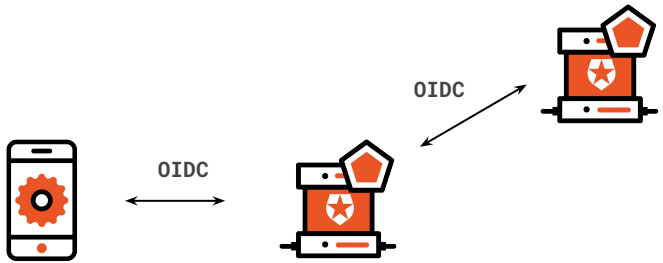
SAML

- XML based
- Often used in enterprise (B2E)
- Much more complex - but also more capable
- Examples - ADFS, Salesforce



OpenID Connect

- Easy to implement (HTTP-based)
- Multiple flows for different clients
- Based on OAuth2
- Auth0 uses JWT token format
- TLS and correct DNS settings are vital for security.



OAuth 2.0

- Standard protocol designed for ***authorization, not user authentication***.
- Enables an application to obtain limited access to a service:
  - On behalf of a user
  - On behalf of the application
- Token-based system that allows a user to be authorized without sharing user credentials
- End result is an **access token** that can be used to call an API.
- Allows for limiting scope of access token i.e. token is good for these operations against the API only.
- Access tokens can be any format, but Auth0 usually uses JWT.

OAuth 2.0 + OpenId Connect 1.0

- OpenID Connect is an identity layer added to OAuth2 to support Authentication.
- OpenID Connect defines its own token type (called an “ID token”) that includes some standard claims such as name, email, birthdate.
- OAuth2 defines methods by which both ID tokens and API access tokens can be delivered to applications

OAuth 2.0 Client Types

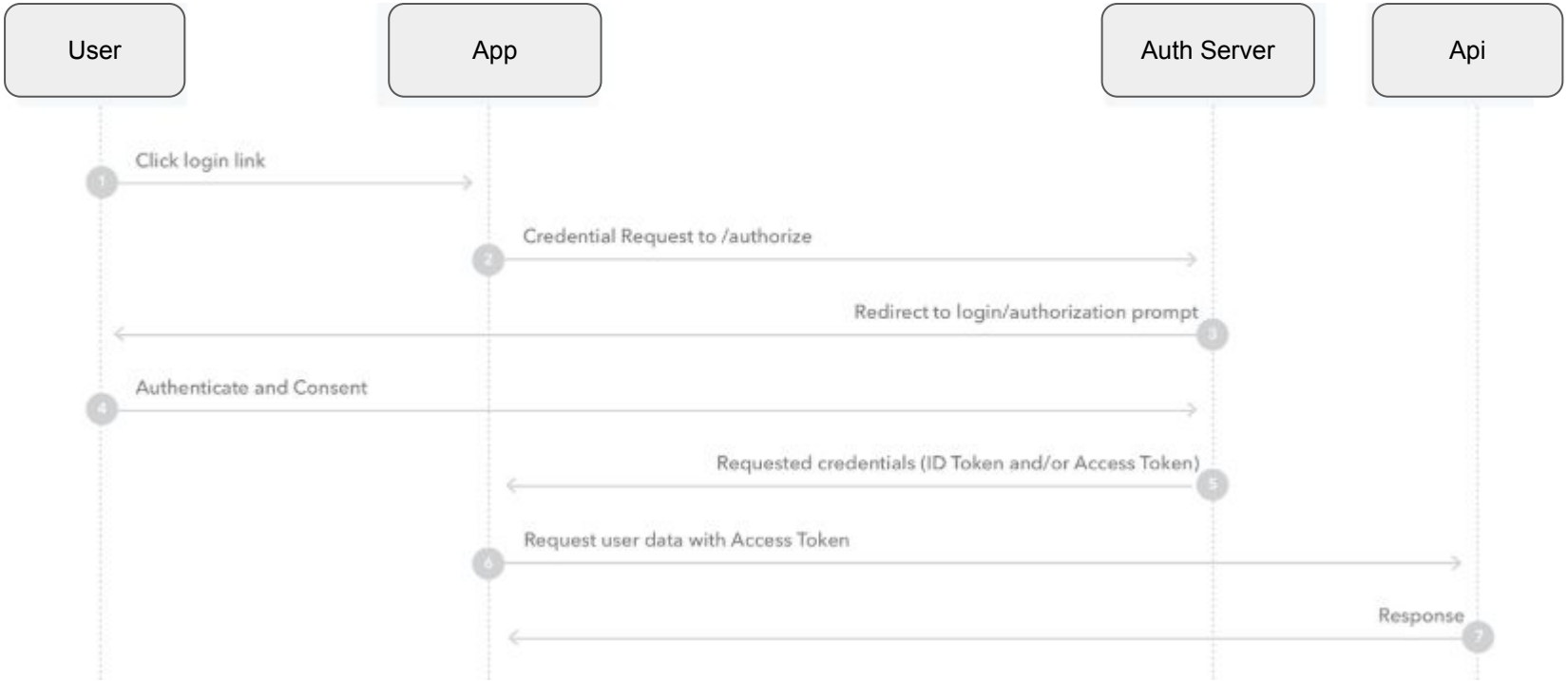
- Confidential
  - Clients capable of maintaining the confidentiality of their credentials, or capable of secure client authentication using other means.
  - Regular Web Application with dedicated backend on a secure server with restricted access to the client credentials
- Public
  - Clients incapable of maintaining the confidentiality of their credentials
  - Clients executing on the device used by the resource owner, such as an installed native application or a web browser-based application
  - Single Page Application

OAuth 2.0 Grant Types

The Oauth2 Spec defines 5 grants types to obtain access tokens for authorization

- Implicit
- Authorization Code
- Resource Owner Password Credentials
- Client Credentials
- Extension Grants
    - Custom grants defined outside the main OAuth2 spec.

Implicit Grant



Implicit Grant

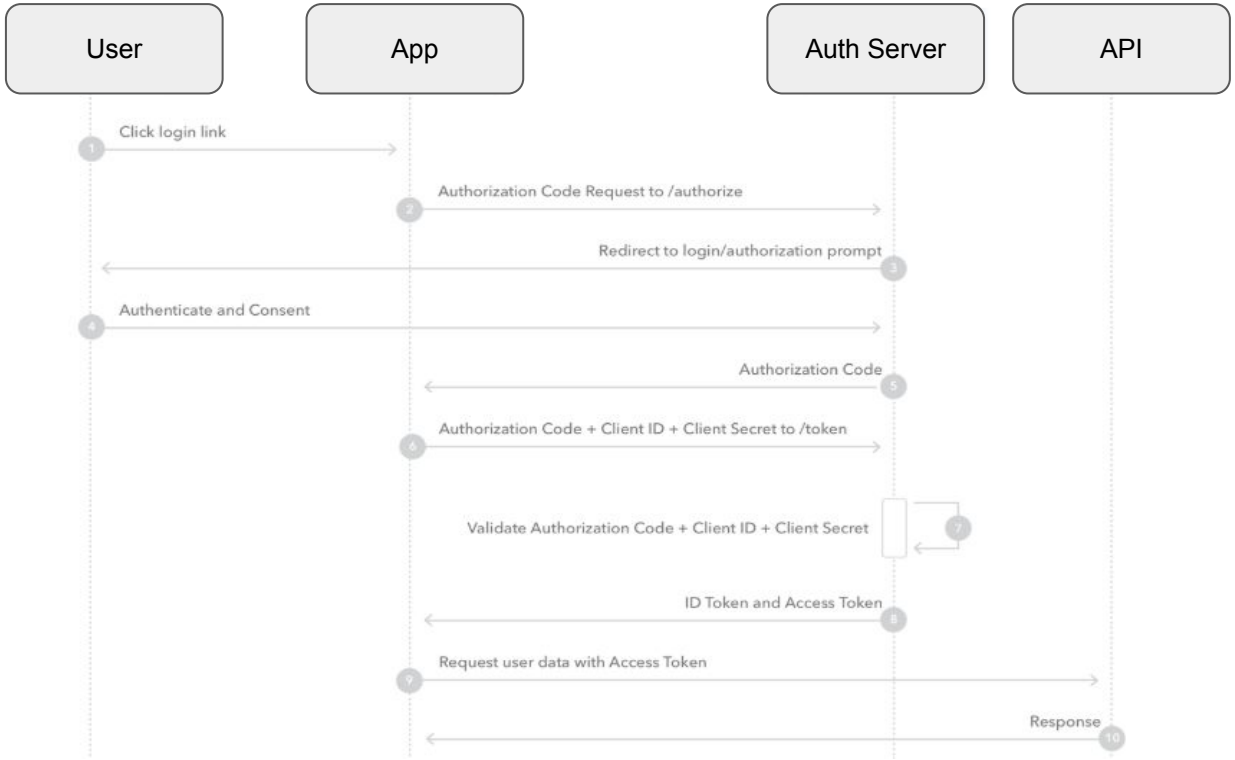
Advantages:

- Can be used by public clients
- Simpler as it has fewer calls.
- Easy storage of tokens as they remain in user's browser.

Disadvantages:

- Less secure as tokens live in user agent (browser) where they are more vulnerable to leaking due to attacks like cross site scripting.
- Auth0 presently does not allow refresh tokens with Implicit.
    - Silent Authentication is a workaround but has limitations
- No longer recommended due to vulnerabilities inherent in flow.

Authorization Code Grant



Authorization Code Grant

Advantages:

- User agent (browser) never has access to Tokens. Only the more secure backend has them.
- Refresh tokens are an option

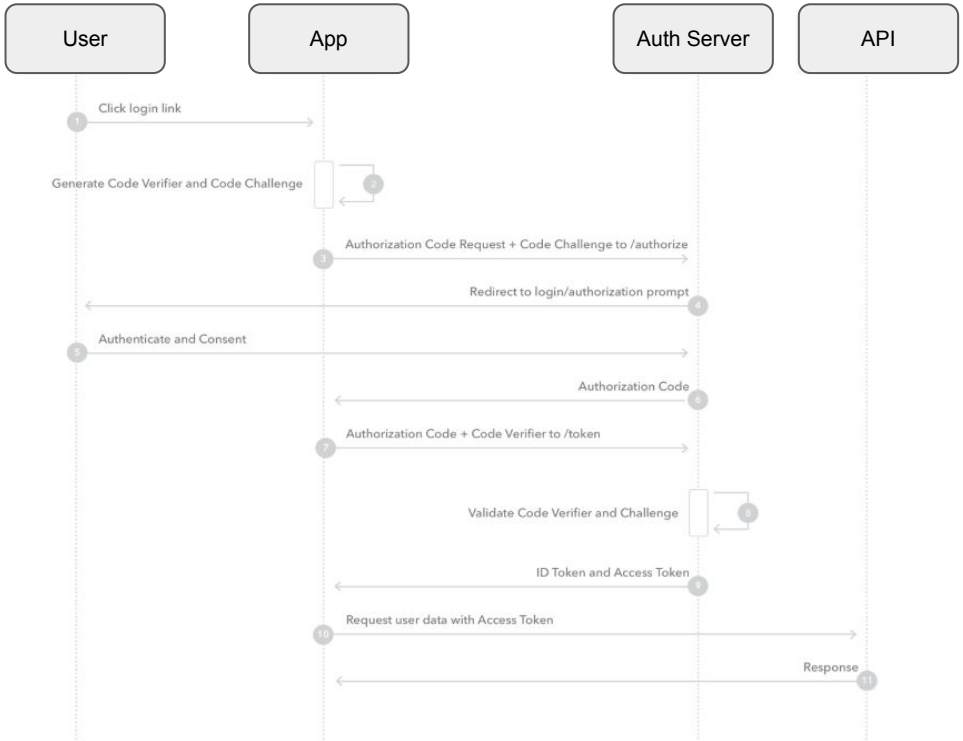
Disadvantages

- Requires confidential client
- A bit extra complexity due to extra calls, must manage session between front and backend.
- Added complexity in storing user's tokens especially if you run multiple instances of backend.

Authorization Code with PKCE

- Uses a code challenge and code verifier instead of client secret.
- Allows Authorization code flow for Public and Native apps.
- Like regular Auth Code grant the tokens are retrieved via a call to the token endpoint avoiding the vulnerabilities implicit has in public clients.
- Already supported by Auth0 SDKs.

Authorization Code with PKCE

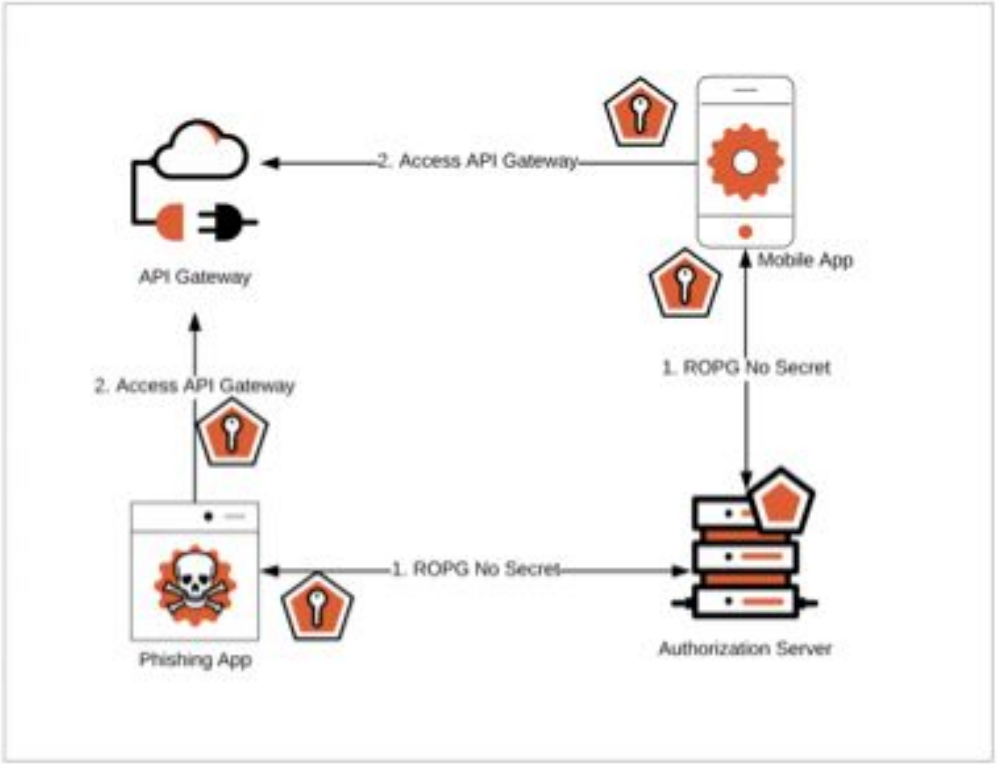


Resource Owner Password Grant

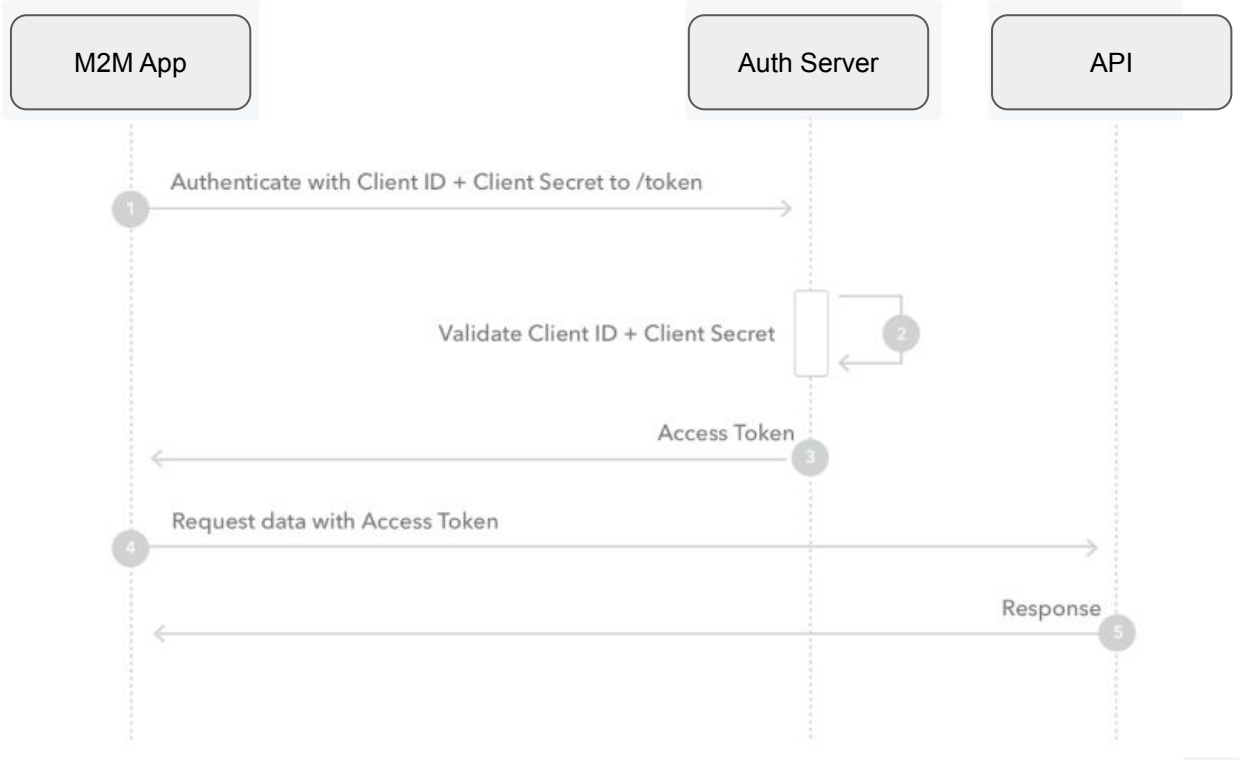
- Meant only for absolutely trusted applications when a redirect flow (Auth Code, Implicit) is not possible.
- Application collects user's username/password and passes them directly to /token endpoint to retrieve access token (and optionally refresh token and/or id token)
- Requires application to have access to user credentials
- Does not support SSO
- Requires extra config for Anomaly Detection to work

Resource Owner Password Grant

- Attack vector - if an attacker can successfully phish a username and password, they can present those to the authorization server and get access tokens if ROPG is allowed.
- This is one reason why flows that require a client secret or the use of PKCE are recommended.



Client Credentials Grant



Client Credentials Grant

- Intended for clients/applications to authenticate and retrieve a token on behalf of themselves.
- Confidential clients only as client secret must be provided.
- Client must be authorized for the API in Auth0 dashboard.
- APIs also allow for defining scopes and granting them on a per application basis
- Client Credentials Hook available to add custom logic to authentications
    - Can manipulate claims
    - Reject authentication
    - Limit scopes
    - Not presently configurable with management api.

Scope In OAuth2

- Limits what actions an application is able to perform on a user's behalf in an api.
  - Access token lets the app call the API on behalf of user, scope says which specific actions are allowed.
  - Ex. Access token allows reading my gmail contacts, but not for sending emails.
- In Auth0 you can define scopes per API.
- When requesting access token for a user application requests specific scopes.
  - For third party applications user will be prompted to approve requested scopes.
  - For first party applications API can be configured to not require consent.
- When requesting access tokens for an application token will have all scopes authorized for that application in Auth0 Dashboard

Scope in OIDC

- Adds scopes to request user information
  - openid - Indicates app wants to use OIDC to verify user's identity. Adds the sub(subject) claim to tokens which indicates the users unique identifier
  - email - App is requesting id token include email and email_verified claim
  - profile - Returns claims that represent basic profile information, including name, family_name, given_name, middle_name, nickname, picture, and updated_at
  - offline_access - app is requesting a refresh token for the user
    - In Auth0 should presently only be done for Auth Code, Auth Code w/ PKCE, and ROPG.
    - API must be configured to allow offline access

Tokens

JSON Web Key (JWK) and JSON Web Key Set (JWKS)

- JSON Web Key is a data structure that represents a cryptographic key
- The cryptographic key is public and used to verify tokens
- JSON Web Key Set is a set of JWK's
- Well known endpoint

  `https://{tenant-name}.auth0.com/.well-known/jwks.json`

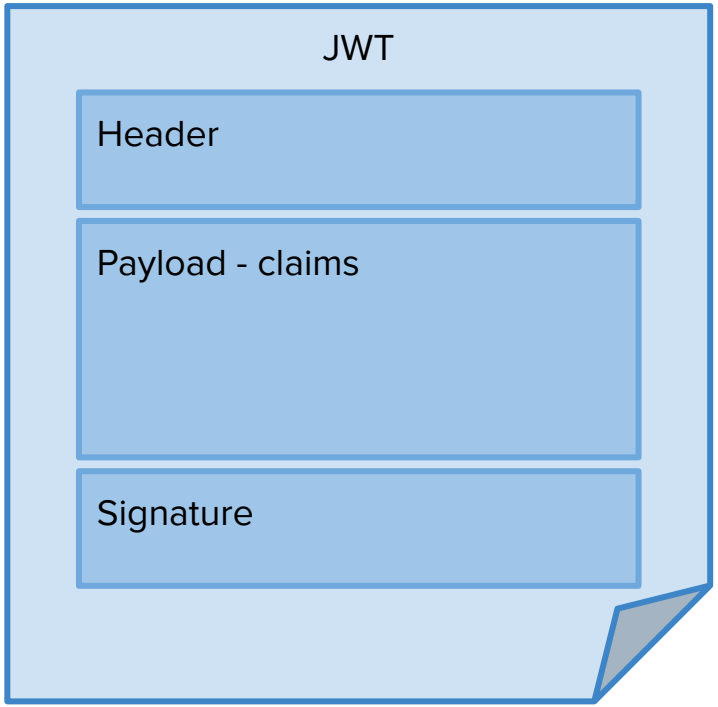
OIDC Tokens

Token types:

- ID Token
- Access Token
- Refresh Token

Typical properties:

- Format - JSON Web Token or opaque
- JWT can be signed and optionally encrypted
- Signed with HS256 or RS256



JSON Web Tokens - JWT

- Open standard (RFC 7519)
- Compact, ideal to sent through a URL, POST parameter or inside an HTTP header
- Self-contained, payload contains all the information no round trip to database
- Base64URL encoded structure `xxxxx.yyyyy.zzzzz`
  - Header, contains used hashing algorithm, HS256 or RS256 (recommended)
  - Payload, contains the claims like **`iss`** (issuer), **`exp`** (expiration time), **`sub`** (subject), **`aud`** (audience), and others.
  - Signature, used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way
- Only available when using OAuth2 (OIDC) between Client and Auth0
  - Note, connections could be some other protocol than OIDC

JSON Web Key Set - JWKS

Tokens can be signed either:

- With a pre-shared key (HS256)
- With an asymmetric key (RS256)

RS256 is preferred as it works with public clients and allows for transparent key rotation by using JWKS.

JWKS - allows the consumer of the token to obtain the public key dynamically from a well-known URL to verify the signature.

```
{
"keys": [
  {
    "alg": "RS256",
    "kty": "RSA",
    "use": "sig",
    "x5c": [
      "MIIC+DCCAeCg...kkgmo="
    ],
    "n": "yeNlzl...oBdjQ",
    "e": "AQAB",
    "kid": "NjV...M2Qg",
    "x5t": "NjVB...2Qg"
  }
]}
```

ID Token

- Specific to OIDC
- Describes the user (“sub” claim) - “who” the user is.
- The Client is the audience (“aud” claim)
- Custom claims must be namespaced
- The client must verify its signature
- Is expected to be consumed immediately
- By default, the token is valid for 36000 seconds, or 10 hours
- Can be changed per Client


```
{
  "header": {
    "typ": "JWT",
    "alg": "RS256",
    "kid": "QjV...1Mw"
  },
  "payload": {
    "iss": "https://artex-dev.eu.auth0.com/",
    "sub": "auth0|5a8e9507a7572d61b3f4b8c9",
    "aud": "gx272e3pkFNAoEKixERWKsUkK5DU3GSP",
    "iat": 1528977654,
    "exp": 1528978654,
    "at_hash": "8ZfpLNmxvhP_VG9zT_PURA",
    "nonce": "awe.fjawy3ftsbmn,"
  },
  "signature": "VGxsT...SmQ"
}
```

How to get an ID Token?

- The `id_token` can be returned when calling any of the Auth0 functions which invoke authentication (must be requested)
- Payload of the `id_token`
  - Controlled by the use of the `scope` parameter
  - Scope parameter passed when authentication is invoked
  - Scope set to `openid` returns `iss`, `sub`, `aud`, `exp` and `iat` claims
  - Scope set to `openid email` returns additionally `email` and `email_verified`
  - Scope set to `profile` returns all default profile Claims;
    - name, family_name, given_name, middle_name, nickname, preferred_username, profile, picture, website, gender, birthdate, zoneinfo, locale, and updated_at

How to verify an ID Token?

- Check that the JWT is well formed
- Verify the Signature
  - HS256
    - Token is validated with the Auth0 *Client Secret*
  - RS256
    - Token is verified with Public Key or Certificate
- Validate the standard claims
  - **`iss`** / issuer
  - **`exp`** / expiration time
  - **`aud`** / audience

Access token

- Specific to OAuth 2.0
- Describes “what” can be accessed (“aud” claim)
- The Resource Server is the audience (“aud” claim)
- Custom claims must be namespaced
- The resource server must verify its signature and the client must treat it as opaque.
- Can have custom scope - it is up to Resource Server to interpret them
- Only one audience per token can be specified
- By default, the token is valid for 86400 seconds, or 24 hours
- Can be changed per API


```
{
  "header": {
    "typ": "JWT",
    "alg": "RS256",
    "kid": "QjVFQzgw...Mw"
  },
  "payload": {
    "iss": "https://artex-dev.eu.auth0.com/",
    "sub": "auth0|5a8e9507a7572d61b3f4b8c9",
    "aud": [
      "http://api.artex-dev.com",
      "https://artex-dev.eu.auth0.com/userinfo"
    ],
    "iat": 1528977654,
    "exp": 1528979654,
    "azp": "gx272e3pkFNAoEKixERWKsUkK5DU3GSP",
    "scope": "openid"
  },
  "signature": "Z-sOQG...U8w"
}
```

How to get an Access Token?

- Access tokens are issued via Auth0's OAuth 2.0 endpoints: `/authorize` and `/oauth/token`.
- Scopes define the additional claims
- Rules / Hooks can set additional claims or restrict scopes

How to verify an Access Token?

- Check that the JWT is well formed
- Check the signature
  - HS256, signature is validated with the Auth0 API Signing Secret
  - RS256, signature is verified with Auth0 tenant's JSON Web Key Set (JWKS)
    - https://{your-tenant}.auth0.com/.well-known/jwks.json
- Validate the standard claims
  - **`iss`** / issuer, **`exp`** / expiration time, **`aud`** / audience, **`sub`** / subject
- Check the Client permissions by validating the content of the **`scope`** claim
  - The `read:users` scope provides access to the `/read` endpoint
  - The `create:users` scope provides access to the `/create` endpoint

Refresh Token

- Completely opaque
- Expiration configurable
  - Non-expiring
  - Rotating
- Must be secure
- Can be revoked
- Not available in Implicit flow
- Apps must store it securely
- Can be disabled at audience (API) level - specific to Auth0

```
https://....auth0.com/authorize?
    ...
    &scope=offline_access

https://artex-dev.eu.auth0.com/oauth/token

{
  "access_token": "eyJ...4ZzWbNw",
  "expires_in": 86400,
  "id_token": "eyJ...L26MZow",
  "refresh_token":
"1QytjXDSk-7KZSPyijcdo05Mdt-MKfToWZRyeOLATPPDD",
  "token_type": "Bearer"
}
```

Refresh Token

- A refresh token allows the application to request Auth0 to issue a new `access_token` or `id_token` directly, without having to re-authenticate the user
- This is useful for mobile applications that are installed on a device or web apps with backend servers
- Refresh tokens can be **obtained** through the authorize or oauth/token endpoint with `scope=offline_access`
- Can be **revoked** programmatically through the Auth0 API or Dashboard
- Refresh token allows a user to remain authenticated essentially forever
  - Cannot have this information in a browser, it must be stored securely
- API setting to prevent issuing refresh tokens

Refresh Tokens vs User Session

A refresh token is NOT a user session!

- **User session**: an active user session with the Identity Provider when IdP is certain that the user is present.
- **Refresh token**: user's permission to access a resource without him being present (note the "offline_access" scope).

Refresh tokens are ok to be used in mobile apps or anywhere where some background activity is made on behalf of the user. Refresh token usage is questionable in interactive web apps, as most of the time silent auth must be used to refresh the tokens - this takes advantage of the user's session and provides correct semantics and security measures.

Due to Auth0 platform limitations, sometimes a refresh token may be acceptable.

End of Chapter 3

Auth0 Training

4. Auth0 Introduction

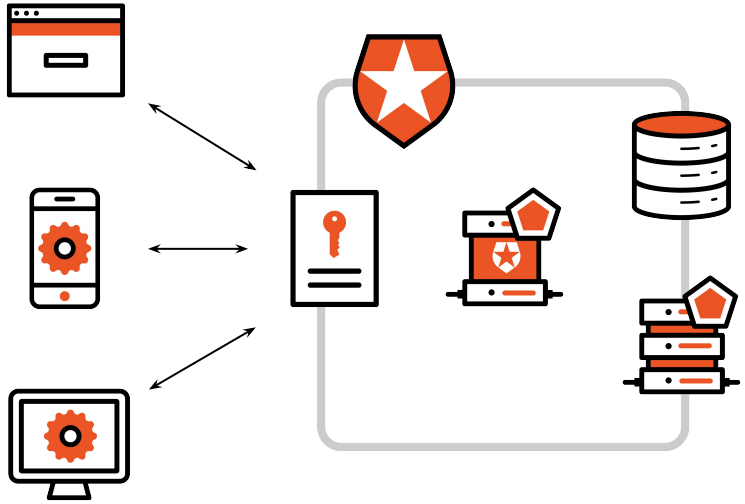
High level introduction to Auth0

Auth0 - Identity Provider as a Service

- IDaaS platform, providing secure, centralized Authentication and Authorization for applications
- Supports multiple types of Identity Providers
  - Social
  - Enterprise
- Pre-built connections to many IdPs
- API Authorization
- Extensible Platform
- Based on open, industry standards

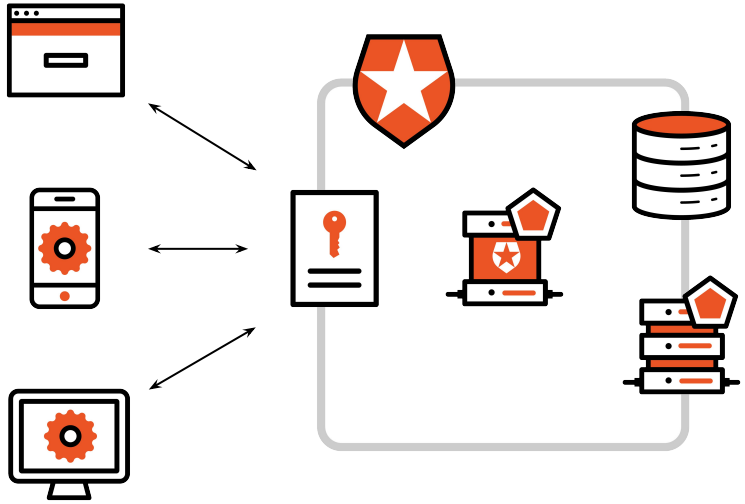
Auth0 - Identity Provider as a Service

- Main logical “unit” - Tenant
- You can have as many tenants as needed
- Tenants can have different purpose (DEV, PROD, QA, etc)
- Each tenant counts its user quota separately
- MAU - Monthly Active Users



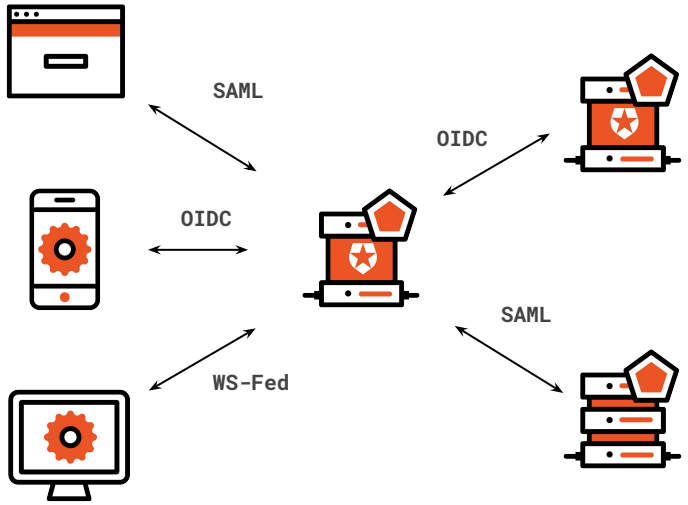
Auth0 Tenant

- Represents a “login brand”
- One set of Hosted Login Pages
- One set of email templates
- One Custom Domain
- Can be combined with other tenants in ac



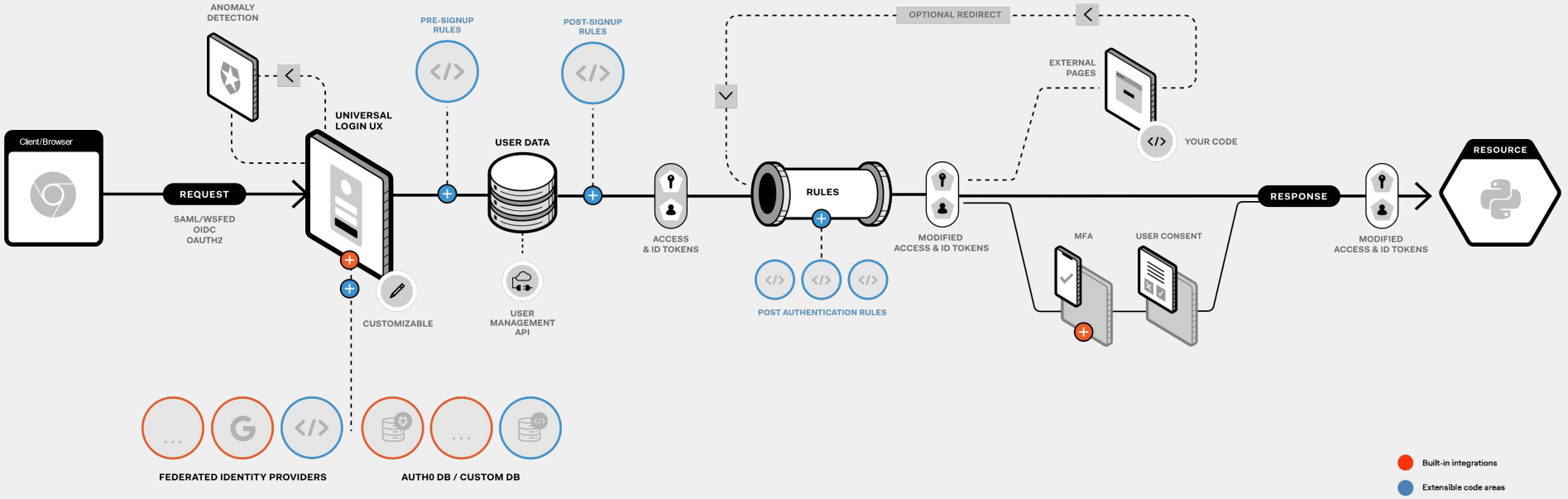
Auth0 Tenant

- Provides Single Sign On
- Can act as a “federation hub”
- Can translate between different protocols



The Auth0 Platform

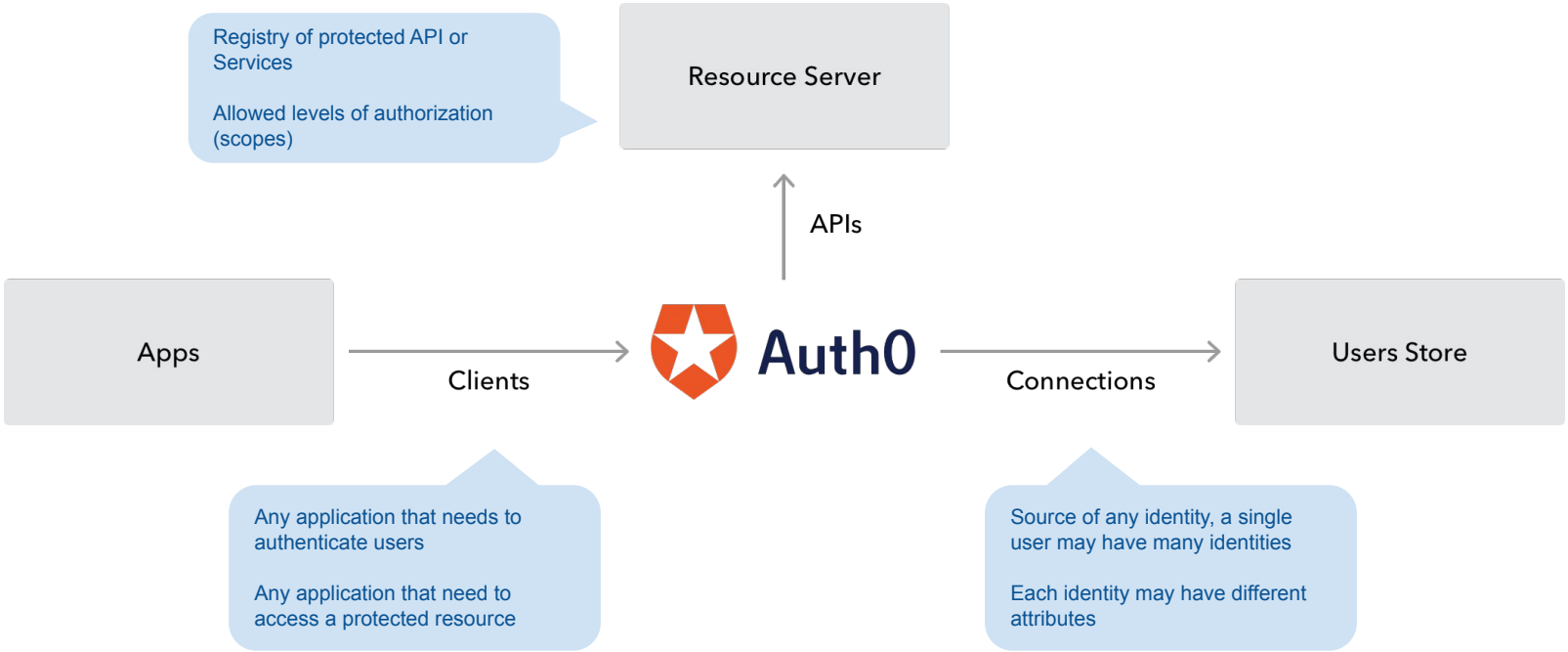
Simple, but highly **customizable** to your needs



Security

- Encryption, Password Hashing
- Attack prevention, Mitigation
  - > Built-in rate limiting
  - > Automated blocking
  - > Advanced DDoS
- Secure Infrastructure
  - > Hardened linux hosts with automatic security patching
  - > Segmented VPCs etc.
- SOC2, Privacy Shield, HIPAA, OpenID Certified

Auth0 Concepts



Clients

- Regular Web Applications
  - Traditional web app (with page refresh)
    - E.g. NodeJS w. Express, Java, ASP.NET
- Native
  - Mobile or Desktop, apps that run natively in a device
    - E.g. iOS, Android or WinForms application
- Single Page Web Application
  - JavaScript front-end apps typically using APIs
    - E.g. AngularJS, React
- Non Interactive Clients
  - CLI, Daemons or Services running on your backend

API Authorization

- OAuth 2.0 authorization framework specification
- Authorization for client-to-server and server-to-server applications
- Own applications
- Third-party applications
- Access to your APIs on behalf of the application itself

Connections

Connections are sources of users, they are categorized into

- Social Connections
- Database Connections
- Enterprise Connections
- Passwordless Connections

Samples and Platform specific SDKs

- Lots of Samples and Quickstarts
- Platform specific SDKs
- Comprehensive Documentation
- Client-side Applications
- Server-side Web Applications & APIs
- Mobile & Desktop Apps

Auth0 Training

Auth0 Quickstarts demo

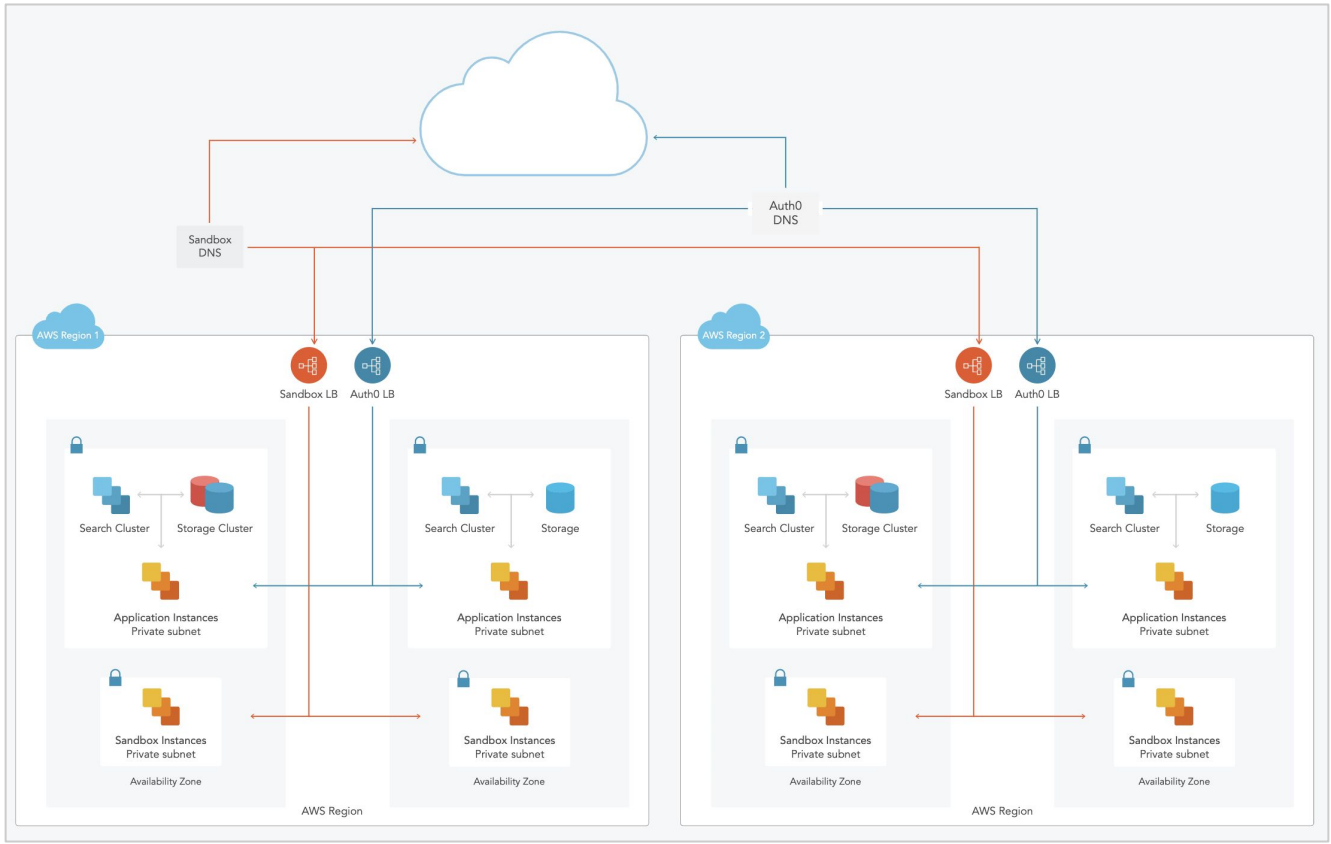
A quick tour of Auth0 Quickstarts documentation

Auth0 Training

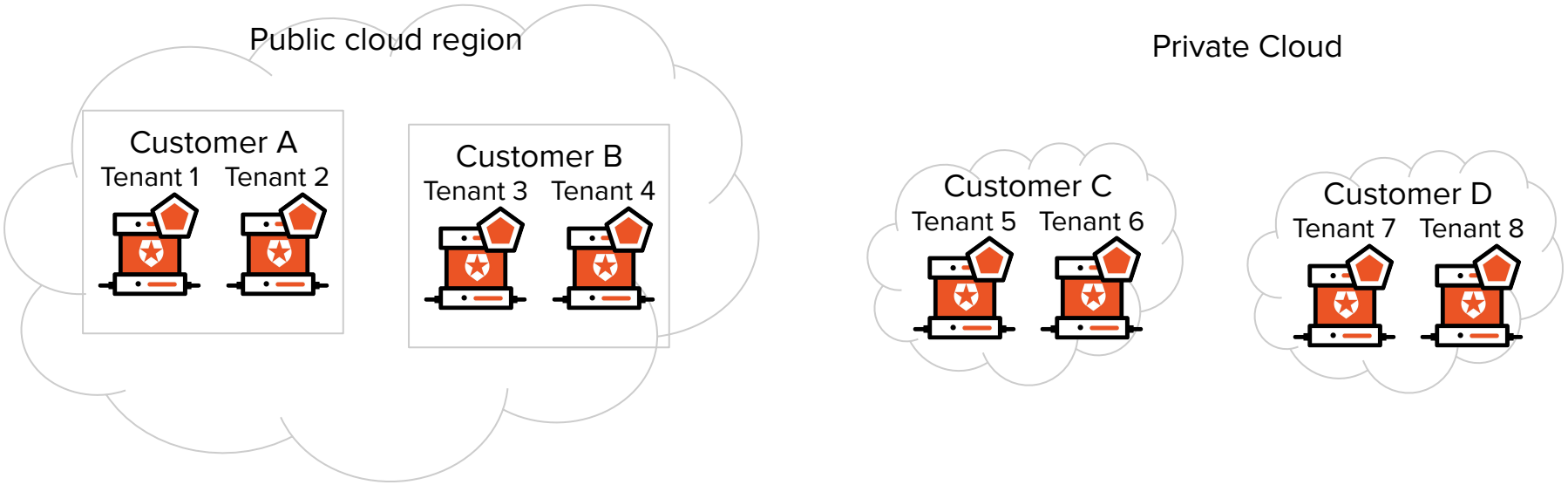
4.1. Deployment Models

Walkthrough on Auth0 deployment models and environments

High Availability

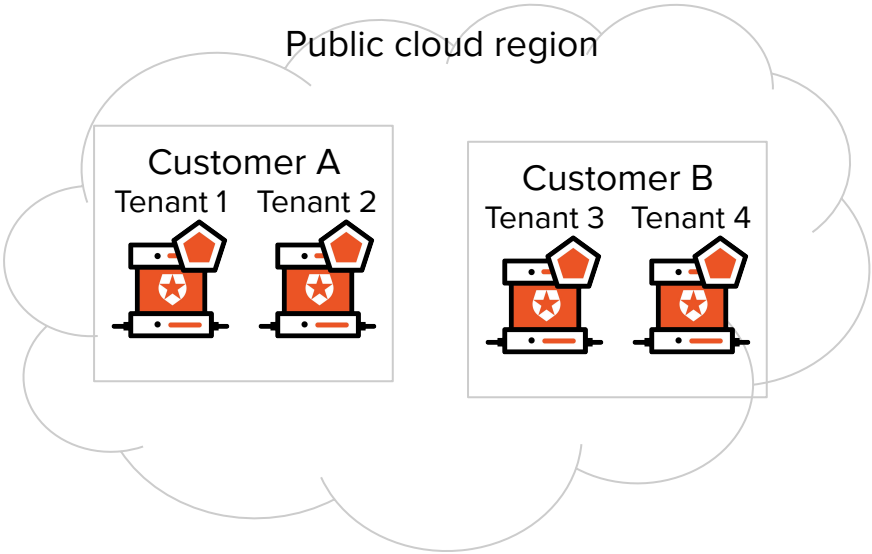


Deployment models



Public Cloud

- Auth0 Cloud
- Cloud Scale
- Primary Data Centers
  - North America
  - Europe
  - Australia
- Secondary in same continent
- High Availability and Real-time DR



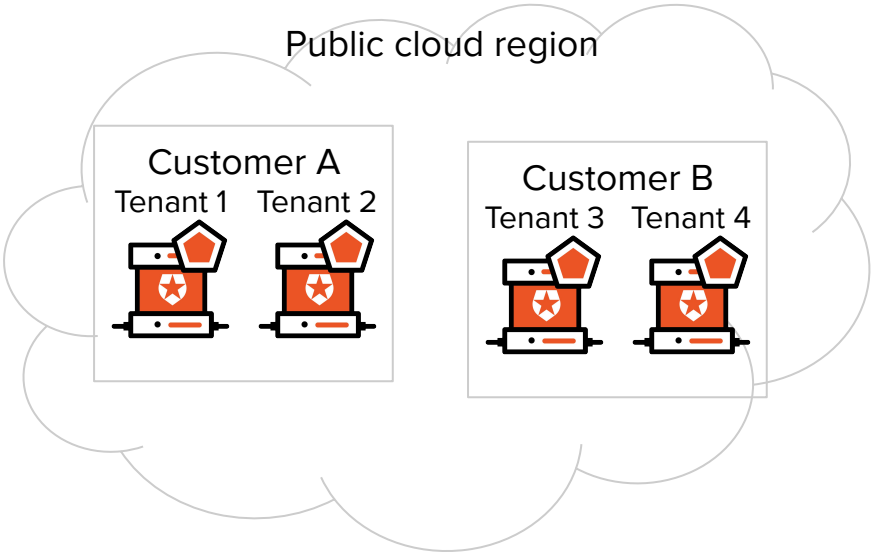
Public Cloud

The good:

- Fast updates
- Cheaper
- Automatic scaling

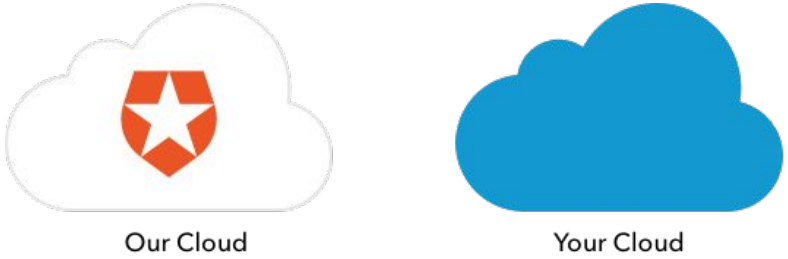
The bad:

- Strict rate limits
- May require escalation in exceptional situations



Private Cloud Options

- Basic Private Cloud
- Performance Private Cloud
- Performance Plus Private Cloud



	Public Cloud	Basic	Performance	Performance Plus
SLA	99.99%			
Tenancy	Multi	Single	Single	Single
RPS Ø	100	100	500	1,500
Data residency	X	✔	✔	✔
Upgrade flexibility	No	No	Yes	Yes
Dev environment	No	No	1	1

Ø Authentication API rate limit

Private Cloud Basic

- There is no dedicated development environment.
- PSaaS updates are released on a monthly cadence
  - Customer cannot schedule maintenance windows.
- Updates to the customers environment is performed by Auth0 outside of the customers core business hours
- During an upgrade, there is no expected downtime
- High capacity and high availability deployment models available
- 100 requests per second
- Bundled Services
  - Architecture Workshop
  - Health Check

Private Cloud Performance

- There is a dedicated development environment
- Customer may request permission to load test or penetration test (black box).
- Customer can work with Auth0 to schedule maintenance windows.
- 500 requests per second
- Includes the GEO High Availability addon
- Bundled Services
  - Architecture Workshop
  - Health Check
  - Go live
  - Load Test (1)

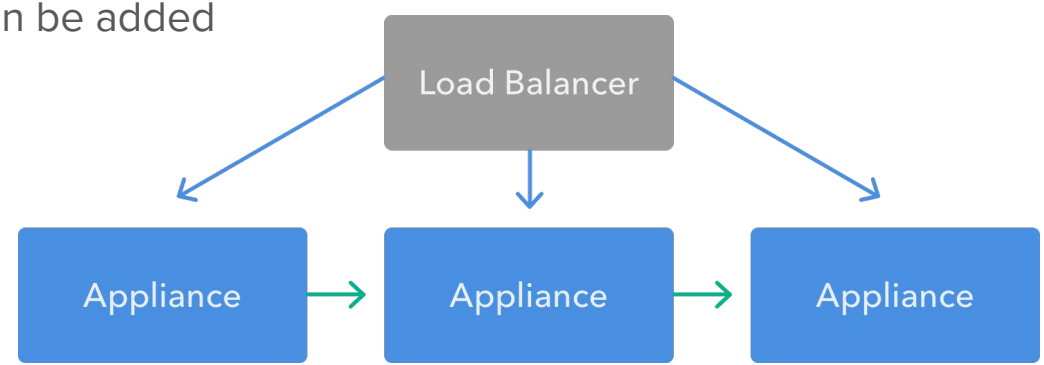
Private Cloud Performance Plus

- There is a dedicated development environment
- Customer may request permission to load test or penetration test (black box).
- Customer can work with Auth0 to schedule maintenance windows.
- 1,500 requests per second
- Includes the GEO High Availability addon
- Bundled Services
  - Architecture Workshop
  - Health Check
  - Go live
  - Load Test (2)

Managed Private Cloud - Deployment Options

High-Availability (HA)

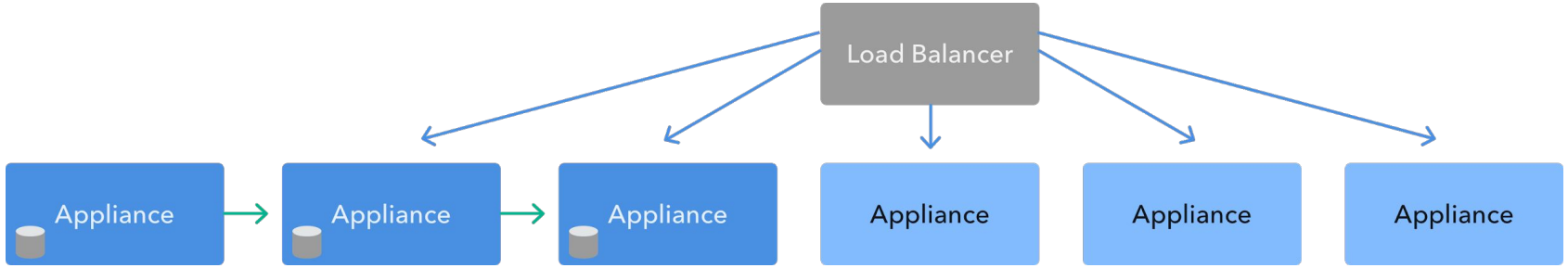
- Typically a three node redundant cluster
- Tolerates a single node outage
- Recommended for most production operations
- For High Capacity more nodes can be added



Managed Private Cloud - Deployment Options

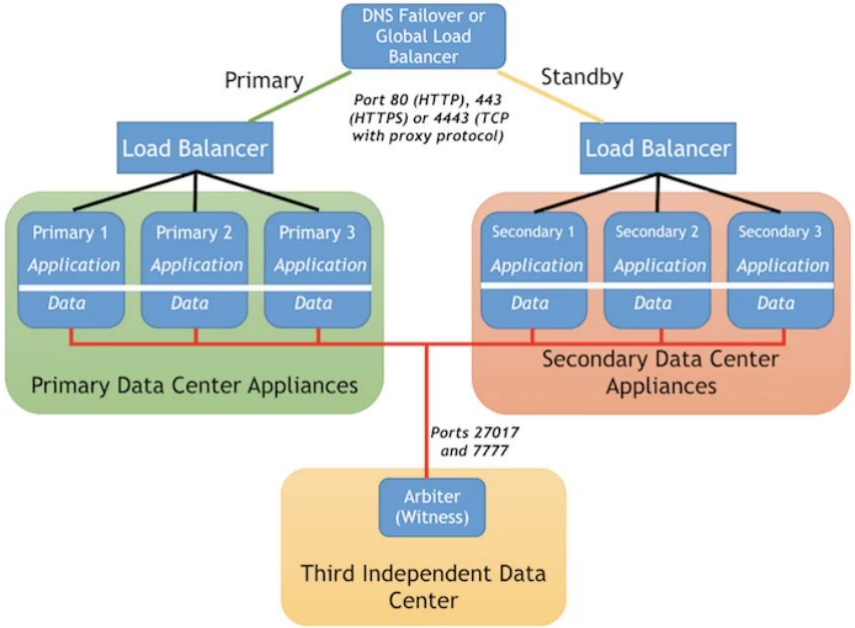
High-Capacity

- Highly scalable for the most demanding apps
- Can tolerate multiple node outages
- Recommended for large scale production operations (millions of users)



Private Cloud Performance and Performance Plus

Geo-High Availability



Private Cloud Performance/Performance Plus - Geo-HA

- All of the advantages of HA, but with added assurance of a standby in a different location, in case of disaster
- Requires arbiter in a 3rd location
- A **stretched cluster** that consists of the following pieces:
  - one geographically-aware global load balancer/DNS failover configuration;
  - one primary AWS region with three VMs;
  - one standby AWS region with three VMs;
  - one arbiter, a seventh instance that is located in a 3rd AWS data center.
- GeoHA is an **active/passive** model (not active/active)
- The AWS regions for the primary and secondary clusters should each have 3 availability zones.
- The recommended maximum round-trip latency between data centers should not exceed 100 ms.

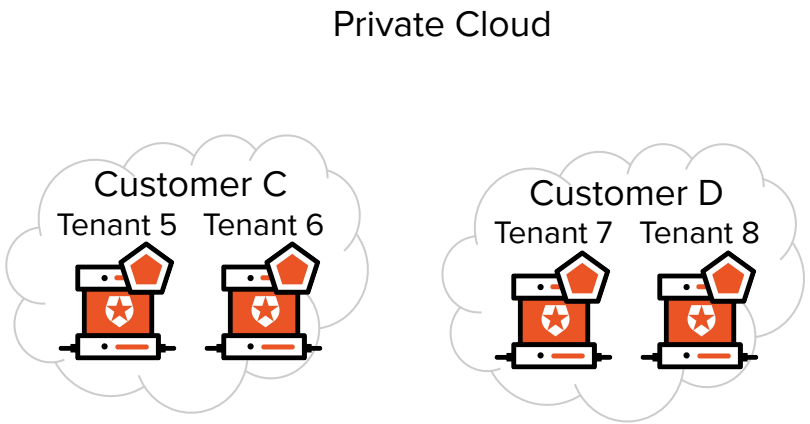
Private Cloud (aka PSaaS)

The good:

- Relaxed rate limits
- More control on updates
- Can use specific AWS regions
- Higher SLA

The bad:

- More expensive
- More difficult scaling



End of Chapter 4

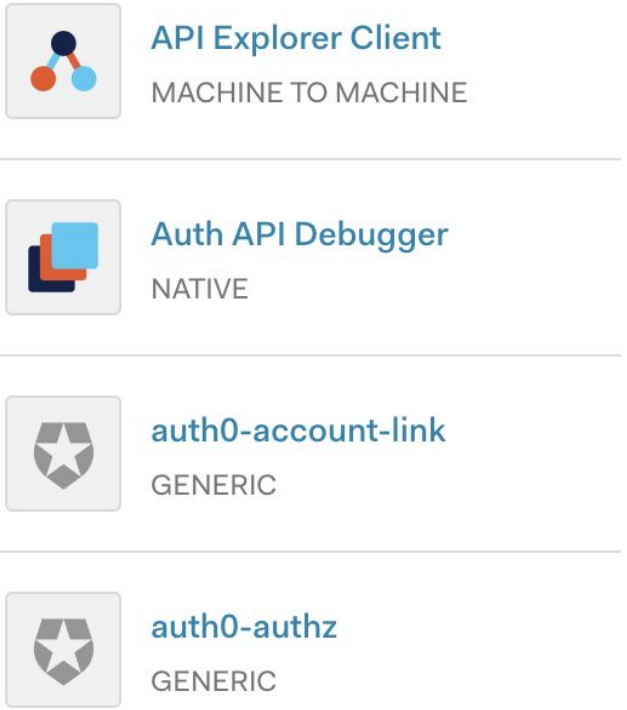
Auth0 Training

5. Features

Walkthrough of Auth0 features and capabilities

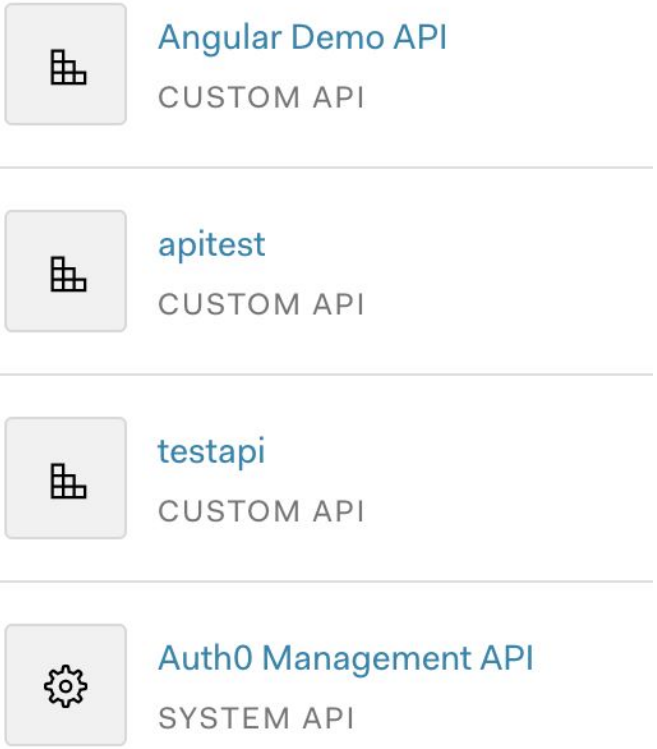
Applications

- Aka Clients
- Have unique Client ID
- Public / Confidential
- 1st Party / 3rd Party
- Interactive / non-interactive
- SAML and WS-Fed can be enabled separately as needed



APIs

- Aka Resource Server
- Have unique Audience
- Can have scopes
- Scope has limited permission modeling ability



Auth0 Training

5.1. Connections

Review of sources of user accounts

Connections

Connections represent the user identity “source”:

- DB connections - either local (stored in Auth0) or external
- Federated from 3rd Party IdP
- May require manual Home Realm Discovery (HRD) implementation.

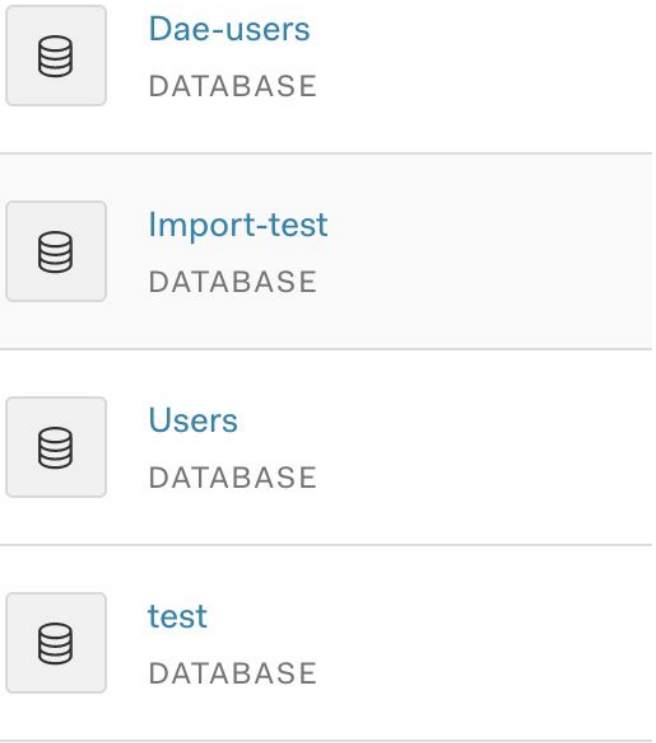
DB Connection

Auth0 Tenant

3rd Party IdP

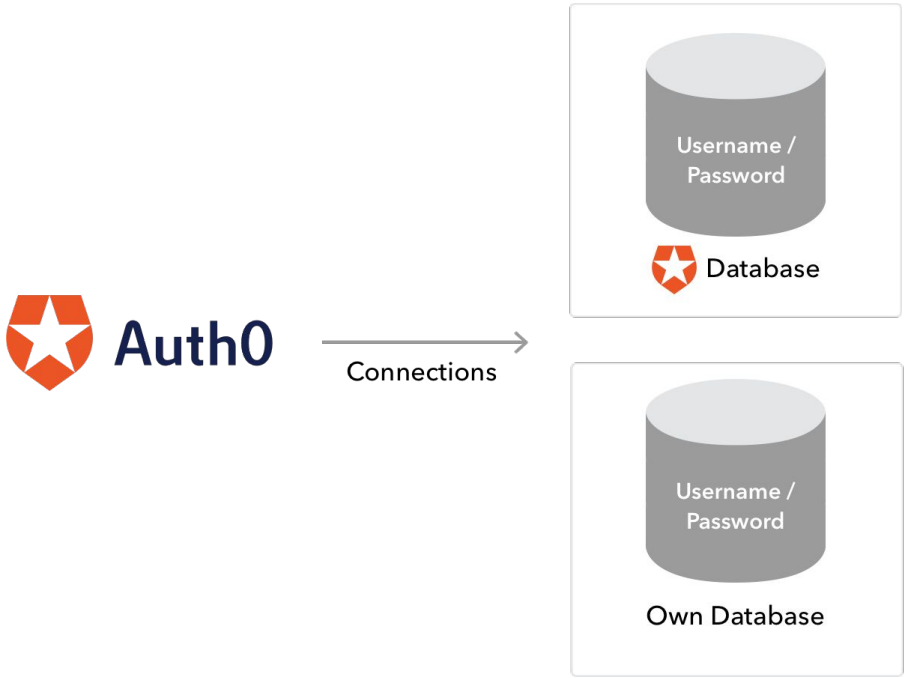
DB Connections

- Main identity storage
- Credentials captured by Auth0
- Can be customised with scripts
- Can have “import mode” for automatic user migration



Database Connections

- Auth0 user store
- Own Database
  - > Mongo
  - > SQL Server
  - > MySQL
  - > PostgreSQL
  - > Any API based service
  - > Many more



Database Connections

- Password Options
  - Password Policies, predefined set of password strength levels
  - Password History, prevents reuse of X passwords used before
  - Password Dictionary
    - Do not allow passwords that are part of the password dictionary
    - Includes by default 10000 most common passwords
    - Add own entries (e.g. YourCompany123)
  - Personal Data
    - Do not allow passwords that contain any part of the user's personal data.
- Requires Username
  - In addition to the email address requires a username
  - User can login with email address or username

Database Connections

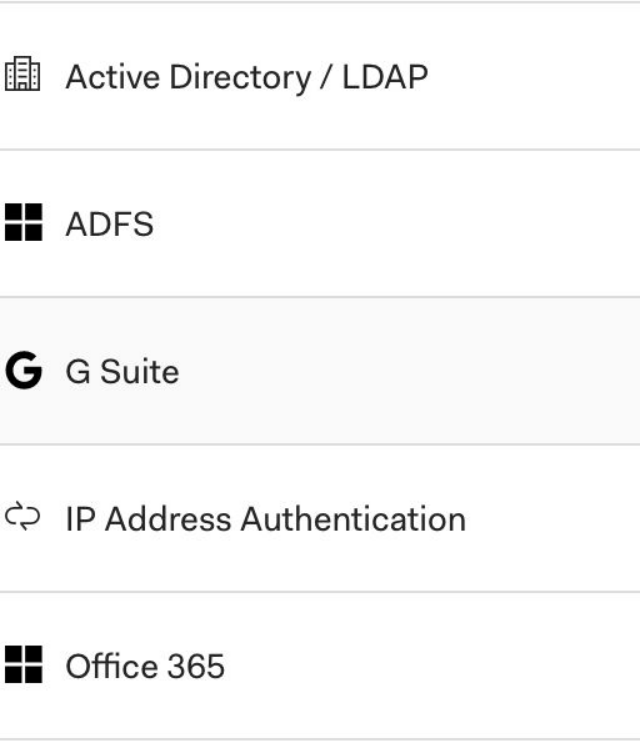
- Database Action Scripts
  - > Import Users to Auth0 is enabled; migrate active users to the Auth0 user store
    - **Login** - executed each time a user attempts to login
    - **Get User** - executed when the user signs up, and at other times
  - > Import Users to Auth0 is disabled; use your datastore
    - Login
    - Get User
    - **Create** - executed when the user signs up
    - **Verify** - executed when a user verifies their email
    - **Change Password** - executed when changing password
    - **Delete** - executed when a user is deleted

Do not disable Import Users after migration is complete. Modify the scripts to do nothing.

Enterprise Connections

- ADFS
- Azure AD
- Generic SAML
- Generic OIDC
- On-prem AD
- ... and others

Auth0 captures credentials only for on-prem AD Connections.

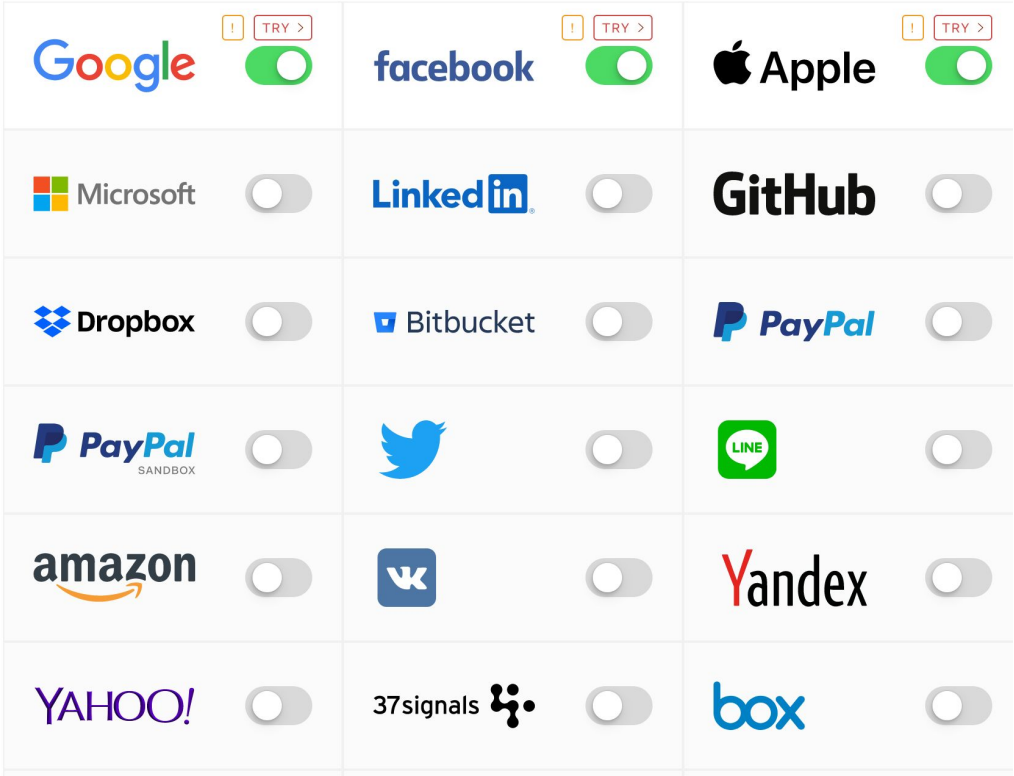


Social Connections

- Google
- Facebook
- Sign In with Apple
- ... and many others

Auth0 never captures social user's credentials.

Can integrate with custom OAuth2 providers



Passwordless Connections

- No need to remember a password
- Improves user experience on Mobile Apps
- Types
  - Email
    - One time code
    - Link
  - SMS

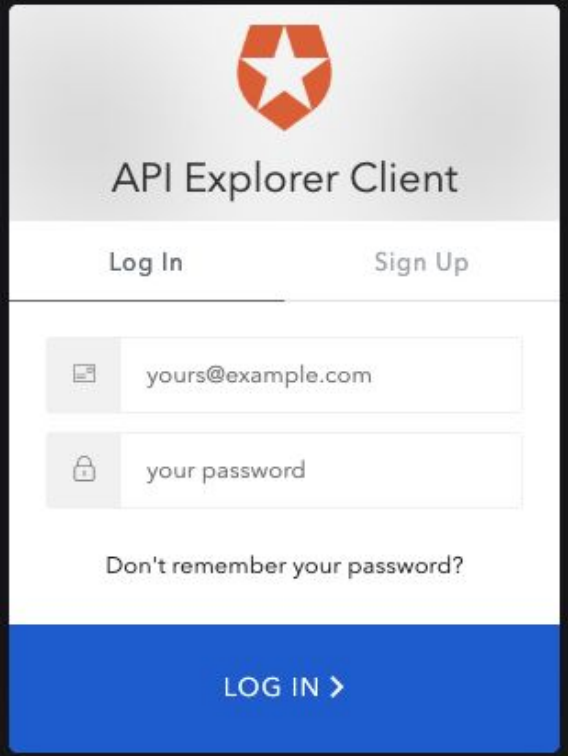
Auth0 Training

5.2. Universal Login

Auth0 hosted login pages

Universal Login

- Also known as “Hosted Login Pages”
- Only one set of pages per tenant.
- Available pages: Login/Registration; Password reset; MFA
- Generic error page
- Full control on HTML



Why Universal Login?

Universal Login	Embedded Login
Recommended best practice since phishing and man-in-the-middle attacks are more likely with embedded. The IETF best practices for OAuth2 and native apps specifically state embedded logins “MUST NOT” be used.	Resource Owner Password credentials were only ever recommended when there is a “high degree of trust between the resource owner and the client … and when other authorization grant types are not available”
Least privilege - With this model your application never sees user credentials.	Application has access to credentials which is unnecessary since the Authorization Server is verifying them.
Login screen is focused on authentication and requires minimal libraries providing a smaller attack vector.	Login is handled inside a larger application that likely uses many more libraries. The application itself and libraries create a larger attack vector.
SSO can be done from web and mobile apps as all logging in is done at one central domain. MFA is easier and follows best practices.	SSO is not possible in mobile apps. MFA requires more work. Authentication with cryptographic credentials, and authentication processes that require multiple steps (WebCrypto, WebAuthn) can be hard or impossible
Only one login UI needs to be created and maintained.	A login UI will need to be created and maintained for each app.

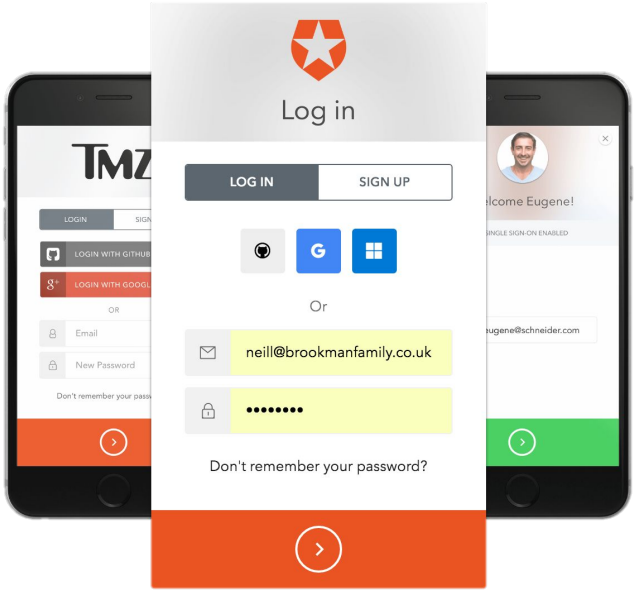
Universal Login is Customizable

- Colors and icons can be customized easily in universal login.
- If you want more in depth customization there are two options:
  - Use Lock and configure it's appearance and behavior to your liking. Colors, text, fields, and more can be configured.
  - Create a custom UI. While it should be focused on authentication only you can create a login form that looks exactly as you wish. It would leverage the Auth0.js library to perform the logins.

Lock

Lock is a configuration-driven, hosted login form option for desktop, tablet and mobile devices.

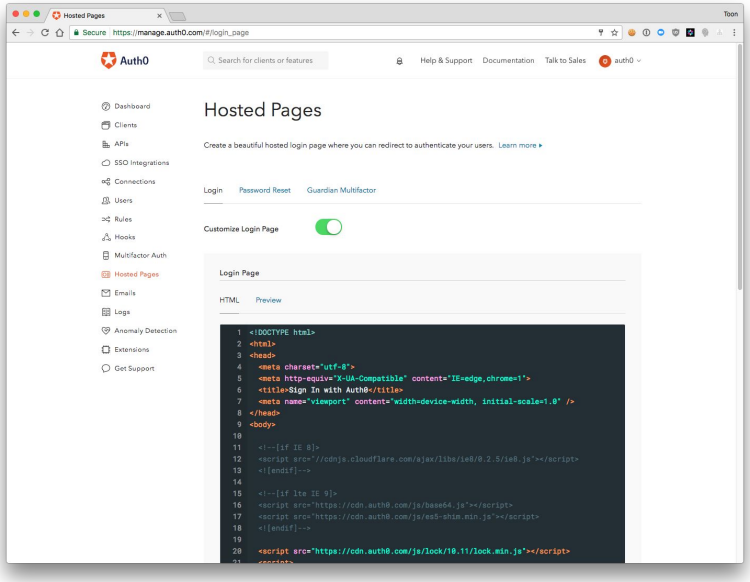
Lock is mobile ready and dynamically display properly on phone and tablets.



Other Hosted Pages

In addition to login, these experiences can also be customized:

- Sign up
- Password Reset
- Guardian multifactor authentication



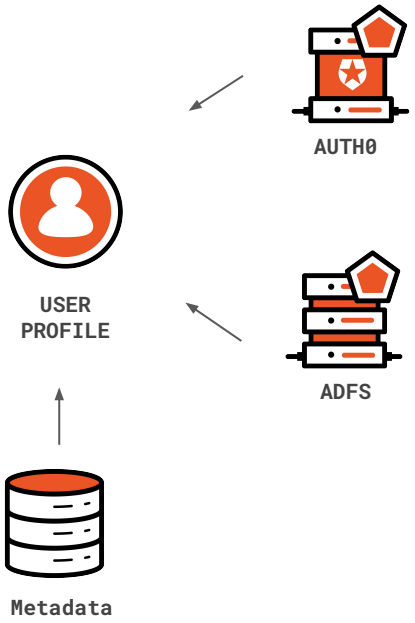
Auth0 Training

5.3. User Profiles

The structure of a user profile in Auth0

Users

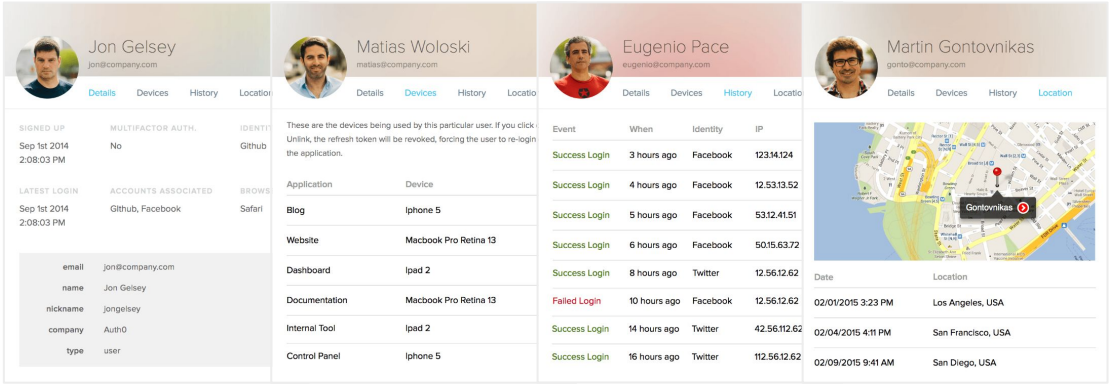
- User profile - can be cached from an 3rd party IdP
- Identity - can be multiple (“Account Linking”)
- Metadata - to store user data needed during login process



Profiles

Access detailed profile information for each of your users, including authentication details, devices, and login history.

- Only accessible by dashboard admins
- Accessible for non-admins using the Delegated Administration Dashboard Extension



Profiles

- Progressive Profiling
  - > Rather than asking your users to fill out extensive registration forms, you can use progressive profiling, a technique to collect user information as users interact with your system.
- Auth0 Normalized User Profile
  - > Since every identity provider provides a different set of information about a user, Auth0 normalizes common profile properties in the User Profile.

Profiles - Account Linking

- Auth0 supports the linking of user accounts from various identity providers, allowing a user to authenticate from any of their accounts and still be recognized by your app and associated with the same user profile.
    - User Initiated Account Linking
    - Account Linking from Server Side Code
    - Using the Authentication API

Profiles - metadata

- `user_metadata`
  - Stores user attributes (such as user preferences) that do not impact a user's core functionality;
- `app_metadata`
  - Stores information (such as a user's support plan, security roles, or access control groups) that can impact a user's core functionality, such as how an application functions or what the user can access.
- An authenticated user can modify data in their profile's `user_metadata`, but not in their `app_metadata`.
- Updateable via Lock, Management API or Rules
- Storage is limited so store only authentication / authorization information

Auth0 Training

5.4. MFA

Multi-Factor Authentication features

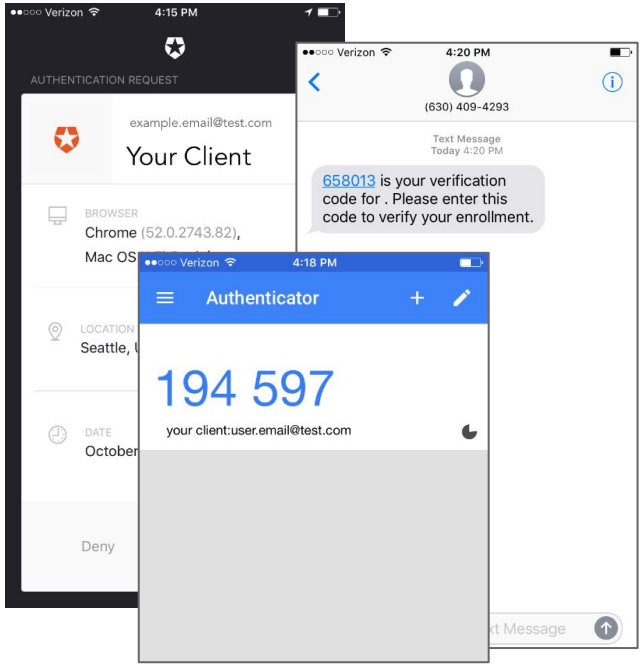
Multi-factor Authentication

Verify a user's identity by requiring them to present more than one piece of identifying information.

Provides an additional layer of security, decreasing the likelihood of unauthorized access.

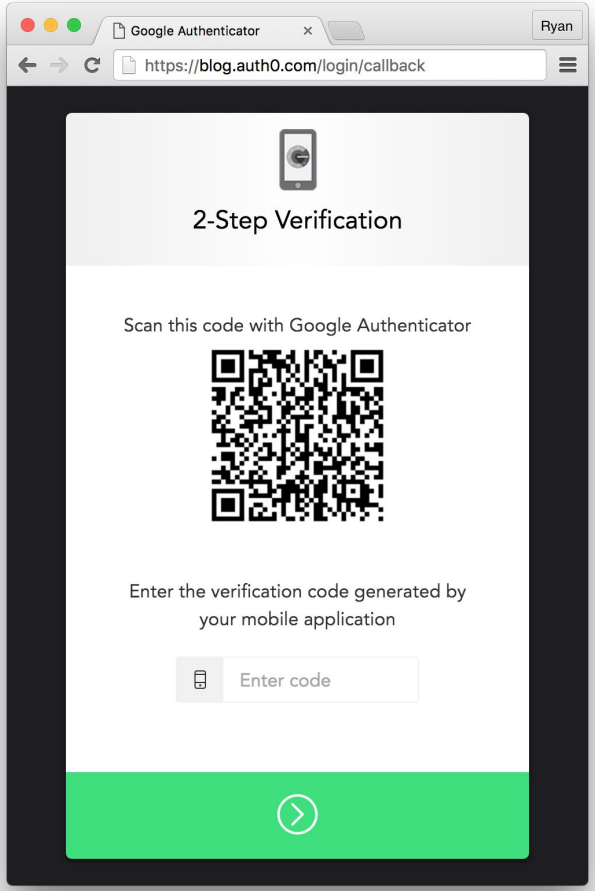
- Push notifications - Auth0 Guardian
- SMS - Six-digit code
- One-time password - Google Authenticator or Duo
- Custom providers (e.g. Yubikey, RSA)

Multi-channel OAuth2 MFA Draft API available 2018Q1



Multi-factor Authentication

- OTP
- Push (Guardian app)
- SMS (Twilio only)
- Duo security
- Email
- Available via Universal Login or API
- Can be triggered from rules (for Step Up)



Auth0 Guardian

Guardian is Auth0's multifactor authentication (MFA) application that provides a simple, safe way for you to implement MFA.

- Push notifications are sent to the app for the user to approve
- iOS or Android app, or integrate Guardian in to your existing apps using our SDK
- SMS
- One-time password

Auth0 Training

5.5. RBAC

Role-Based Access Control

RBAC

- Allows to define and assign roles for users
- Uses API scopes as permissions
- Uses an “additive’ model - permissions are calculated as “union” of all roles
- Authorization Extension will be deprecated - use Authorization Core instead

RBAC vs Scopes

- Scopes are part of OAuth2 spec - mean “what user allows for the client to access”
- Meant for 3rd party applications
- They may not be granular enough to model permissions

RBAC and Applications

- Roles and permissions show “what the user can do” (actions)
- Roles do not model which specific entities a user or an application can access (no ownership)
- The IdP only provides the information - the access control must be enforced at API level

RBAC Limitations

Entity	Limit
Roles per tenant	1000
Scopes per API (enforced by api2)	1000
Roles per user (through direct assignment)	50
Permissions per user (through direct assignment)	1000
Permissions per role	1000

Auth0 Training

5.6. Emails

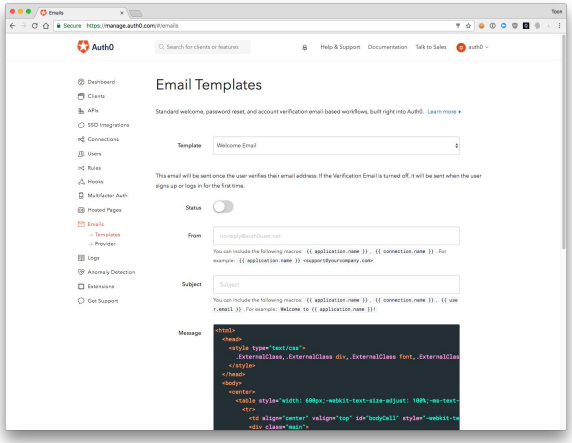
Emails sent by Auth0

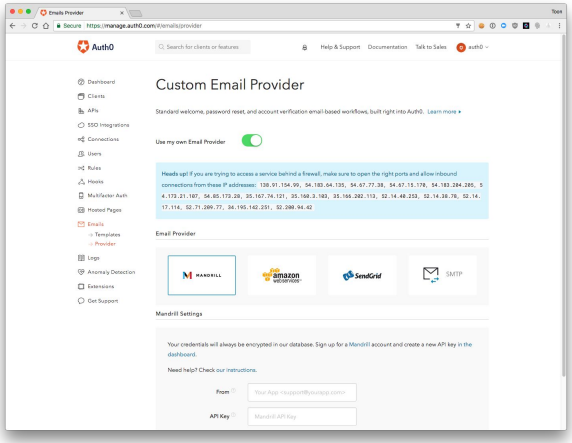
Emails

Auth0 provides built-in email services to easily communicate with your users. This includes verification emails, welcome emails, change password emails, and blocked account emails.

Providers

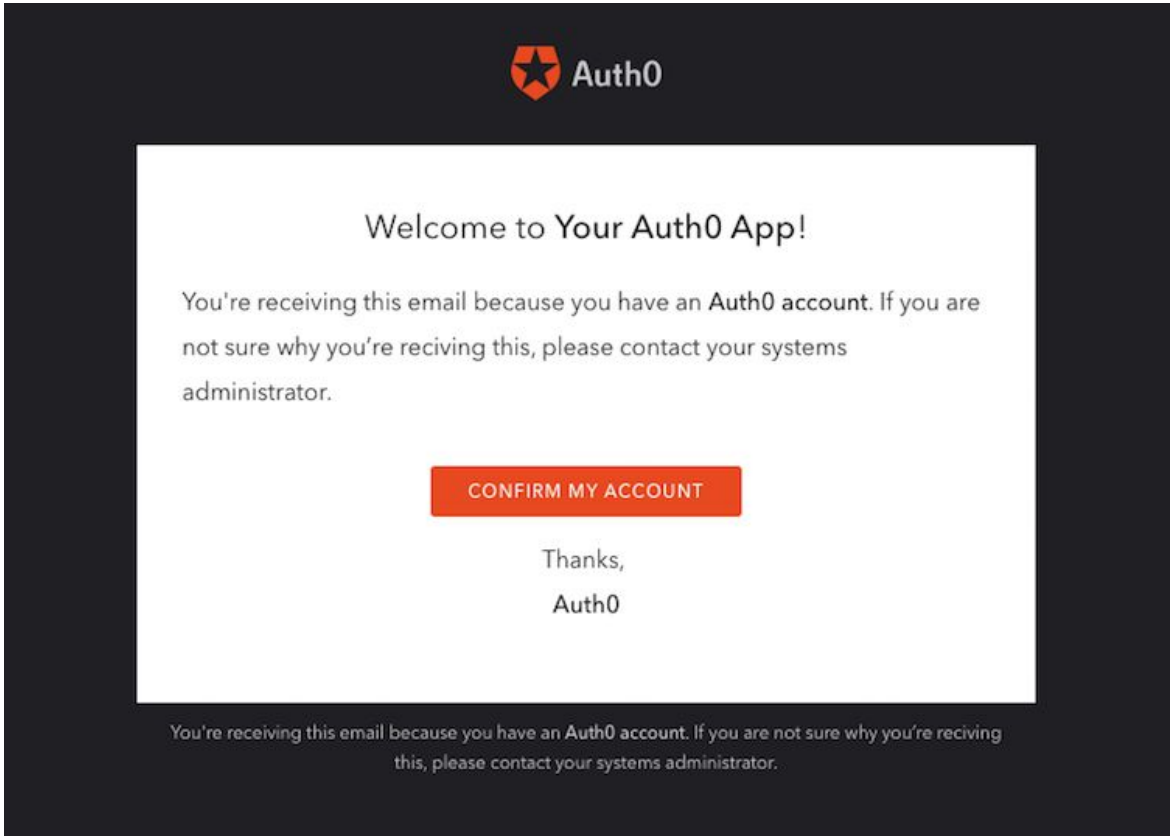
- Amazon SES
- Mandrill
- SendGrid
- SparkPost
- Mailgun
- BYO Custom SMTP





Email templates

- Only one set of email templates per tenant.
- Uses “liquid” syntax, can have if/else logic.
- Must use own email provider.



Emails

Emails:

- Email verification
- Welcome email
- Password reset
- Blocked account
- Etc

Configuration options:

- Email templates - every aspect of the email can be customized
- Email provider - a custom one must be used for production and for customized emails

Customization options:

A. Use default email functionality
   a. Easiest to use and least flexible
   b. Uses Auth0 email templates (can be customized)

B. Use Auth0 “tickets” with custom email
   a. Still uses Auth0 email verification and password reset
   b. Requires custom email handling

Auth0 Training

5.7. Anomaly Detection

Built-in protection mechanisms

Brute-force Protection

- Always generates a log entry
- Notifications and blocks can be enabled separately
- Make sure to whitelist automated test IPs (up to 50 CIDR entries)

Brute-force Protection

Limit the amount of signups and failed logins from a suspicious IP address.

This will perform the following actions:

Send email notifications.

Block suspicious IP addresses.

Breached-password Detection

- Always generates a log entry
- Triggered on login and password change/reset
- Works only when Auth0 captures the password
- Actions can be turned on individually

Breached-password Detection

Detects login attempts with credentials that have been known to be breached.

This will perform the following actions:

Send an e-mail to affected tenant.

Block login attempts.

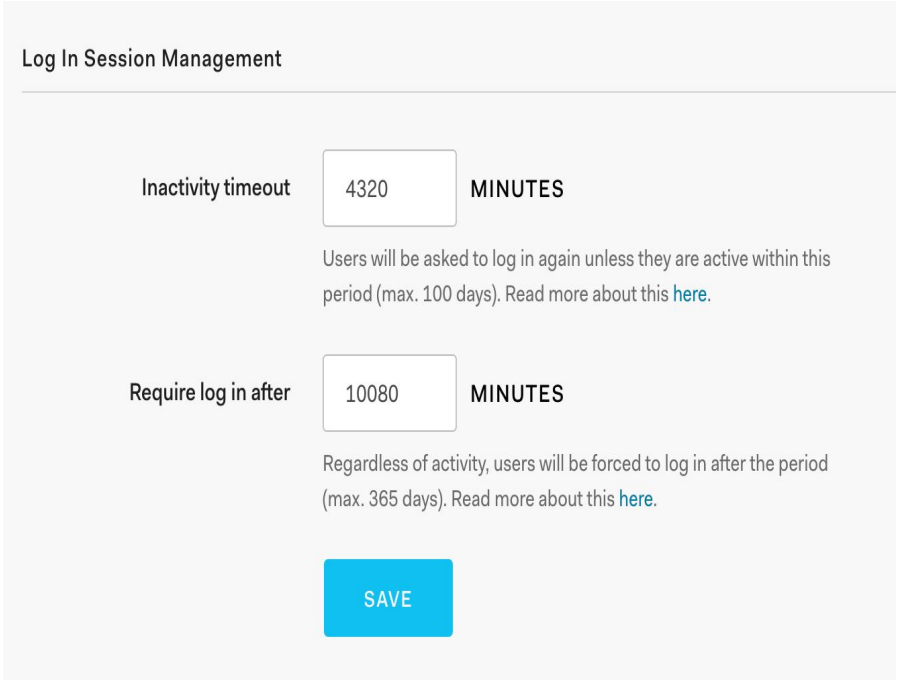
Notify administrators.

5.8. Other features

A few more notable features

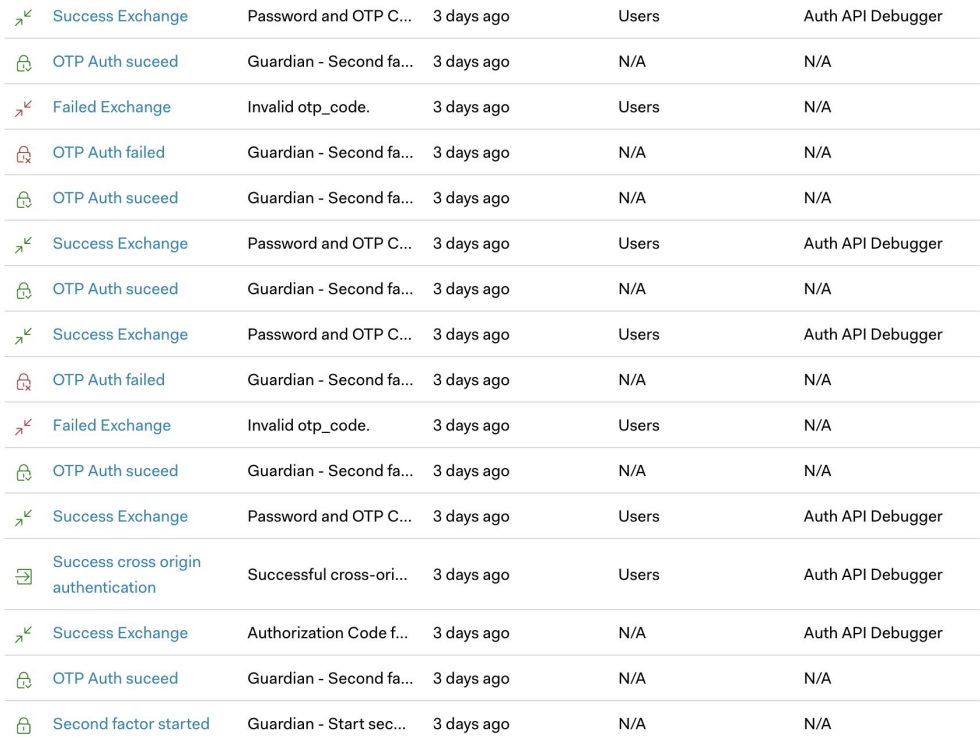
Long lived sessions

- Auth0 session cookie TTL at Tenant level
- Single Logout
  - Front channel only: responsibility of application
- In SSO mode, Auth0 acts as IdP to validate session, irrespective of original connection provider



Logs

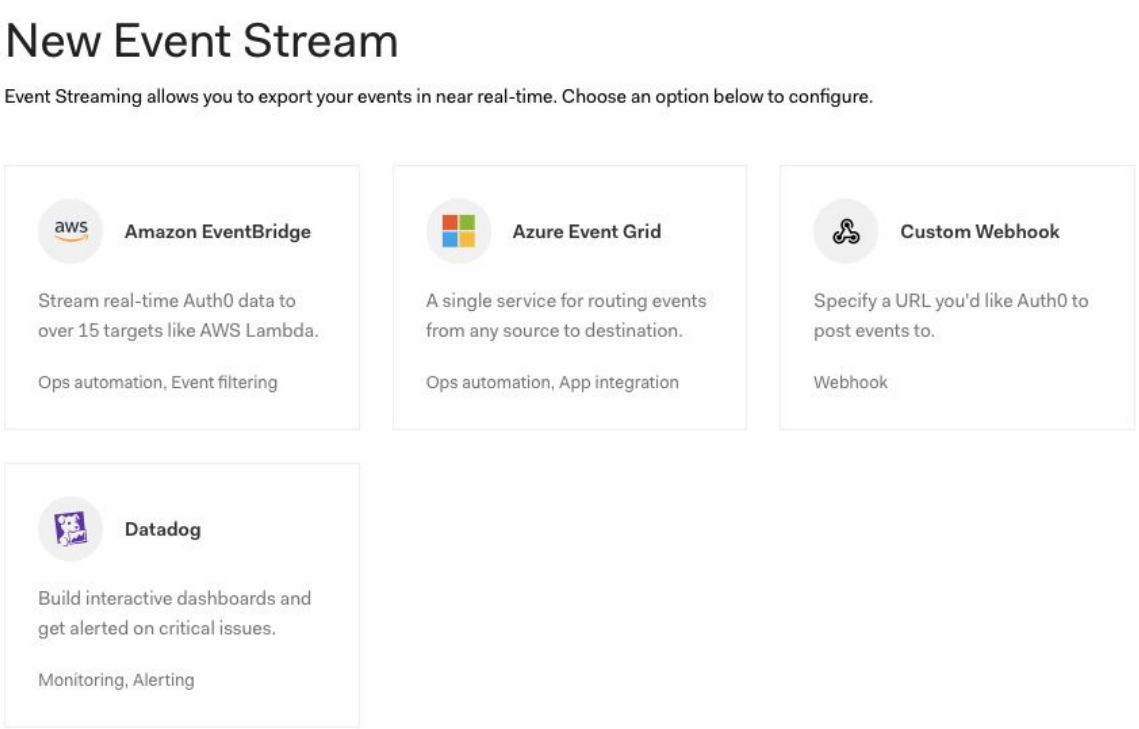
- Always export logs - Auth0 storage limited to 30 days
- First step in diagnosing issues
- Can provide a lot of data for mining/reporting
- Script Management API to export in an unsupported system



Success Exchange	Password and OTP C...	3 days ago	Users	Auth API Debugger
OTP Auth suceed	Guardian - Second fa...	3 days ago	N/A	N/A
Failed Exchange	Invalid otp_code.	3 days ago	Users	N/A
OTP Auth failed	Guardian - Second fa...	3 days ago	N/A	N/A
OTP Auth suceed	Guardian - Second fa...	3 days ago	N/A	N/A
Success Exchange	Password and OTP C...	3 days ago	Users	Auth API Debugger
OTP Auth suceed	Guardian - Second fa...	3 days ago	N/A	N/A
Success Exchange	Password and OTP C...	3 days ago	Users	Auth API Debugger
OTP Auth failed	Guardian - Second fa...	3 days ago	N/A	N/A
Failed Exchange	Invalid otp_code.	3 days ago	Users	N/A
OTP Auth suceed	Guardian - Second fa...	3 days ago	N/A	N/A
Success Exchange	Password and OTP C...	3 days ago	Users	Auth API Debugger
Success cross origin authentication	Successful cross-ori...	3 days ago	Users	Auth API Debugger
Success Exchange	Authorization Code f...	3 days ago	N/A	Auth API Debugger
OTP Auth suceed	Guardian - Second fa...	3 days ago	N/A	N/A
Second factor started	Guardian - Start sec...	3 days ago	N/A	N/A

Log Streams

- High-performance log exporting mechanism.
- Logs are exported as soon as they are registered in the system.
- Use for:
  - Log export
  - Custom webhooks



Auth0 Training

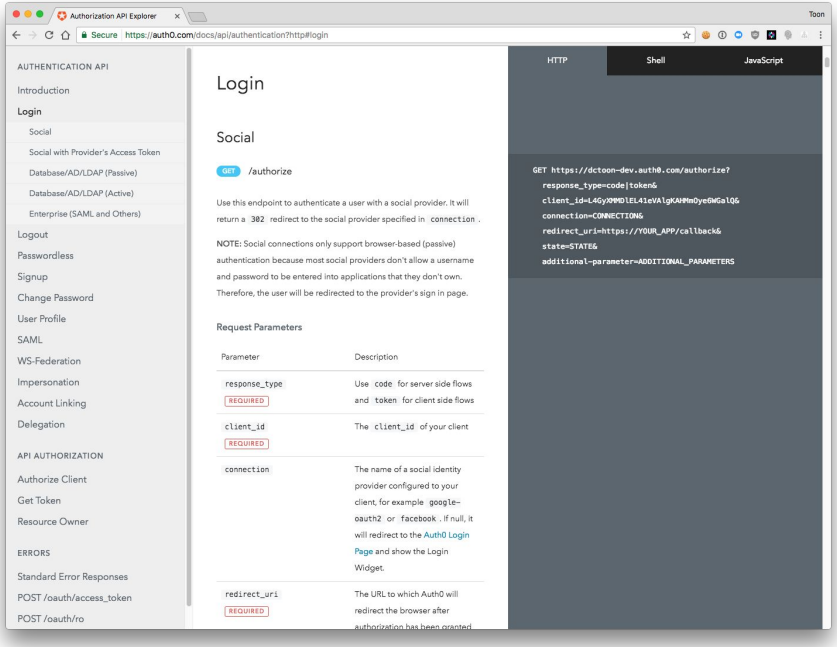
5.9. Authentication API

Public-facing API

Authentication API

The Authentication API exposes the identity functionality of Auth0, as well as the supported identity protocols like OpenID Connect, OAuth 2.0, and SAML.

Most users consume this API through our Quickstarts, the Auth0.js library or the Lock widget. However, if you are building all of your authentication UI manually you will have to interact with this API directly.



Authentication API

“Public” API

- Users interact with it during login, logout, registration, etc.
- Can be used programmatically - for example to send a password reset email.
- Rate limits are individual - usually per user or per IP/user combination.
- Has built-in anomaly detection (if enabled in tenant settings).

Authentication API

- RESTful JSON API
- Public - used by Lock and Auth0.js
- All supported protocols are available - OIDC, SAML, WS-Fed, etc
- Has wrapper libraries for various frameworks

The Authentication API can be called programmatically to trigger various workflows - for example user registration, passwordless or MFA association.

```
/authorize
/v2/logout
/passwordless/start
/dbconnections/signup
/dbconnections/change_password
/userinfo
/mfa/challenge
...
```

/authorize Parameters

- **client_id** - Identifier for the application the user is logging into. Defined at Auth Server
- **client_secret** - passed by confidential clients to prove they are making the request
- **response_type** - determines the authorization processing flow to be used, including what parameters are returned from the endpoints used. For example a value of code indicates authorization code flow and will result in an authorization code being returned
- **scope** - space delimited list of strings, each corresponding to an item the application is requesting access too. openid scopes control access to information about a user. Per the oauth spec other scopes are defined at the api level and restrict what operations can be performed against that application.
- **redirect_uri** - The URL to which the auth server will redirect the browser after authorization has been granted for the user. Must be registered at auth server
- **connection** - allows you to specify the connection the user should login against.

/authorize Parameters

- **audience** - specifies the api you are requesting an access token for
- **state** - An arbitrary value included in the request and saved. This same value will be in the callback received by the application after login. The application compares against the stored value and rejects the callback if there is no match. Prevents CSRF attacks and can also be used to store state across redirects.
- **prompt** - A value of login will force the user to enter credentials even if there is a valid session. A value of none will result in a succesful login only if a session already exists. If the user would have to enter credentials the login fails.
- **max_age** - specifies a maximum allowable time in seconds since the user last entered their credentials. If this time is exceeded the user will be forced to enter them again even if they have a valid session. If used the resulting id token will include an auth_time claim indicating when the user authenticated.

/authorize Parameters

- **response_mode** - defines how parameters should be returned to the callback endpoint. Supported values are query, fragment and form_post.
- **login_hint** - Can be used as a hint to the Authorization Server about the login identifier the End-User might use to log in. For example if you know what email a user will log in with you can pass it with login_hint and have the login screen prepopulate that.
- **acr_values** - When the value “http://schemas.openid.net/pape/policies/2007/06/multi-factor” is passed the application is requesting that MFA is required on the login attempt. When used the application should verify the id_token has an “amr” claim with value “mfa” . This confirms mfa was actually performed.

Auth0 Training

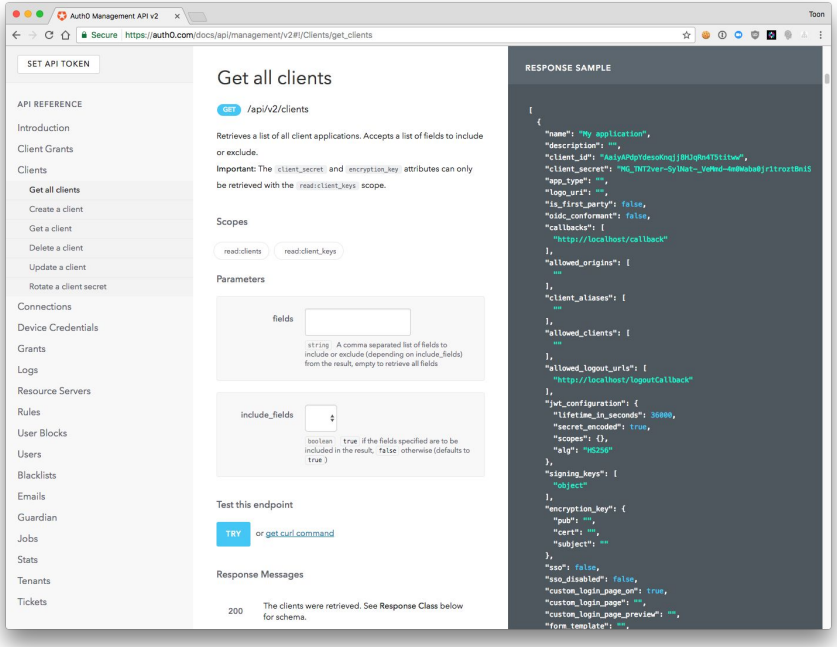
5.10. Management API

Backend API

Management API

The Management API provides full programmatic access and automation of the Auth0 platform.

You can use the Explorer to browse the API and interact with it dynamically with your tenant's data! Or use our Postman Collections.



Management API

“Internal” API

- Meant to be used from backend for administration purposes.
- Requires an JWT Access Token obtained through Machine-to-Machine authentication.
- One rate limit for all - 50 rps and 1000 rpm sustained for PROD.
- Must not be exposed to Public Clients.
- All calls are subject to rate limits - even from rules.

Management API

- RESTful JSON API
- Should be accessed only from backend (Confidential Clients)
- Uses Client Credentials to get Access Tokens
- Rate limited - 50 requests per second/ 1000 per minute
- Loads exceeding that will be throttled
- Has wrapper libraries for various frameworks

Overall, the Management API has more functionality than is exposed in the Dashboard - if you can't find something in the Dashboard - there may be a hidden "gem" in the API.

All extensions are webtasks built on top of Management API.

```
/api/v2
  /client-grants
  /clients
  /connections
  /custom-domains
  /device-credentials
  /grants
  /logs
  /resource-servers
  /rules
  /rules-configs
  /user-blocks
  /users
  /users-by-email
  /blacklists/tokens
  /email-templates
  ...
```

Management API Rate Limits

- Management API rate limits are tenant specific
  - Production tenants - 50 requests per second or 1000 per minute is sustainable
  - Non production tenants - 10 requests per second or 120 per minute is sustainable
- Responses will include Rate limit related headers
  - X-RateLimit-Limit: The maximum number of requests available in the current time frame.
  - X-RateLimit-Remaining: The number of remaining requests in the current time frame.
  - X-RateLimit-Reset: A UNIX timestamp of the expected time when the rate limit will reset.
- You should always throttle your code by checking the rate limit headers and backing off when remaining nears 0.
- If you do exceed the allowed limits you will receive a 429 TooManyRequests response (including the 3 rate limit headers).

Management API Tokens

- Lifespan of tokens for the management api can be configured per tenant.
- Allowed scopes can be configured per application that calls the management API. Each application should only be granted the scopes it requires.
    - The management api explorer lists the scopes required for each endpoint.
- Access tokens should be cached and reused, many libraries handle this automatically.
- Client Credentials hook can be used to modify the access token returned to applications. Custom claims can be added and scopes modified.

Auth0 Training

Auth0 Dashboard demo

A quick tour of Auth0 Tenant Management dashboard

End of Chapter 5

Auth0 Training

6. Extensibility

Detailed walkthrough of Auth0 extensibility

Webtasks

- “Function as a Service” built for Auth0.
- Runs on Node.js v12.
- Supports many NPM modules
- Limit of 20 seconds for total execution, 2 seconds for blocked event loop.
- 100kb of code max.
- Abusing tenants will be temporarily quarantined webtask container.

```
const tools = require('auth0-extension-express-tools');

const expressApp = require('./server');
const config = require('./server/lib/config');
const logger = require('./server/lib/logger');

const createServer = tools.createServer((config, storage) => {
 logger.info('Starting Auth0 Logging Extension - Version:',
process.env.CLIENT_VERSION);
 return expressApp(config, storage);
});

module.exports = (context, req, res) => {
 const publicUrl = (req.x_wt && req.x_wt.ectx &&
req.x_wt.ectx.PUBLIC_WT_URL) || false;
 if (!publicUrl) {
   config.setValue('PUBLIC_WT_URL', tools.urlHelpers.getWebtaskUrl(req));
 }

 createServer(context, req, res);
};
```

Rules

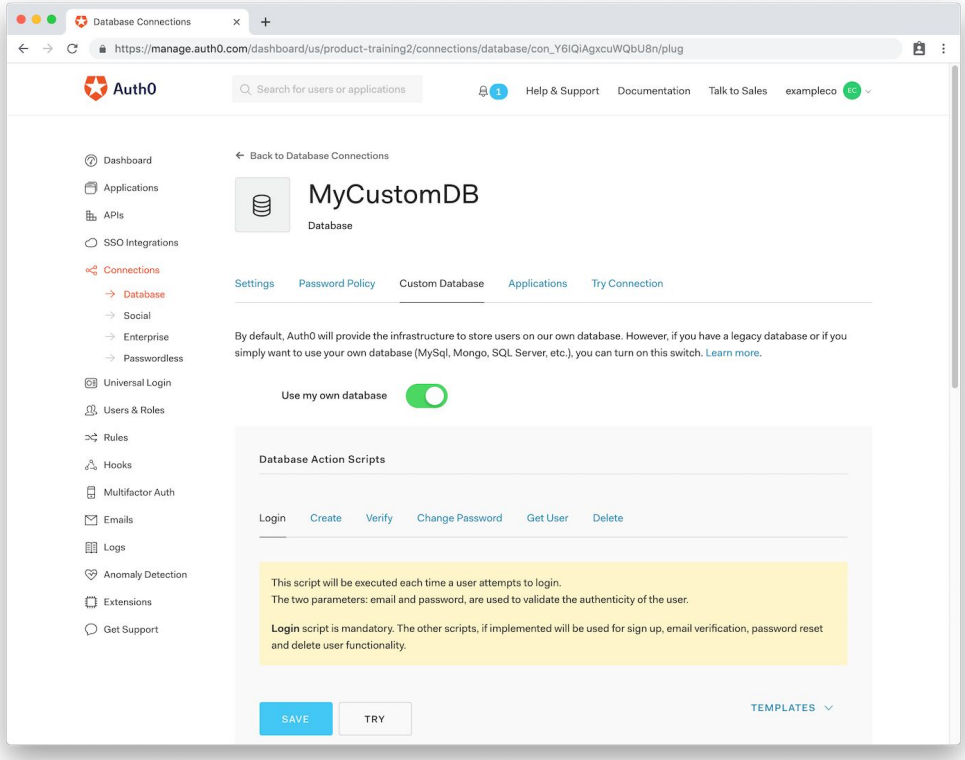
- Run after user successful credential verification.
- Can inject data in JWT.
- Can customise the SAML mapping.
- Can initiate MFA.
- Can interrupt the login.
- ...
- Most used extensibility feature in Auth0.

```
function (user, context, callback) {
 if (context.connection === 'company.com') {
   context.idToken['https://example.com/vip'] = true;
 }


 callback(null, user, context);
}
```

Custom DB scripts

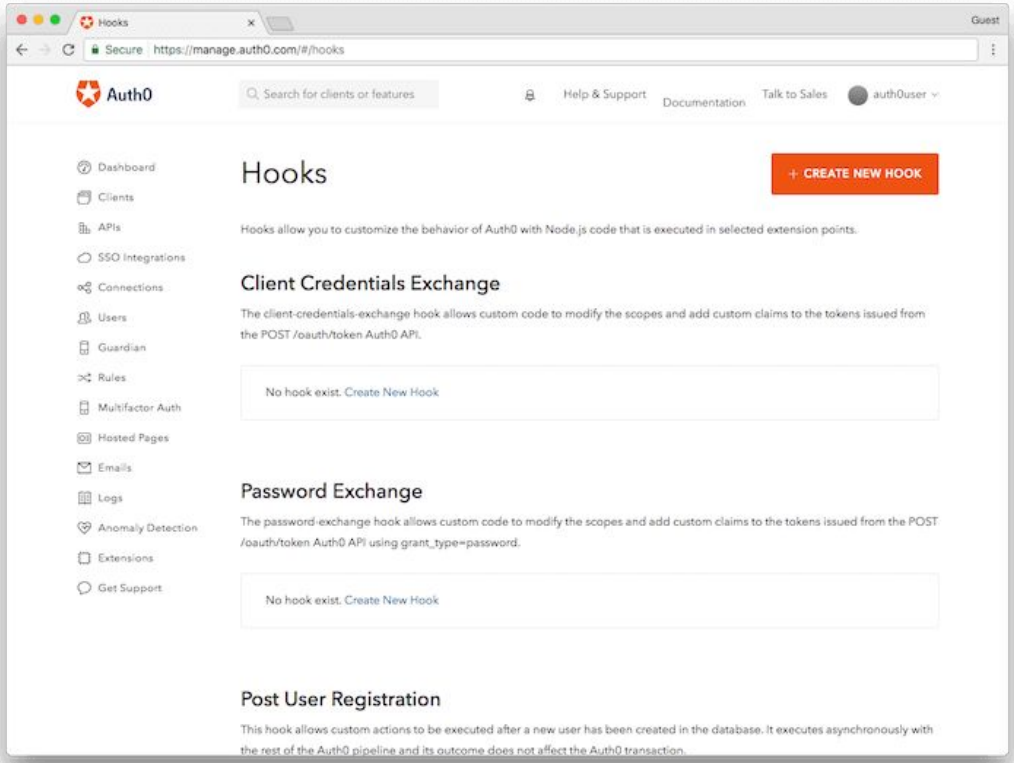
- Allows to use own user storage or import users to Auth0.
- Based on webtasks, full customisation available.
- No automatic Home Realm Discovery when multiple DB connections are used.



Hooks

Similar to Rules but run at different stages:

- Pre-registration - before user profile is created.
- Post-registration - after user profile is created. Is asynchronous.
- Client Credentials exchange - for M2M authentication.



Extensions

- Pre-built apps running as webtasks and using Auth0 APIs.
- Can run as apps or cron jobs.
- Open-source with MIT license.

Securing external calls

Any scripts can make external HTTP calls. The following can be implemented to secure these calls:

- MUST use TLS (https).
- May use API keys or M2M Access Tokens.
- May use client certificate
- May deploy firewall rules to whitelist source IPs from Auth0.

Make sure to follow general Node.js guidelines - correct callback/promise logic flow, etc.

Logging

- Cannot create custom tenant logs yet.
- Can call any external log providers to post custom messages.
- “console.log(...)” can be used for debugging.

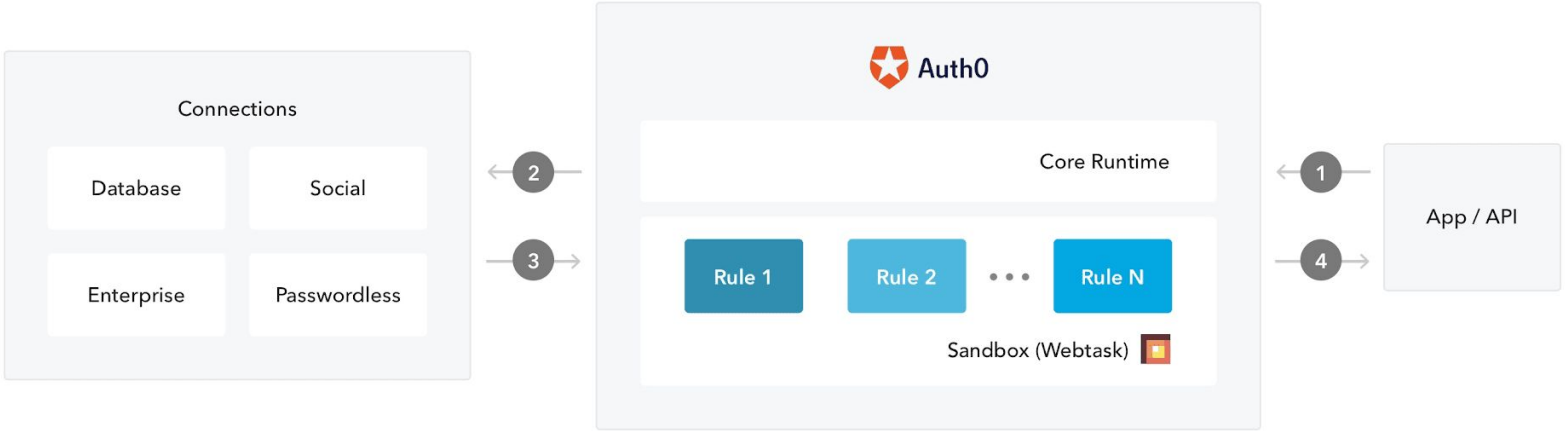
Auth0 Training

6.1. Rules

Most used extensibility point for login process customisation

Rules

Rules are JavaScript functions that are executed in Auth0 as part of the transaction every time a user authenticates to your application. Rules allow you to easily customize and extend Auth0's capabilities. Rules can be chained together for modular coding and can be turned on and off individually.



Rules

- Written in JavaScript
- Full power of the ECMAScript 5 language
- For security reasons, the Rules code runs in a sandbox based on Webtask
- Isolated from other customers
- +1000 NPM modules available to use
- Module not there? request what you need

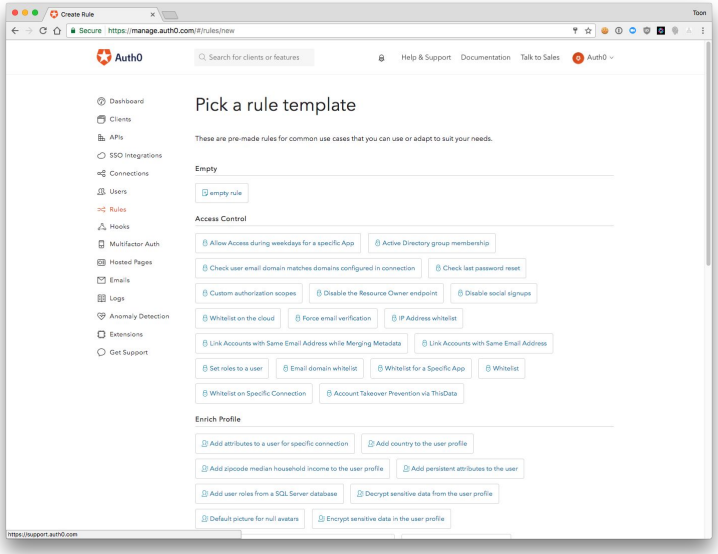
Possibilities

- **Profile enrichment**: query for information on the user from a database/API, and add it to the user profile object. **Use Carefully**
- Create **authorization** rules based on complex logic
- **Normalize attributes** from different providers beyond what is provided by Auth0
- Reuse information from existing **databases** or **APIs** for migration scenarios
- Keep a **whitelist of users** and **deny access** based on email
- **Notify other systems** through an API when a login happens in real-time
- Enable counters or persist other information (e.g. storing user data information in profile)
- Enable **multi-factor authentication**, based on context (e.g. last login, IP address of the user, location, etc.)

Templates

Auth0 provides pre-made rules for common use cases that you can use or adapt to suit your needs.

- Active Directory Group Membership
- Set roles to a user
- Force email verification
- SAML Attributes mapping
- Change SAML configuration
- Send email
- ...



Rule example

- Store environment-specific values in configuration.
- NO hardcoded secrets in rule code.
- Full user profile available in the “user” parameter.
- “context” contains info about client, environment and protocol.

```
function (user, context, callback) {
 // short-circuit if the user signed up already or is using a refresh token
 if (context.stats.loginsCount > 1 || context.protocol === 'oauth2-refresh-token')
{
   return callback(null, user, context);
 }

 // get your slack's hook url from: https://slack.com/services/10525858050
 const SLACK_HOOK = configuration.SLACK_HOOK_URL;

 const slack = require('slack-notify')(SLACK_HOOK);
 const message = 'New User: ' + (user.name || user.email) + ' (' + user.email +
')';
 const channel = '#some_channel';

 slack.success({
   text: message,
   channel: channel
 });

 // don’t wait for the Slack API call to finish, return right away (the request
will continue on the sandbox)`
 callback(null, user, context);
 }
```

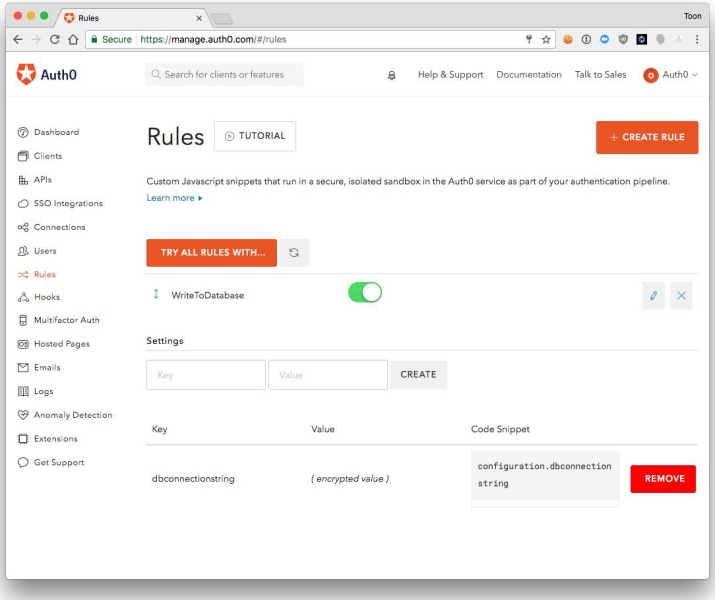
Rule Function Syntax

- `function(user, context, callback)`
- `user`
  - > User object as it comes from the identity provider
- `context`
  - > Object containing contextual information of the current authentication transaction, such as user's IP address, application, location.
- `callback`
  - > Function to send back the potentially modified user and context objects back to Auth0 (or an error).

```
function (user, context, callback) {
  user.hello = 'world';
  console.log('===> set "hello" for ' + user.name);
  callback(null, user, context);
}
```

Secrets

- Key/Secret pair
- Accessible via global `configuration` object
- `configuration` object available to all rules.
- Secret value is encrypted, and can only be retrieved in Rule execution context.



Cache expensive resources

- The `global` object allows storing expensive resources that will survive individual execution
- The `global` object is shared between all rules and authentication transactions

  Notice that the code sandbox in which Rules run on, can be recycled at any time. So your code must always check `global` to contain what you expect.

```
...

//If the db object is there, use it.
if (global.db){
  return query(global.db, callback);
}

//If not, get the db (mongodb in this case)
mongo('mongodb://user:pass@mymongoserver.com/my-db',  function (db){
  global.db = db;
  return query(db, callback);
});

//Do the actual work
function query(db, cb){
  //Do something with db
  ...
});

...
```

Redirect Rule 1

Rules can also be used to programmatically redirect users before an authentication transaction is complete, allowing the implementation of custom authentication flows which require input on behalf of the user, such as:

- Requiring users to provide additional verification when logging in from unknown locations
- E-mail verification
- GDPR acceptance

```
function (user, context, callback) {
    context.redirect = {
        url: "https://example.com/foo"
    };
    return callback(null, user, context);
}
```

Redirect Rule 2

- User is logging in, a rule is setting `context.redirect` with *redirect url*
- Authentication transaction is interrupted, user is redirected to *redirect url* with a `state`.
  - > State is used to restore user context when resuming authentication transaction
  - > E.g. https://example.com/foo?state=abc123
  - > User action takes place, e.g. change password
- Resume authentication transaction by redirecting to Auth0 endpoint with `state`
  - > https://{your-tenant-name}.auth0.com/continue?state=THE_ORIGINAL_STATE
- Rules execute again, distinguish between user-initiated login and resumed login flow by; `context.protocol === "redirect-callback"`

```
function (user, context, callback) {
    if (context.protocol === "redirect-callback") {
        // User was redirected to the /continue endpoint
```

```
function (user, context, callback) {
    context.redirect = {
        url: "https://example.com/foo"
    };
    return callback(null, user, context);
}
```

Redirect Rule 3

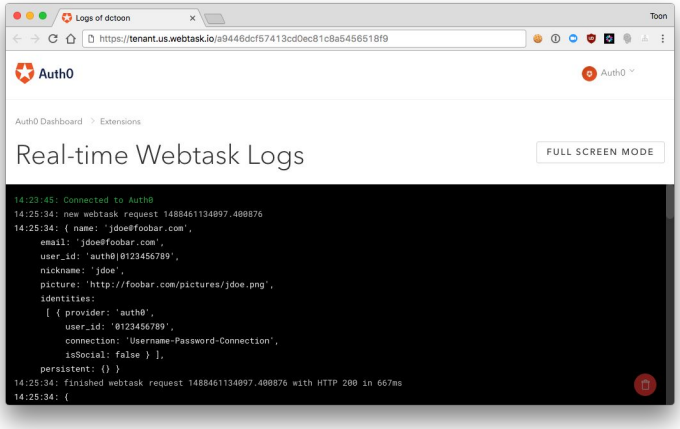
Redirect rules won't work for the *Resource Owner endpoint*, the *Password exchange* or the *Refresh Token exchange*. You can detect these cases by checking `context.protocol`:

- For Password exchange: `context.protocol === 'oauth2-password'`
- For Refresh Token exchange: `context.protocol === 'oauth2-refresh-token'`
- For Resource Owner logins: `context.protocol === 'oauth2-resource-owner'`

```
function (user, context, callback) {
    context.redirect = {
        url: "https://example.com/foo"
    };
    return callback(null, user, context);
}
```

Debugging

- Try This Rule
- Debug Rule
  - > Specify User and Context
- Real-time Webtask Logs Extension
- Authentication API Debugger extension



Source Control

- Use your Source Control system for all your rules and database connection scripts
- Have them automatically deployed to Auth0 each time you push to your repository via
  - Source Control Extensions
    - Github Deployments
    - Gitlab Deployments
    - Visual Studio Team Services Deployments
    - Bitbucket Deployments
    - Convention based folder structure
      https://github.com/auth0-samples/github-source-control-integration
  - Auth0-deploy-cli https://github.com/auth0/auth0-deploy-cli
- Via custom code that interacts with the Auth0 Management API

Code sharing

All enabled rules are “glued” together before execution - they share the same memory space. This means that any changes to user or context parameters will be visible to subsequent rules.

```
function rule1(user, context, callback) {
  // set up some re-useable function
  context.my_sdk.my_func = function () { ... };
}

function rule2(user, context, callback) {
  // invoke the re-useable function
  context.my_sdk.my_func();
}
```

Testing

In the current implementation a rule must be loaded as a raw text fragment and executed.
The “vm” NPM package can be used for this - the rule results can be verified for the expected state.

```
  const vm = require('vm');
  const fs = require('fs');
  var user = {
      "email":       "jdoe@foobar.com",
      "user_id":     "auth0|0123456789"
    };
  var context = {
      "clientID":            "123456789"
    };

  global.configuration = { DEBUG: true };

  vm.runInThisContext(
      "(()=>{return " + fs.readFileSync('./rules/Normalized Profile
Claims.js') + " })();", {
        filename: 'Normalized Profile Claims.js',
        displayErrors: true
      }
  )(
    user,
    context,
    function callback() { console.log("Complete"); }
   );
```

6.2. Custom DB scripts

Use an existing storage or migrate users across as they log in

Execution + error reporting

Custom Database Connections can run in 2 modes:

1. Custom DB - with scripts Login, GetUser, Create, Delete, Verify, ChangePassword.
2. Custom DB with Automatic Migration - with scripts Login, GetUser.

In case of Automatic Migration, the “GetUser” script must return the full user profile to account for Password Reset requests for non-migrated users.

Errors are not returned to the UI yet from all the custom DB scripts.

Configuration and deployment

Configuration and deployments use same approach as Rules using the “/api/v2/connections” endpoint.
It is possible to use another Auth0 tenant as a database by calling it via ROPG.

```
function login (username, password, callback){
    request({
        url: 'https://' + configuration.Domain + '/oauth/token',
        method: 'POST',
        json: {
            grant_type: "http://auth0.com/oauth/grant-type/password-realm",
            realm : configuration.Connection,
            scope: 'openid profile email', // todo: add name to scope
            client_id: configuration.Client_ID,
            client_secret: configuration.Client_Secret,
            username: username,
            password: password
        },
        headers: {'content-type' : 'application/json'}
    }, function(error, response, body){
        if(error) {
            callback(error);
        } else {
            if(response.statusCode !== 200){
                callback();
            } else {
                var profile = {};
                var openidProfile = jwt.decode(body.id_token);
                // ... profile mapping omitted
                callback(null, profile);
            }
        }
    });
 }
```

Code sharing and testing

Code sharing across Custom DB scripts is not supported out-of-the-box. If this is a requirement - it is recommended to store the code separately and “compile” the scripts during deployment by assembling them in the CI/CD pipeline from multiple fragments. Script testing can use the same approach as rules - load the code as raw text and execute.

Auth0 Training

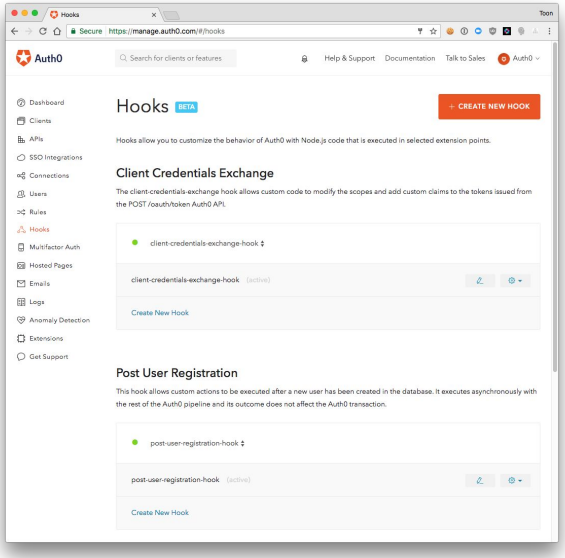
6.3. Hooks

Additional extensibility points

Hooks

Hooks allow you to customize the behavior of Auth0 using JavaScript (Node.js) code that is executed against extensibility points;

- Credentials Exchange
- Pre-User Registration
- Post-User Registration
- Post change password
- Send Phone Message



Execution + error reporting

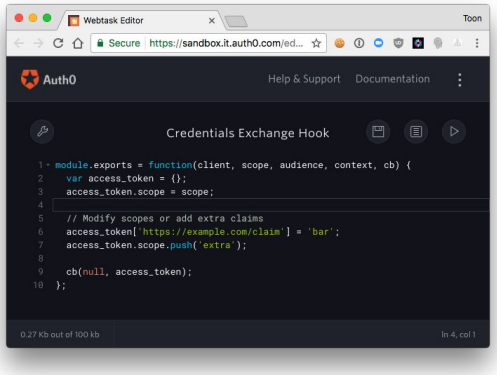
Additional extensibility endpoints:

- Pre-registration - on error it is possible to return back to registration page.
- Post-registration (async) - errors are logged, but cannot interrupt the process.
- Client Credentials Exchange (M2M) - can interrupt the process.
- Post change password - asynchronous, can be used for notifications.
- Send Phone Message - Synchronous, must call provider to send text message

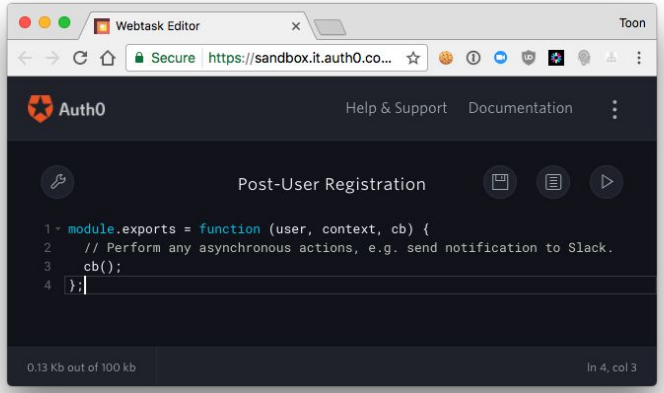
The “*-registration” hooks do NOT run for federated logins. If additional data validation or collection is required - use “Redirect from rules” feature instead.

Client Credentials Exchange

- Use Case
  - > Change the scopes and add custom claims to the tokens
- Triggered when calling the https://{your-tenant}.auth0.com/oauth/token endpoint
- `module.exports = function(client, scope, audience, context, cb) { … }`



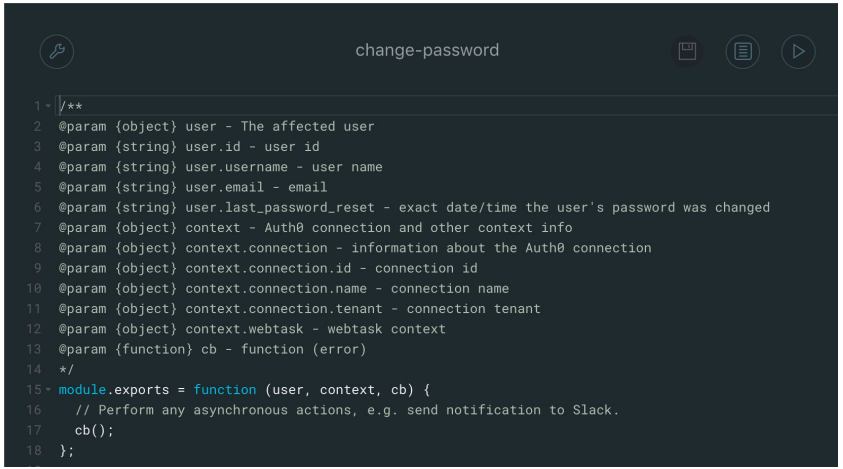
Pre- and Post-User Registration



- Only triggered for Database Connections
- Pre-User Registration
  - > Triggered *before* user is created
  - > Use case
    - prevent user registration and add custom metadata to a newly-created user
- Post-User Registration
  - > Triggered *after* user is created
  - > Use case
    - implement custom actions that execute asynchronously from the Auth0 authentication process after a new user registers and is added to the database

Post Change Password

- Only triggered for Database Connections
- Post-Change password hook
  - > Triggered *after* user's password is changed



  - > Use case
    - implement custom actions that execute asynchronously from the Auth0 authentication process after a user changes his password
      - > Send a password change email
      - > Revoke refresh tokens issued to the user etc.

Configuration, code sharing and testing

Hooks have a separate configuration mechanism - “secrets”, each hook has its own secrets.
Hooks cannot share code - only one hook can be active at a time for each endpoint - any shared code should be injected dynamically, similar to Custom DB scripts.
Hooks are easily tested as they represent fully featured NPM modules.

```
module.exports = function (user, context, cb) {
  var response = {};

  const metadata = user.user_metadata || {};
  metadata.requiresTermsAcceptance = true;
  user.user_metadata = metadata;

  response.user = user;

  cb(null, response);
 };
```

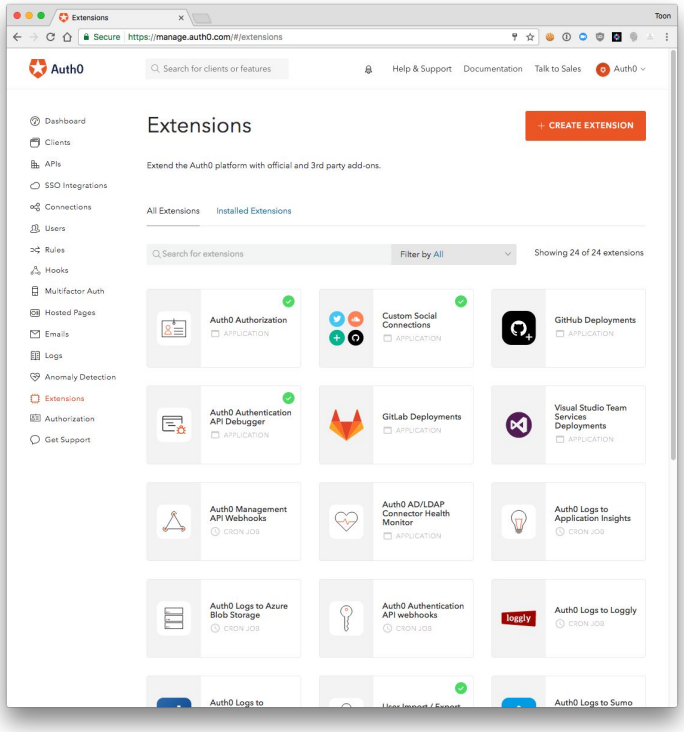
Auth0 Training

6.4. Extensions

Prebuilt helper applications

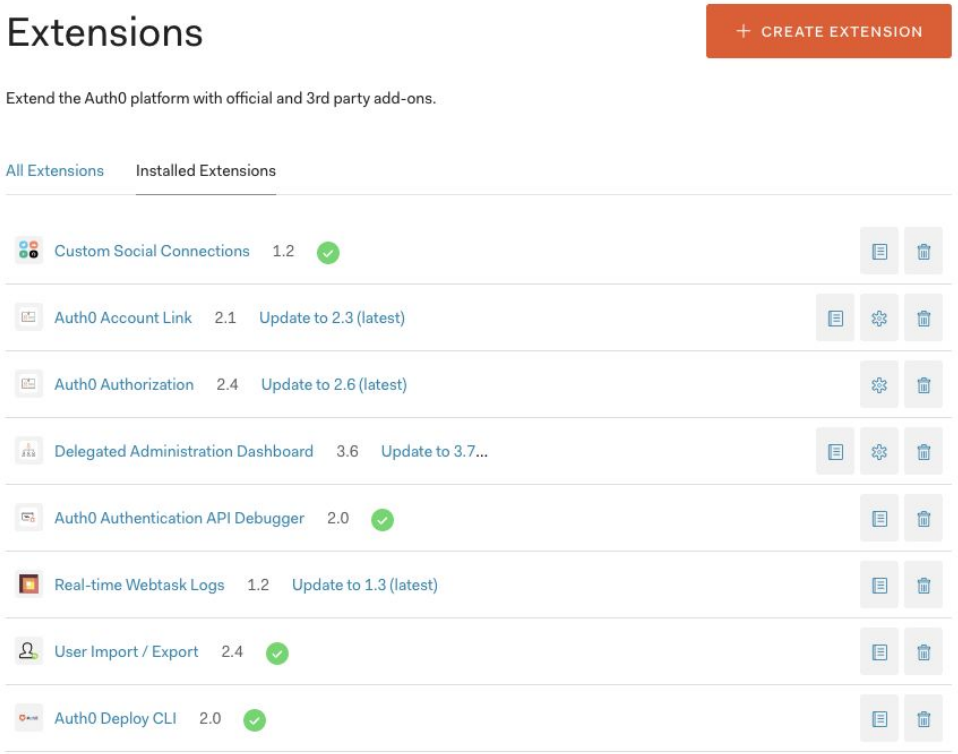
Extensions

Extend the Dashboard with extensions from a 3rd party. Click, configure, add new features instantly.



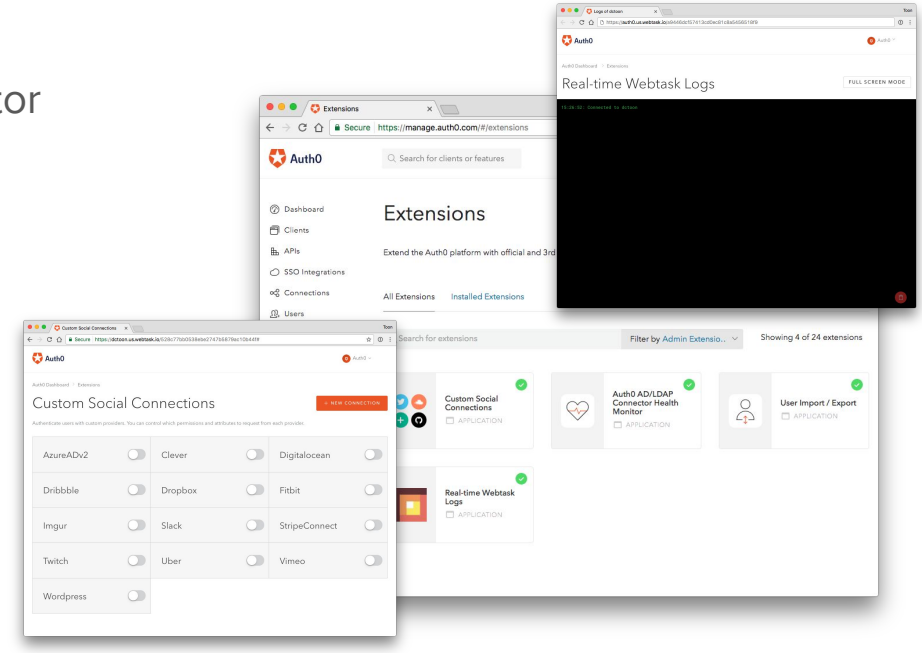
Execution and configuration

Extensions are fully-featured apps run on webtask runtime and using available Auth0 APIs.
All are open-source (MIT license).
Multiple extensions combine both web apps and cron-jobs - like logs export extensions.
Custom extensions can be deployed from Github repositories.



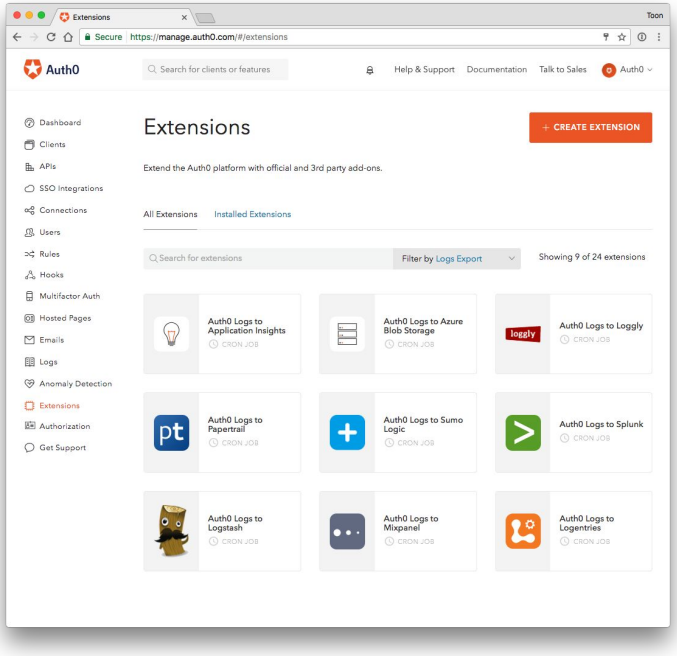
Administration Extensions

- Custom Social Connections
- Auth0 AD/LDAP Connector Health Monitor
- User Import / Export
- Real-time Webtask Logs



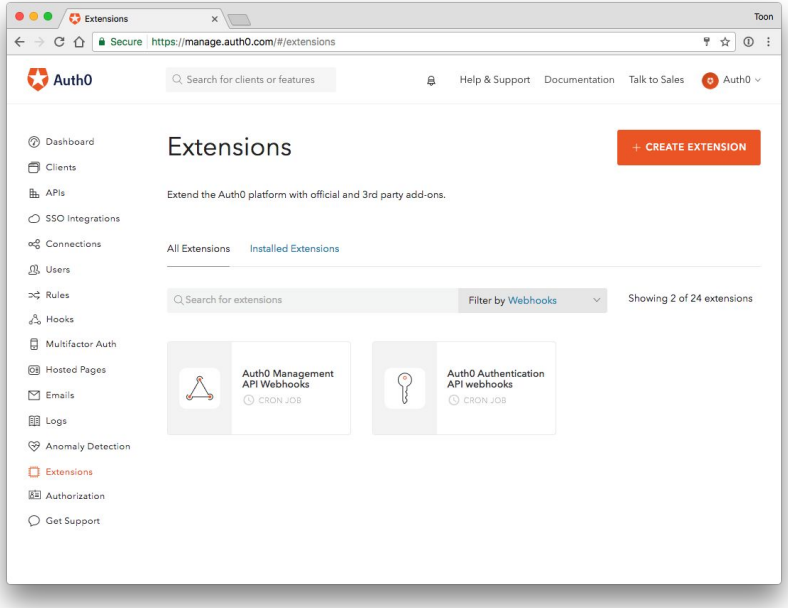
Logs export extensions

- Scheduled Job (every 5 minutes)
- All Auth0 Logs to
  - > Application Insights
  - > Azure Blob Storage
  - > Loggly
  - > Papertrail
  - > Sumo Logic
  - > Splunk
  - > Logstash
  - > Mixpanel
  - > Logentries



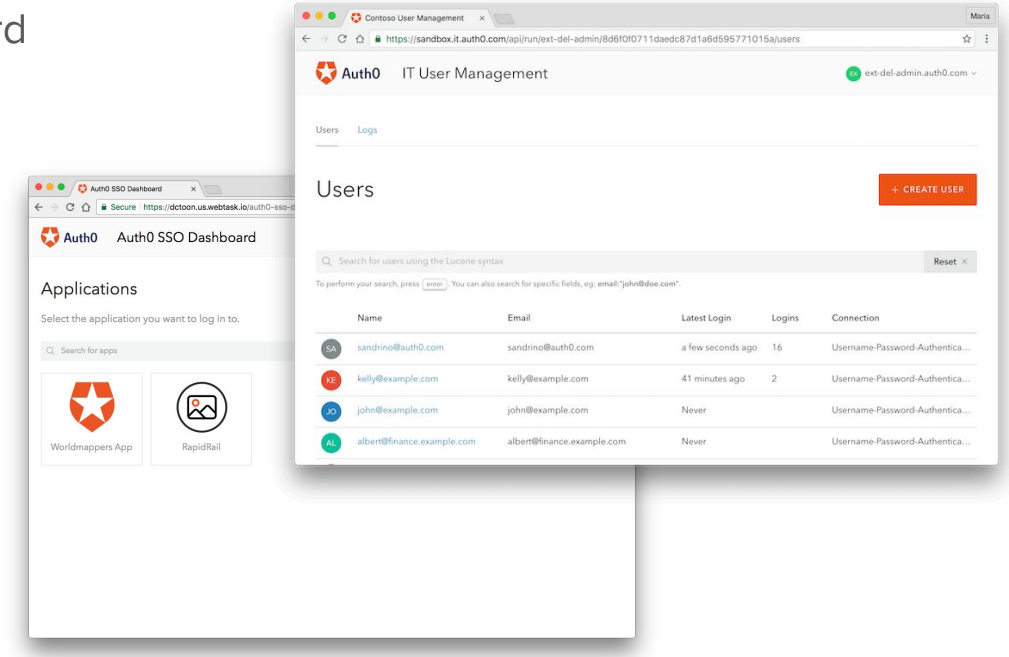
Webhooks

- Auth0 Management API Webhooks
- Auth0 Authentication API webhooks
- Streams - New (Built into Core)



Applications

- Delegated Administration Dashboard
- SSO Dashboard



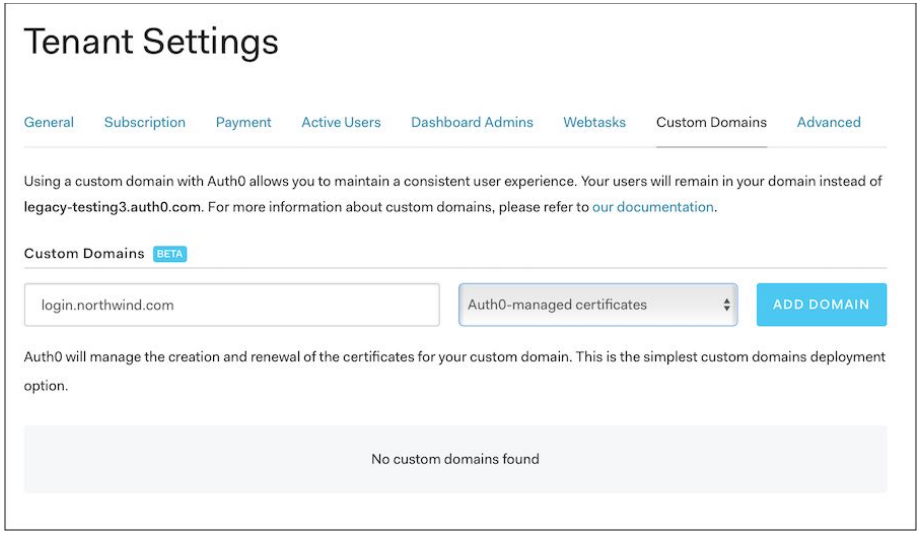
Auth0 Training

6.5. Other extensibility features

A few more customization features

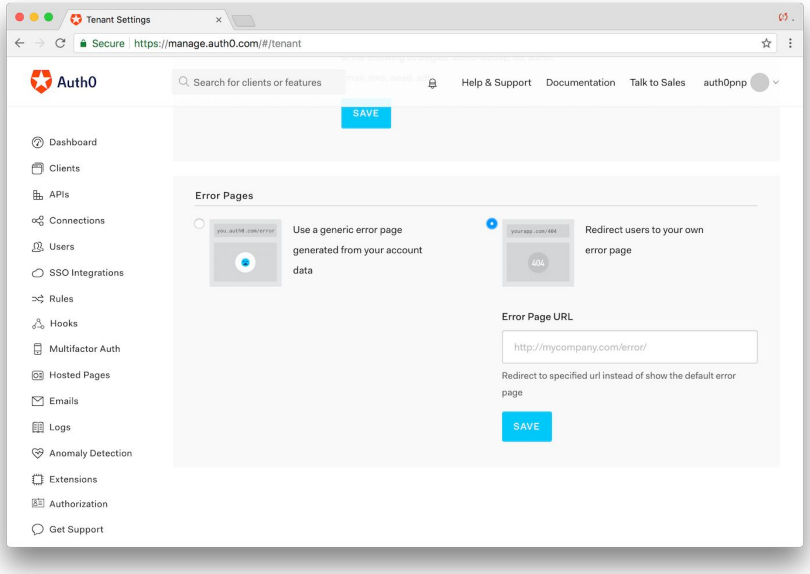
Custom Domains

- Most seamless and secure experience
- To be used in combination with hosted pages
- Two types of certificates
  - Auth0-Managed Certificates
  - Self-Managed Certificates



Error pages

- The error page displayed by Auth0 can be customized:
  - Custom HTML (via API only)
  - Custom page URL



Auth0 Training

6.6. Troubleshooting

What to do when something doesn’t work as expected

Diagnostics

If anything goes wrong:

- Check tenant logs.
- Use Realtime Webtask Logs extension to troubleshoot custom code.
- Make sure the callbacks are called exactly once.

Always set up tenant logs export - this is useful for both monitoring and statistics.

```
function bad(user, context, callback) {
  // visible through Realtime Webtask Logs
  console.log("Start rule...");
  if (user.email === "blah@blah.com") {
    // bad, callback will be called twice
    callback(new UnauthorizedError("Access denied"));
  }
  callback(null, user, context);
}

function good(user, context, callback) {
  if (user.email === "blah@blah.com") {
    // callback called only once due to "return"
    return callback(new UnauthorizedError("Access denied"));
  }
  callback(null, user, context);
 }
```

APIs and Rate Limits

Any calls from rules/hooks/custom DB scripts to Auth0 APIs are still subject to API rate limits:

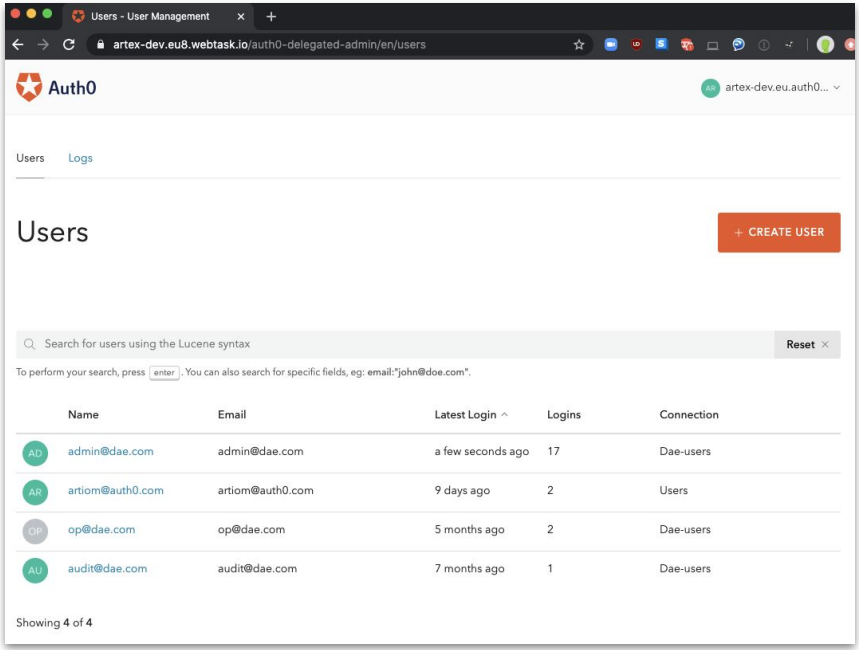
- If possible, minimize calls to Management API from rules.
- If user profile update is needed - make sure to combine multiple calls into one.
- Always plan in advance if there are expected spikes in user logins (events similar to Black Friday).
- When calling Auth0 APIs, inspect the HTTP Response headers for current rate limits - this is useful when implementing request throttling.

6.7. Bonus: Delegated Administration Extension

Built-in and extremely flexible support tool for user management

Delegated Admin Extension (DAE)

- Define a separate connection for DAE users.
- Multiple rules - Admin, Support, Audit, etc.
- Manage users and filter access according to the extension's own hooks.



Delegated Administration Hooks

- Hooks that allows customization of DAE appearance and behavior by DAE admins
- Logging
  - Ctx.log function logs items to webtask logs
- Caching
  - Hooks provide a cache (ctx.global)
  - Persists until webtask container recycles
- Custom Data
  - Up to 400kb of custom data can be stored
- Payload and Request
  - Provides data on current DAE user and user being created/edited.
- Remote Calls
  - Request object available and can be used to call APIs

Context

- Ctx.payload contains two properties
  - Action: operation being performed: read:user, delete:user, etc...
  - User: Information about the user on which the action is being executed
- Ctx.request.user
  - Provides information on DA user performing the operation.

Filter Hook

- By default DAEs are able to see all users in the account. The Filter hook can change that.
- Allows providing a query to filter users.
  - Queries can use the lucene syntax
  - Do not use single quotes, double quotes, or any other special characters (such as + or -) in terms on which you'll want to filter.
  - ```
    // Return the lucene query.
    return callback(null, 'app_metadata.department:"' + department + '"');
    ```
    ```

Access Hook

- Access Hooks control whether the current DA user is allowed to modify a specific user.
- Without the hook all DA users can modify all users.
- Can be used to block specific actions like view details, delete, update, etc...

Access Hook

```
function(ctx, callback) {
  if (ctx.payload.action === 'delete:user') {
    return callback(new Error('You are not allowed to delete users.'));
  }

  // Get the department from the current user's metadata.
  var department = ctx.request.user.app_metadata && ctx.request.user.app_metadata.depart
  if (!department || !department.length) {
    return callback(new Error('The current user is not part of any department.'));
  }

  // The IT department can access all users.
  if (department === 'IT') {
    return callback();
  }

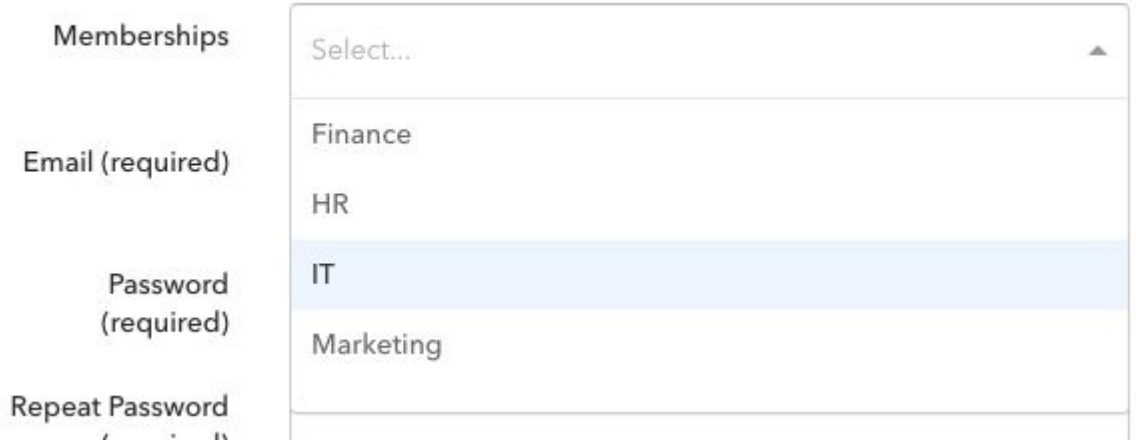
  ctx.log('Verifying access:', ctx.payload.user.app_metadata.department, department);

  if (!ctx.payload.user.app_metadata.department || ctx.payload.user.app_metadata.departm
    return callback(new Error('You can only access users within your own department.'));
  }

  return callback();
}
```

Membership Hook

- Can be used to define departments to organize users.
- Adds a membership dropdown to the create user dialog.
- Write hook needed to persist these changes.



Write Hook

- Write hooks allow you to modify the user being created.
  - Set default values
  - Change user profile information
  - Persist custom DA fields

Settings Query

- Settings Query allows you to customize the look and feel of the DA extension
- Customize labels, add custom css.
- Localization
- Add Custom Fields to create/edit dialogs
  - Custom fields can be configured for Create and/or Edit screens
  - Custom fields editable via change profile profile screen
  - Inputs can be a simple string, select dropdown, or searchable combobox.
- Custom fields are not persisted without a write hook

Limitations

- Cannot configure a custom domain
- Does not support core RBAC
- Does not provide way to store secrets.
  - This leaves you unable to store a client secret to fetch an access token or api key to call a protected endpoint.
- Log events for changes made in delegated admin do not show user making change.

Auth0 Training

Auth0 Extensibility demo

A demo of Auth0 extensibility, testing and troubleshooting

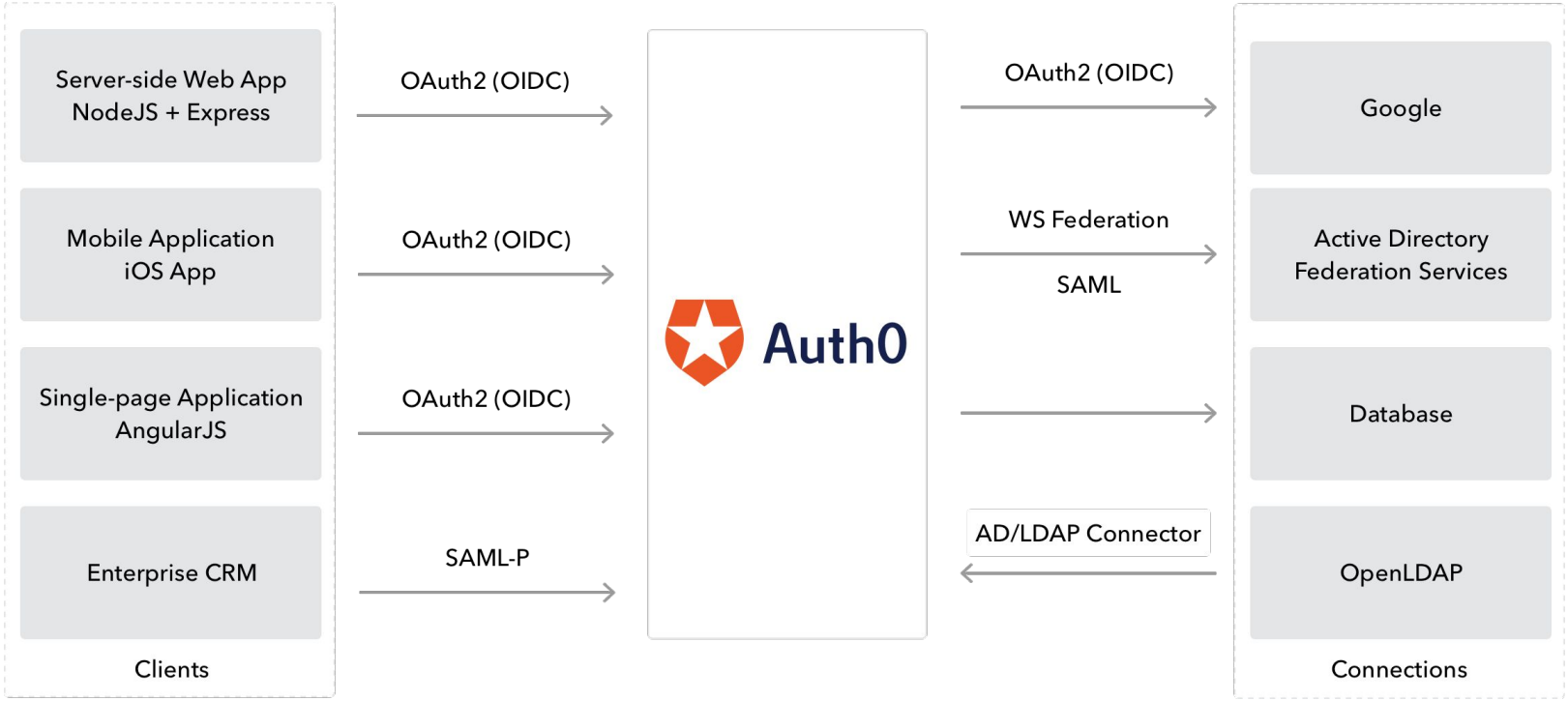
End of Chapter 6

Auth0 Training

7. Architecture Scenarios

Typical use-cases and how to implement them

Architecture Scenarios



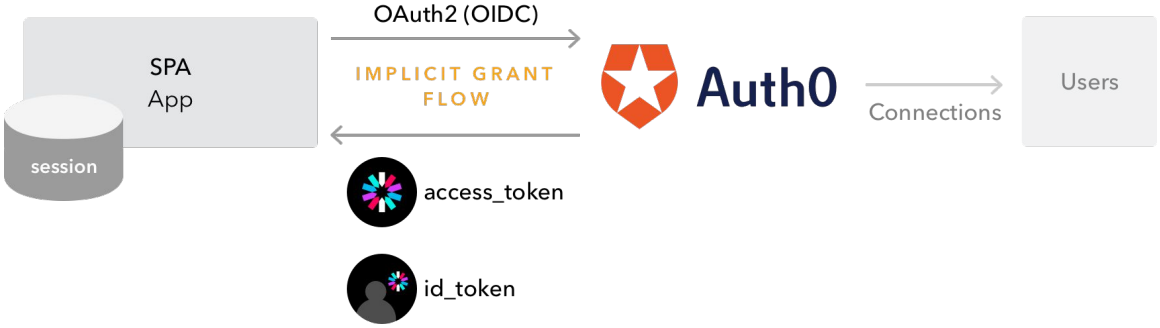
Auth0 Training

7.1. Single Page Apps

Web apps running in the browser

Single Page Apps

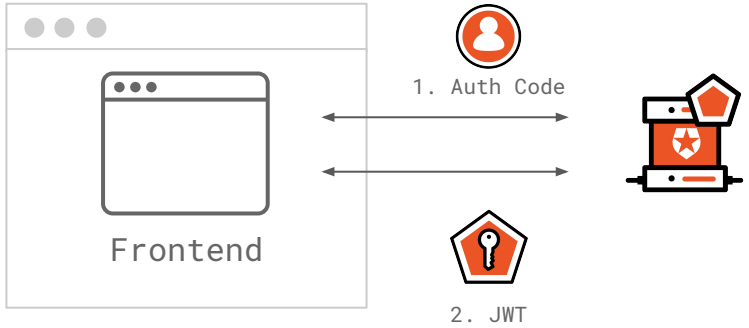
- OAuth2 Implicit Grant Flow - must not be used for any new development!
- OpenID Connect
- No secret storage (runs in front end)



Single Page Apps

Web apps without a backend.

- Auth Code + PKCE flow
- Public Client
- Only rotating refresh tokens
- Prefer to store tokens in memory
- Possible issues with Silent Authentication
- Avoid Implicit flow for new apps



PKCE

Proof Key of Code Exchange - makes Authorization Code interception useless for attacker.

1. App generates a random secret (“code verifier”).
2. When requesting an Authorization Code - sends the hash of the secret (“code challenge”).
3. When exchanging the Authorization Code for tokens - the app sends the code verifier and the Identity Provider computes the code challenge to match the one sent previously.

It makes sense when the app runs in an environment where its communication with the user agent can be intercepted. Does not make sense for traditional web apps as the browser communicates directly with the web server with TLS.

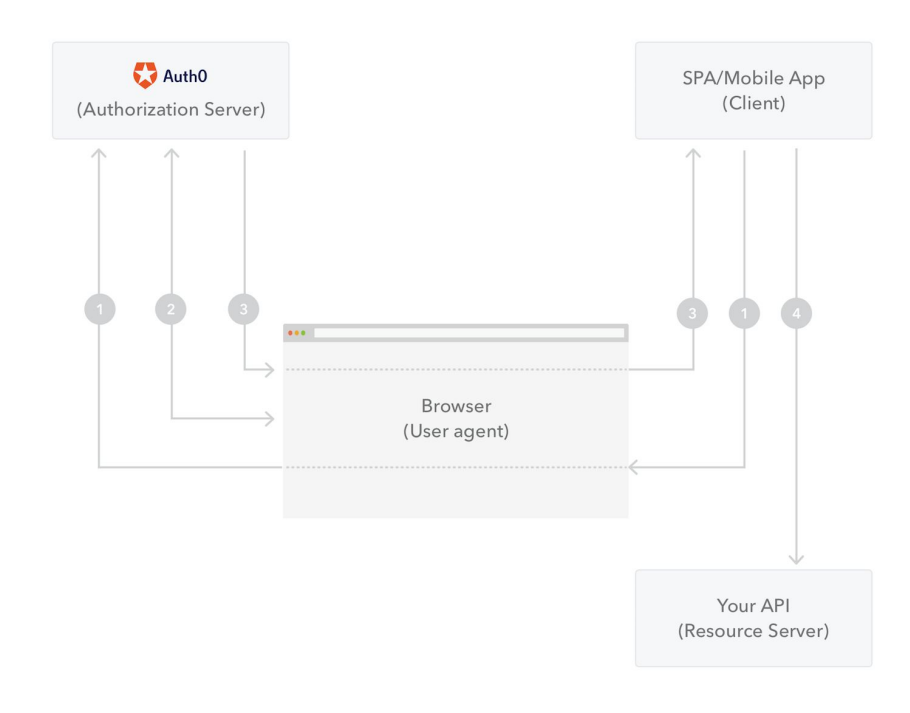
SPA token storage

Bad access token storage options:

- Cookies - SPA would have to create them from javascript, meaning it is vulnerable to XSS and XSRF. Not good.
- Local Storage - still vulnerable to XSS. Also don't forget to clean it if user logs out. This is the only viable option for rotating refresh tokens.

Good storage options:

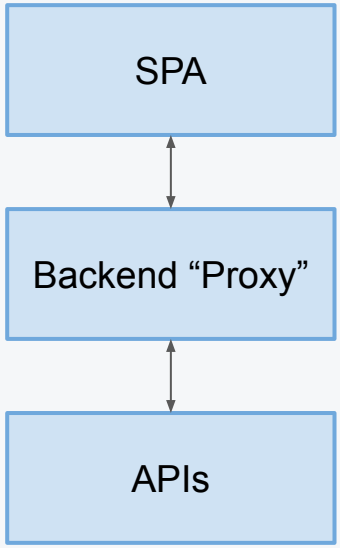
- No storage (=RAM) - just keep it in a variable. While still not ideal, it is the only good way for an SPA - an attacker would have to know app's internal structure, which increases the relative cost of an attack.



SPA - maximum security

Maximum security of an SPA can be achieved when it is reduced to a Traditional Web Application type:

- A backend is created that stores all tokens server-side.
- The SPA frontend switches to Authorization Code flow (the Code is sent to Backend to be exchanged for tokens).
- The SPA frontend gets a Secure and HTTPOnly session cookie with a random identifier.
- Any calls to APIs are proxied through the Backend which identifiers the user's session by the cookie and uses tokens for subsequent API calls.



Auth0 Training

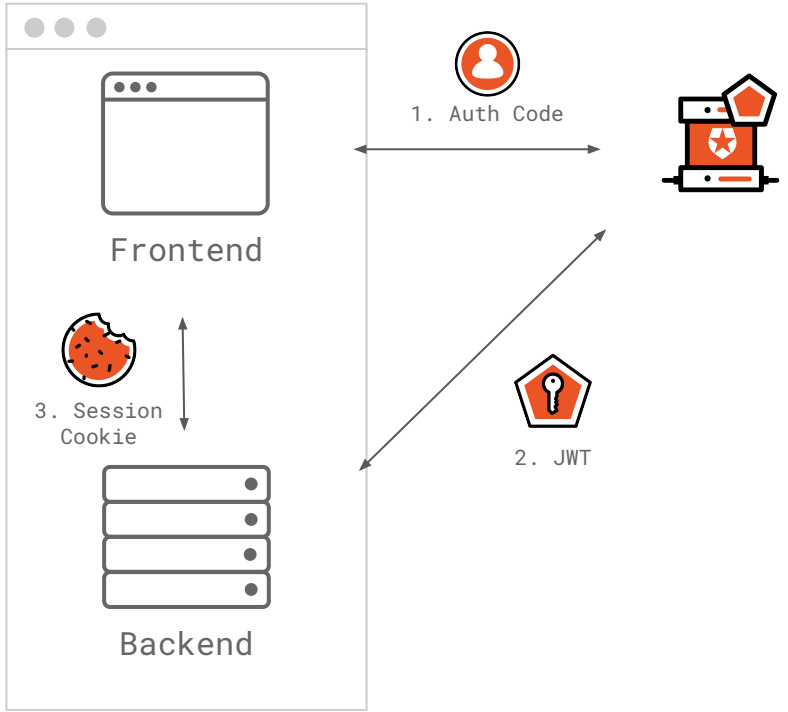
7.2. Traditional Web Apps

Web apps with a backend

Traditional Web Apps

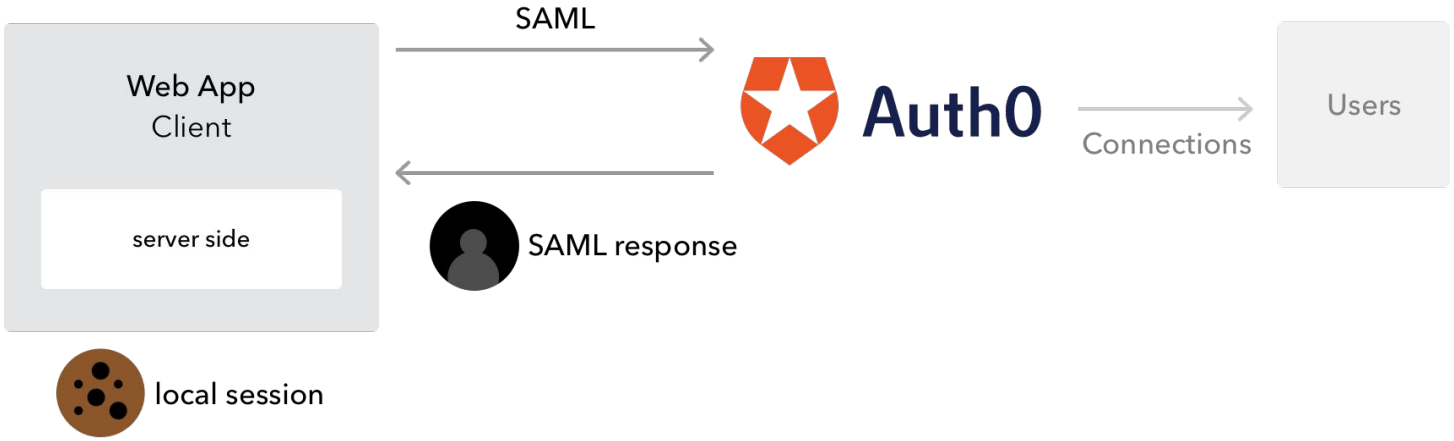
Typical web applications with a backend capable of storing client secret.

- Auth Code or Hybrid flow
- Confidential Client
- Can use ID Token for session tracking
- Prefer HTTPOnly cookies for session management



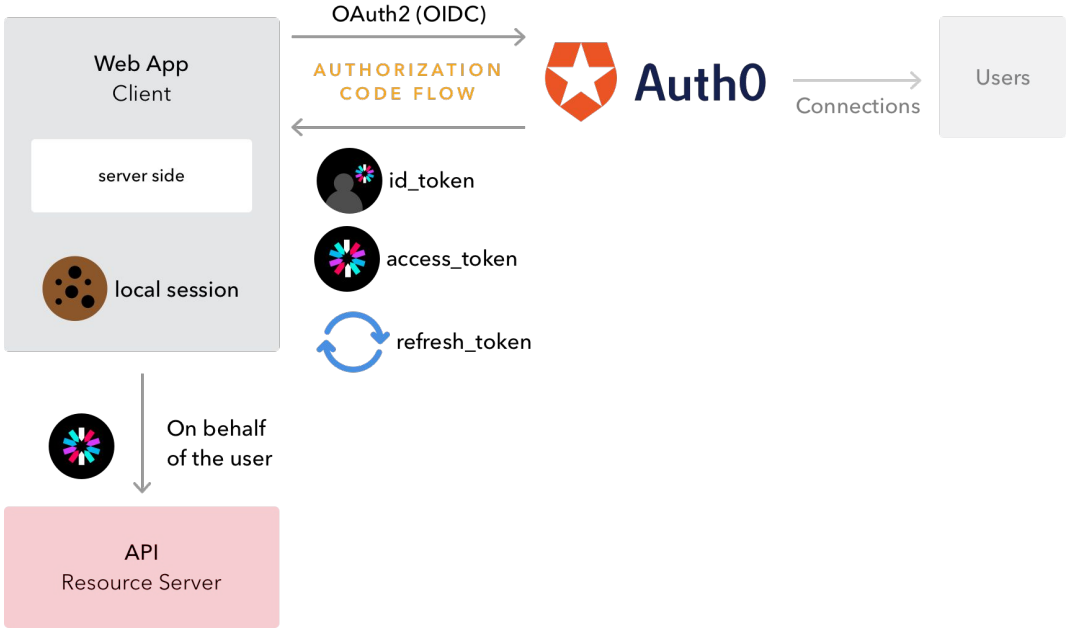
Regular Web Application Using SAML

- App creates a SAML request
- Auth0 returns SAML response
- Auth0 can act as SP, IdP or both



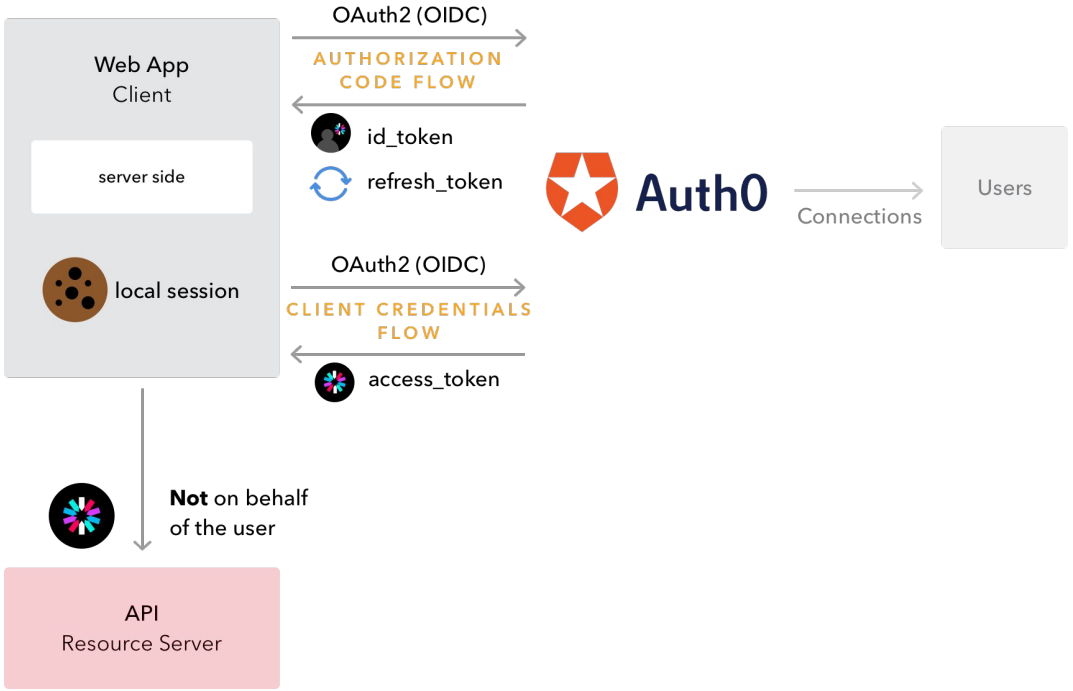
Authorization

- Access Token for Authorization
- Use Access Token to call API
- On behalf of the user



Authorization

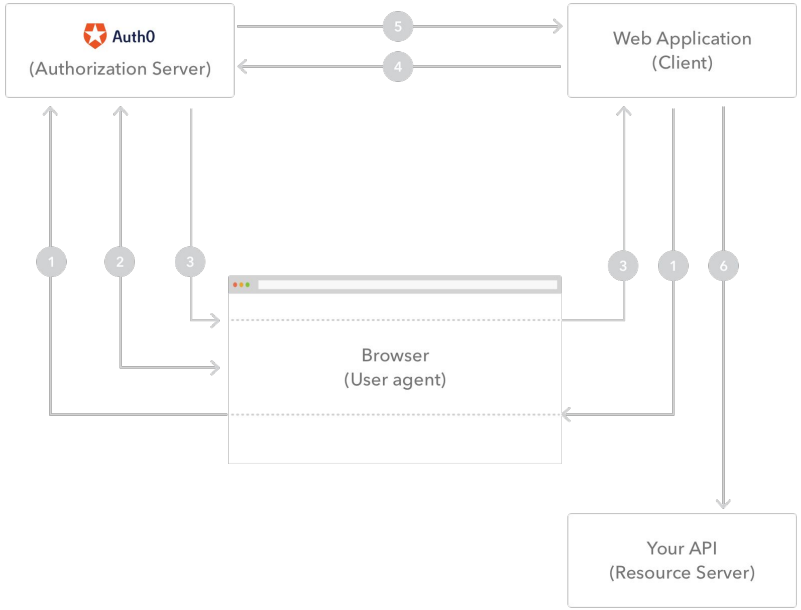
- Client Credentials Flow is used to get the `access_token`
- **Not** on behalf of the user



Authorization Code flow

1. The web app initiates the flow and redirects the browser to Auth0 (specifically to the `/authorize?response_type=code` endpoint), so the user can authenticate

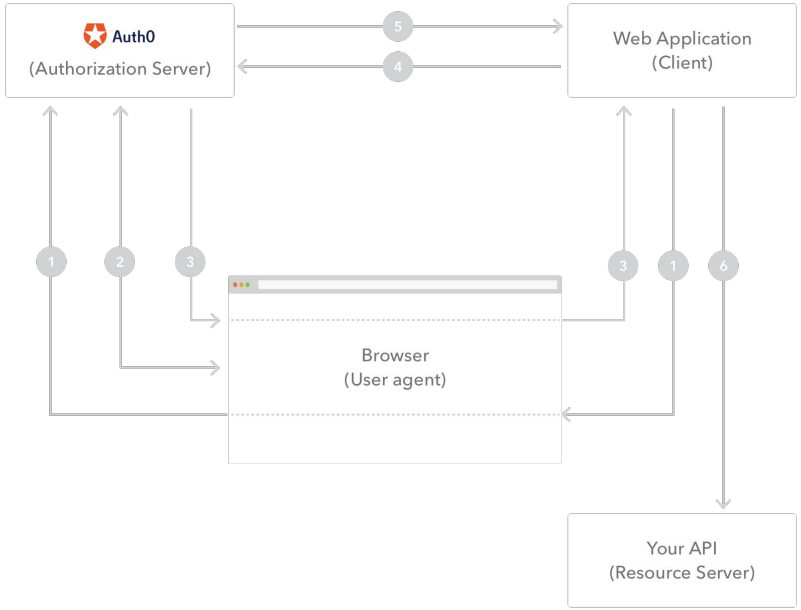
2. Auth0 authenticates the user (via the browser). Optionally a consent page will be shown where the permissions are listed that will be given to the Client



Authorization Code flow

3. Auth0 redirects the user to the web app (specifically to the `redirect_uri`, as specified in the `/authorize` request) with an *Authorization Code* in the querystring (code).

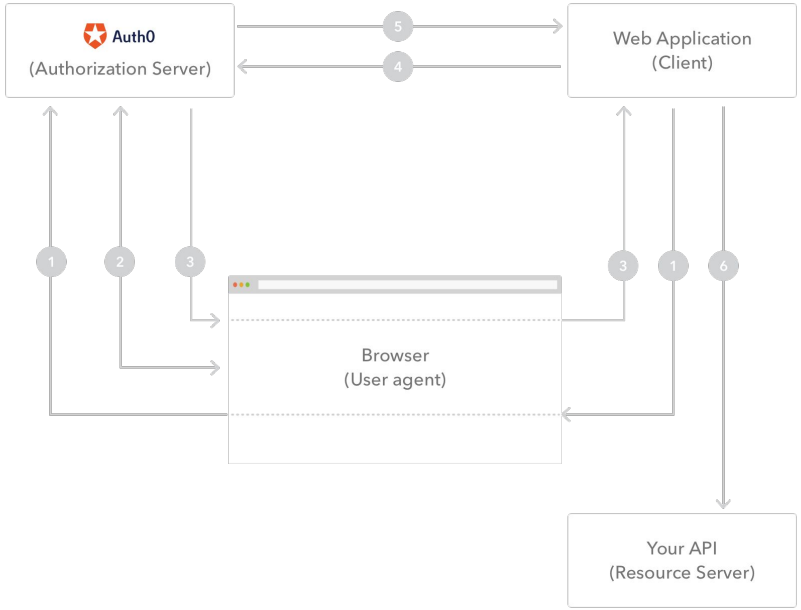
4. The web app sends the *Authorization Code* to Auth0 and asks to exchange it with an `access_token` (and optionally an `id_token` and a `refresh_token`). This is done using the `/oauth/token` endpoint. When making this request, the web app authenticates with Auth0, using the *Client Id* and *Client Secret*.



Authorization Code flow

5. Auth0 authenticates the web app, validates the Authorization Code and responds back with the token

6. The web app can use the access_token to call the API on behalf of the user



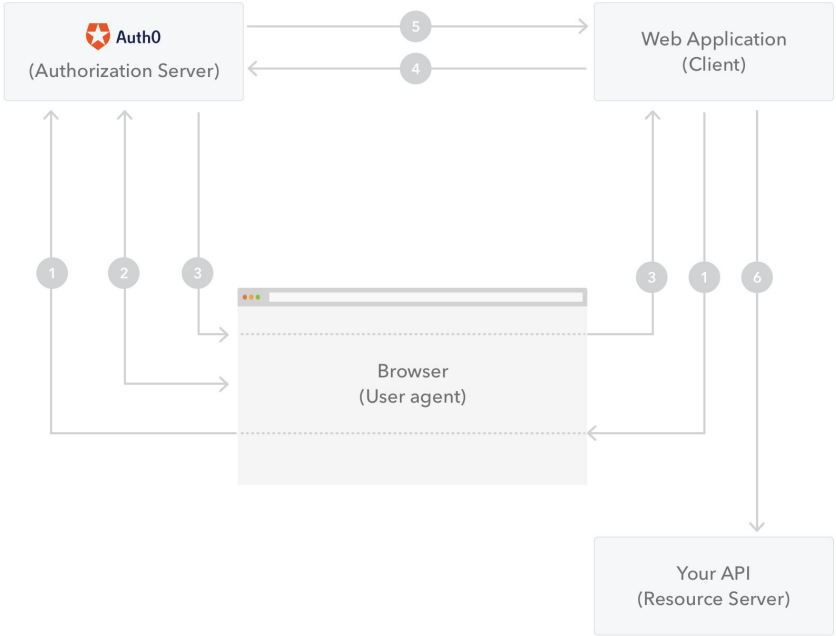
Token storage

Bad token storage options:

- Cookies to make my app stateless and RESTful! That's not ideal as it makes it vulnerable to XSRF and potentially data leaks.

Good token storage options:

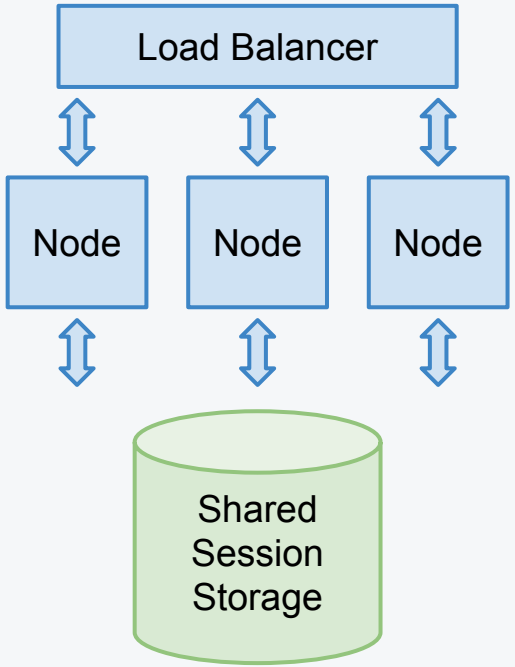
- Server-side session storage - in memory, a database or anything really that can keep tokens server-side.
  - Access token is short-lived - session storage is enough.
  - ID token must be consumed.
  - Refresh token requires durable storage and thinking about revocation.



High-Availability Deployments

HA deployments for horizontal scaling require a separate Shared Session Storage to keep the tokens server side.

Because of this some customers prefer to store tokens in cookies (!) which makes them vulnerable to XSRF. The argument “we don’t want to install and maintain yet-another-service” is not acceptable from security point of view.



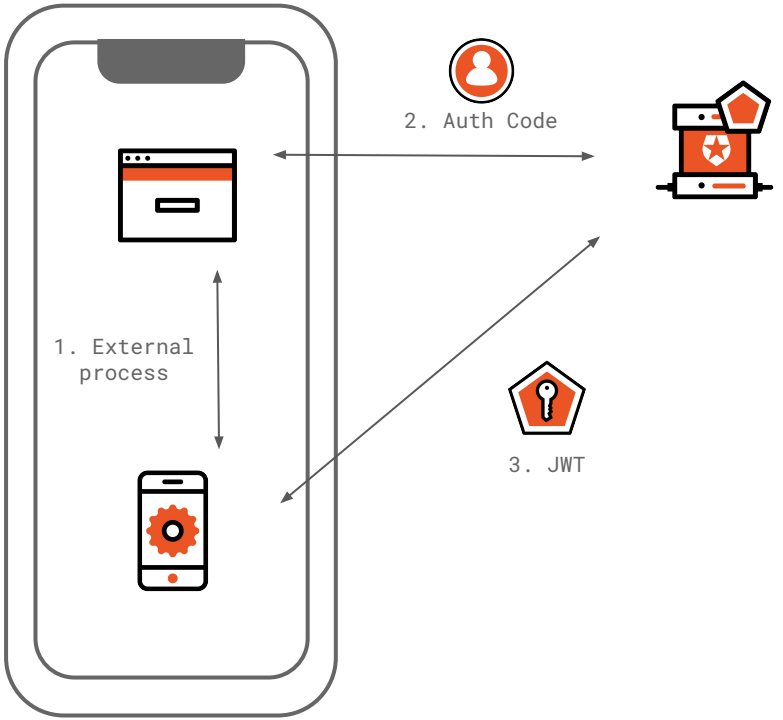
7.3. Native Apps

Apps running on user's device (desktop or mobile)

Mobile Apps

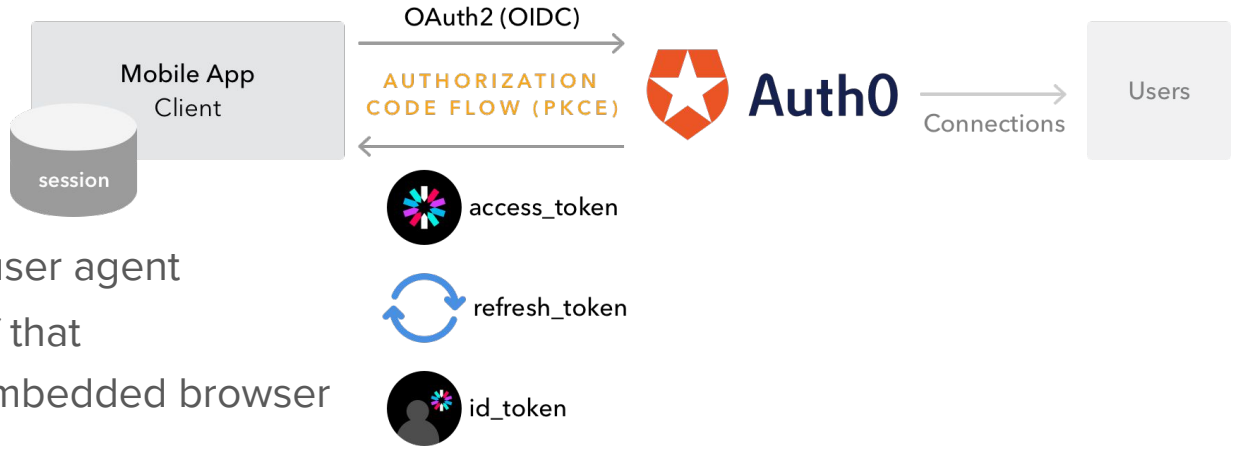
Apps running on mobile devices.

- Auth Code + PKCE
- Public Client
- Has secure storage for Refresh Token
- Prefer Native Browser for authentication



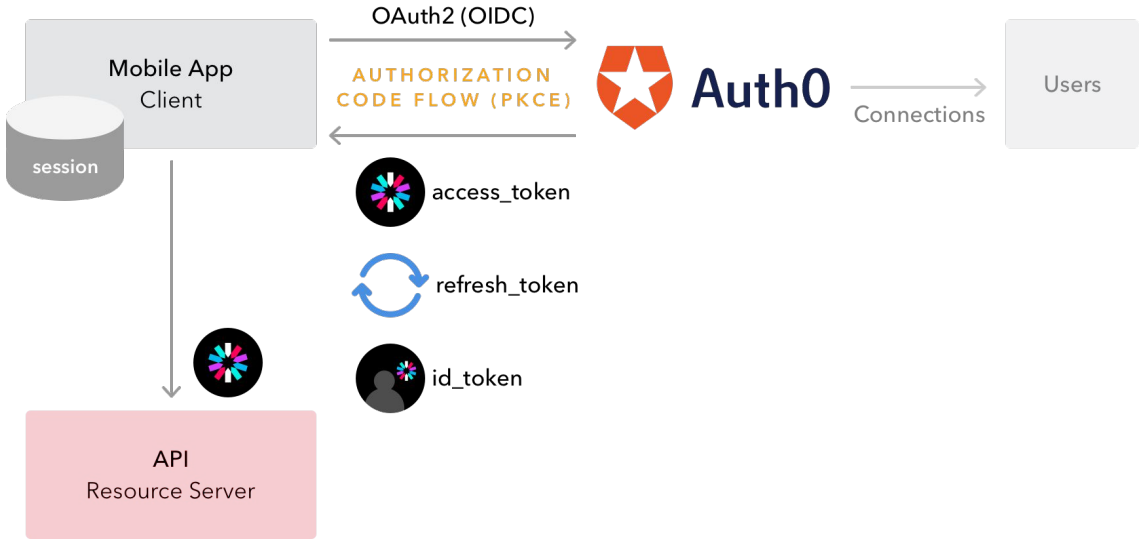
Mobile / Desktop - Authentication

- OAuth2 Authorization Code Flow PKCE (Proof Key for Code Exchange)
- OpenID Connect
- Typically an Android or iOS Application
- Public client
- Can use refresh tokens
- Use the native browser/user agent
- PKCE is used because of that
- Desktop apps can use embedded browser



Mobile / Desktop - Authorization

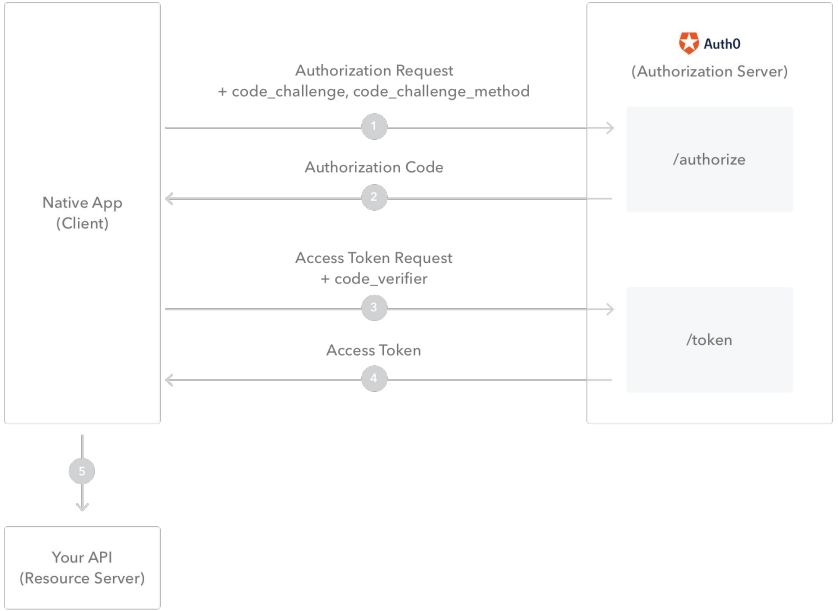
- Access Token for Authorization
- Use Access Token to call API
- On behalf of the user



Proof Key of Code Exchange (PKCE)

1. The native app initiates the flow and redirects the user to Auth0 (specifically to the /authorize endpoint), sending the `code_challenge` and `code_challenge_method` parameters

2. Auth0 redirects the user to the native app with an `authorization_code` in the querystring

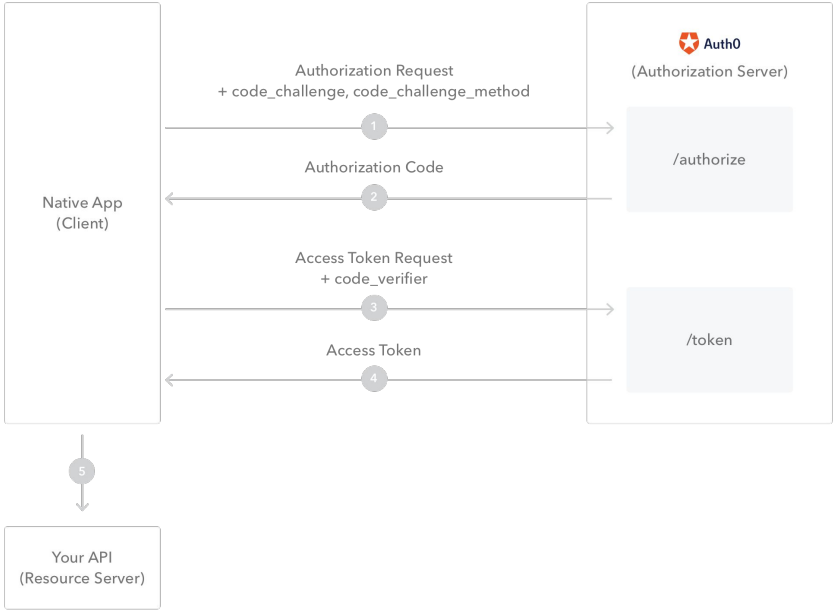


Proof Key of Code Exchange (PKCE)

3. The native app sends the `authorization_code` and code_verifier together with the `redirect_uri` and the `client_id` to Auth0. This is done using the `/oauth/token` endpoint

4. Auth0 validates this information and returns an `access_token` (and optionally a `refresh_token`)

5. The native app can use the `access_token` to call the API on behalf of the user



Auth0 Training

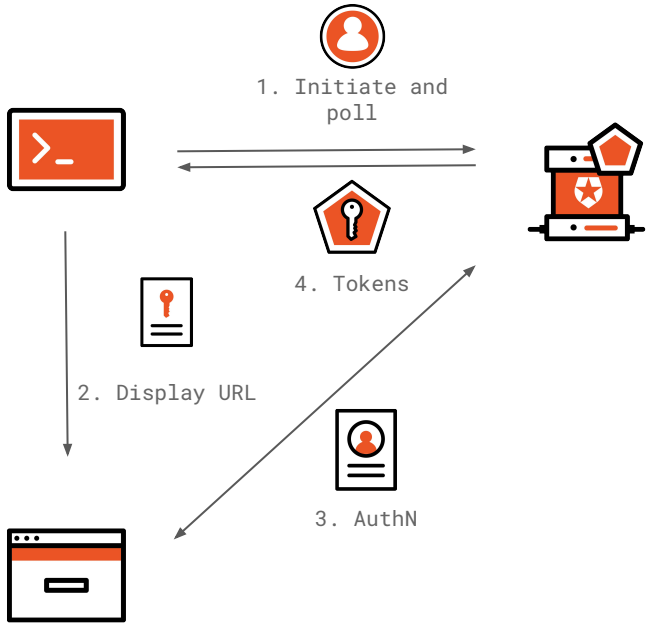
7.4. Limited Input Apps

Apps with limited input - TVs and IoT

Device Flow

Web apps that can't use native browser or have limited input capabilities.

- Device flow
- Public or confidential client
- Can use Refresh Token
- TV, IoT, Desktop, CLI
- Can use Display or NFC to communicate auth URL



Auth0 Training

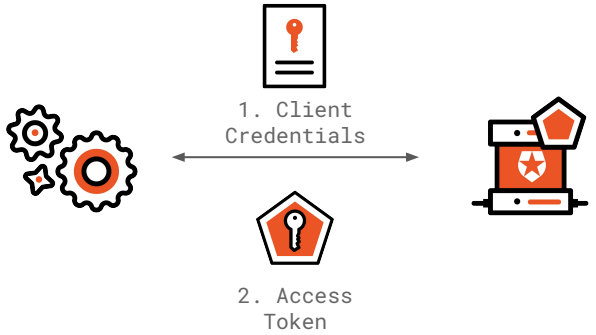
7.5. Non-interactive Apps

Machine-to-machine authentication

Non-interactive apps

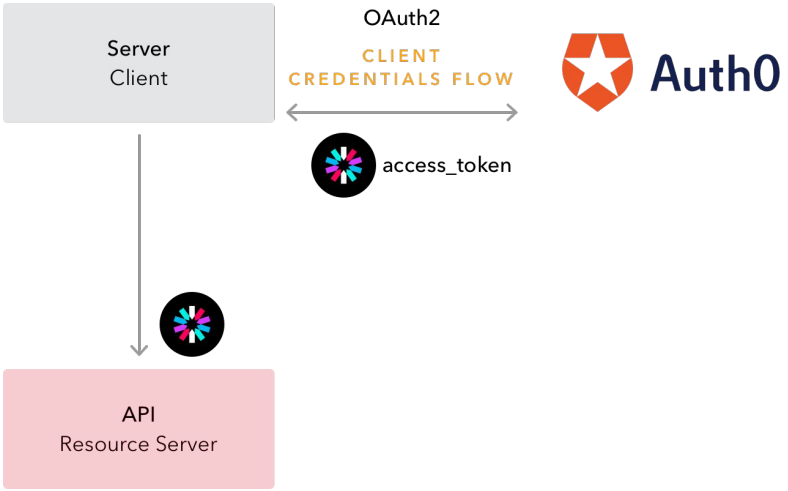
Services with no user interaction.

- Client Credentials flow
- Confidential Client only
- Prefer to cache the token until expiration
- No Token Exchange (for now)



Authentication

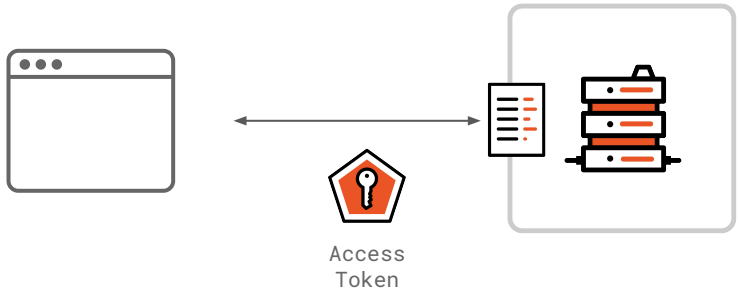
- OAuth2 Client Credentials Grant
- Non Interactive Client
  - Cli
  - Daemon
  - Service running on your backend
- Programmatically and securely obtain access to an API
- Access Token for Authorization
- Use Access Token to call API
- On behalf of the client itself



Calling APIs

Apps can call external API which are not part of the app itself.

- Use Access Tokens
- Designate APIs with Audience
- Verify token: signature, issuer, audience, expiration +others

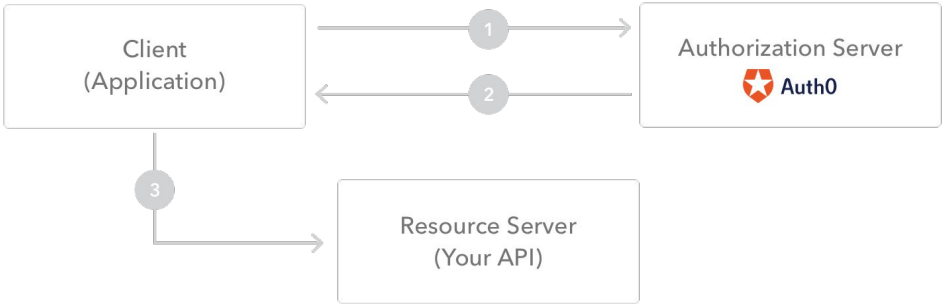


Client Credentials Flow

1. The Client authenticates with the Authorization Server using its **Client Id** and **Client Secret**

2. The Authorization Server validates this information and returns an **`access_token`**

3. The Client can use the **`access_token`** to call the Resource Server on behalf of itself



Auth0 Training

7.6. Legacy apps

Integration of apps which cannot use redirect flows

Resource Owner Password Grant

- Also known as ROPG
- Directly obtain an access token and optionally a refresh token
- Only be used if high degree of trust between user and client
- If not other authorization flows are available
- MFA support coming soon
- Realm support via extension grant
  - Allows you to choose an Auth0 connection
- Can be used in automated testing

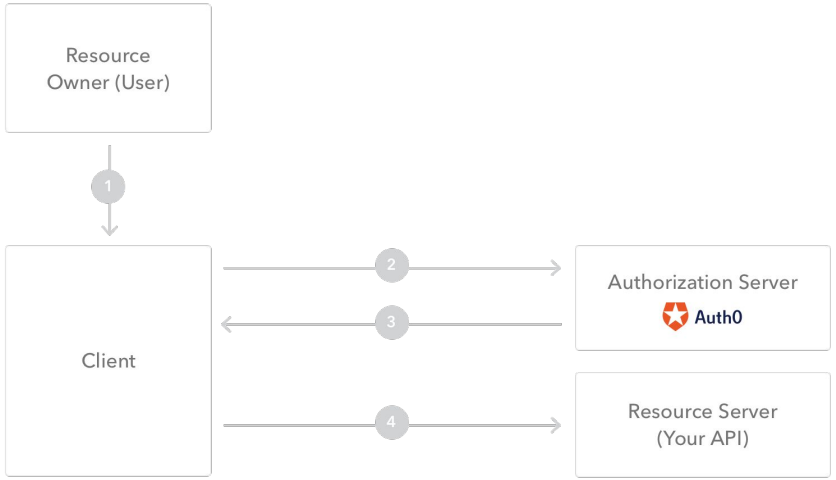
Resource Owner Password Grant

1. The Resource Owner enters the credentials into the client application

2. The client forwards the Resource Owner credentials to the Authorization Server

3. The Authorization server validates the information and returns an `access_token`, and optionally a `refresh_token`

4. The Client can use the `access_token` to call the Resource Server on behalf of the Resource Owner



Auth0 Training

7.7. Migration to Auth0

Step-by-step integration

User Account Migration options

- **Automatic migration** - users migrated upon login. If user is not present in Auth0 DB, credentials verified in legacy store, then user created in Auth0 database. User not required to set a new password.
- **Bulk Import** with Management API. Call the management api to trigger an async import job. Can query status of job and details of any failed migrations.
- **User Import/Export Extension**. Works similarly to Management API but via a UI.

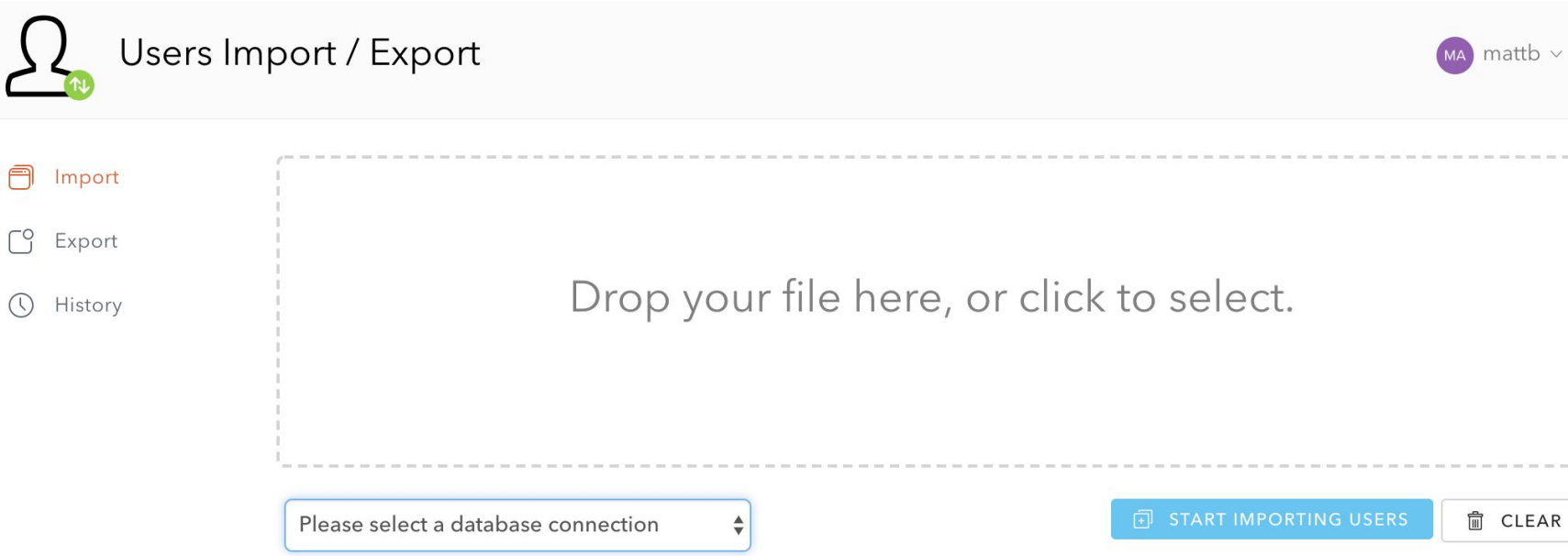
Automatic Migration

- Requires two scripts that will run against your legacy db.
  - **Login script** authenticates credentials against your legacy flow.
  - **Get user script** checks that user exists prior to flows that do not require authentication like sign-up and reset password.
- After first login or password reset the user will be copied to the Auth0 database and legacy database will not be queried for that user.
- Automatic migration from a login will allow passwords that do not satisfy password strength requirements.

Bulk Import Via management API

- Management APIs v2/jobs/users-imports endpoint allows you to create a job to bulk import users.
- By default two concurrent jobs allowed.
- Endpoint parameters:
  - (Required) Json file containing user list
  - (Required) Connection user's are being added to
  - Upsert: if true update an existing user
  - External_id: Associates an arbitrary id with the job
  - Send_completion_email: if true all tenant owners are notified upon job completion
- User must reset password after migration if custom password hash not supported, can do this manually or you can trigger it via the Authentication API's change password endpoint

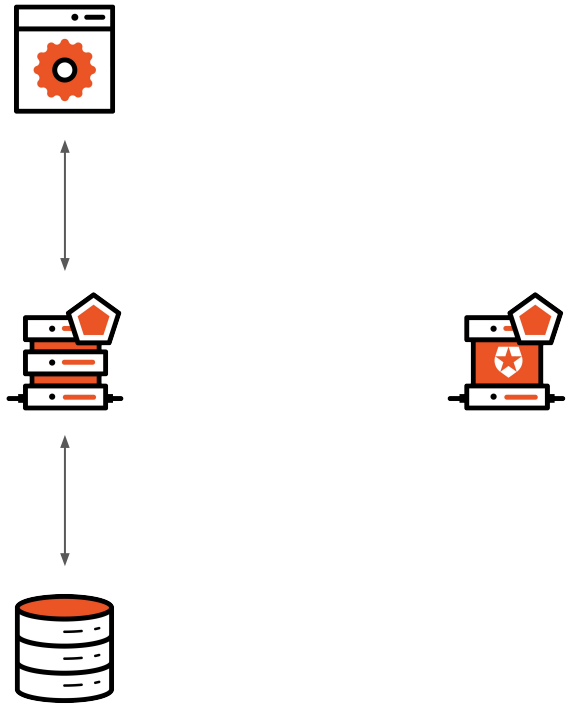
Import/Export Extension



Legacy Systems Integration

Typical steps to integrate a legacy system:

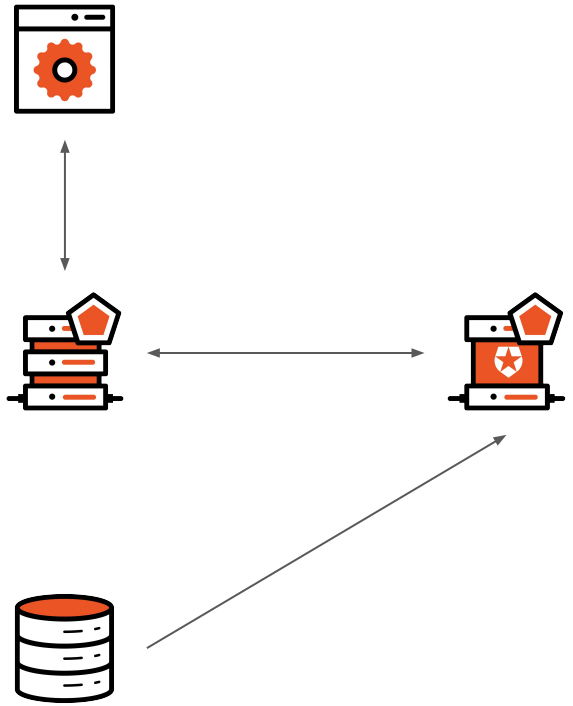
1. Centralize authentication and set up a Custom DB in Auth0
2. Start migrating apps to Auth0 and set up Automatic Migration
3. Complete migration and decommission legacy system



Legacy Systems Integration

Typical steps to integrate a legacy system:

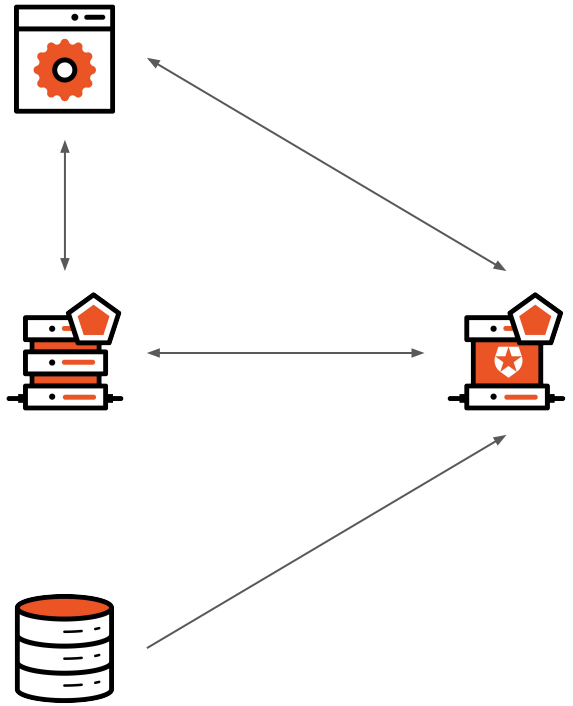
1. **Centralize authentication and set up a Custom DB in Auth0**
2. Start migrating apps to Auth0 and set up Automatic Migration
3. Complete migration and decommission legacy system



Legacy Systems Integration

Typical steps to integrate a legacy system:

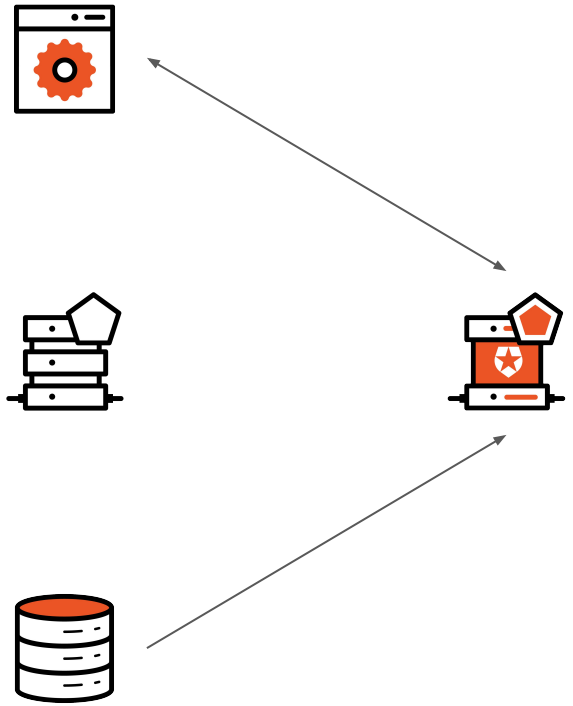
1. Centralize authentication and set up a Custom DB in Auth0
2. **Start migrating apps to Auth0 and set up Automatic Migration**
3. Complete migration and decommission legacy system



Legacy Systems Integration

Typical steps to integrate a legacy system:

1. Centralize authentication and set up a Custom DB in Auth0
2. Start migrating apps to Auth0 and set up Automatic Migration
3. **Complete migration and decommission legacy system**



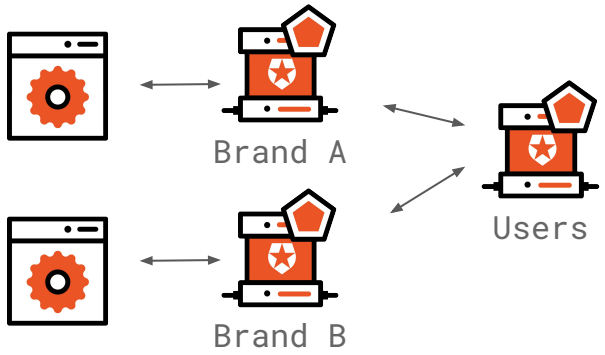
7.8. Bonus: Using multiple Auth0 tenants

For advanced scenarios

Using Multiple Auth0 Tenants

Auth0 tenants can federate between themselves to achieve different combinations.

- Can federate via: Custom DB, OIDC, SAML
- Represent different brands
- Allow employee access to public apps
- Centralize user management



End of Chapter 7

Auth0 Training

8. Day-to-day Operations

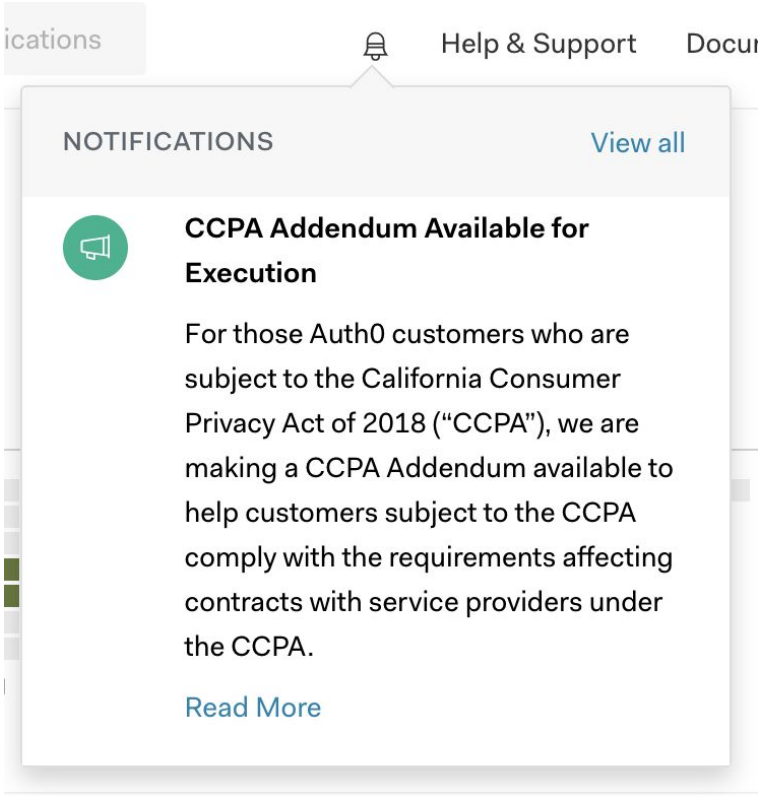
Typical operations, monitoring and troubleshooting

General operations

While Auth0 is designed to work without intervention, the functionality evolves over time.

- Monitor the notifications for any deprecations or maintenance.
- Deprecations usually have 6 months notice unless is a critical vulnerability.
- Monitor for any spikes in errors after configuration changes

https://support.auth0.com/notifications



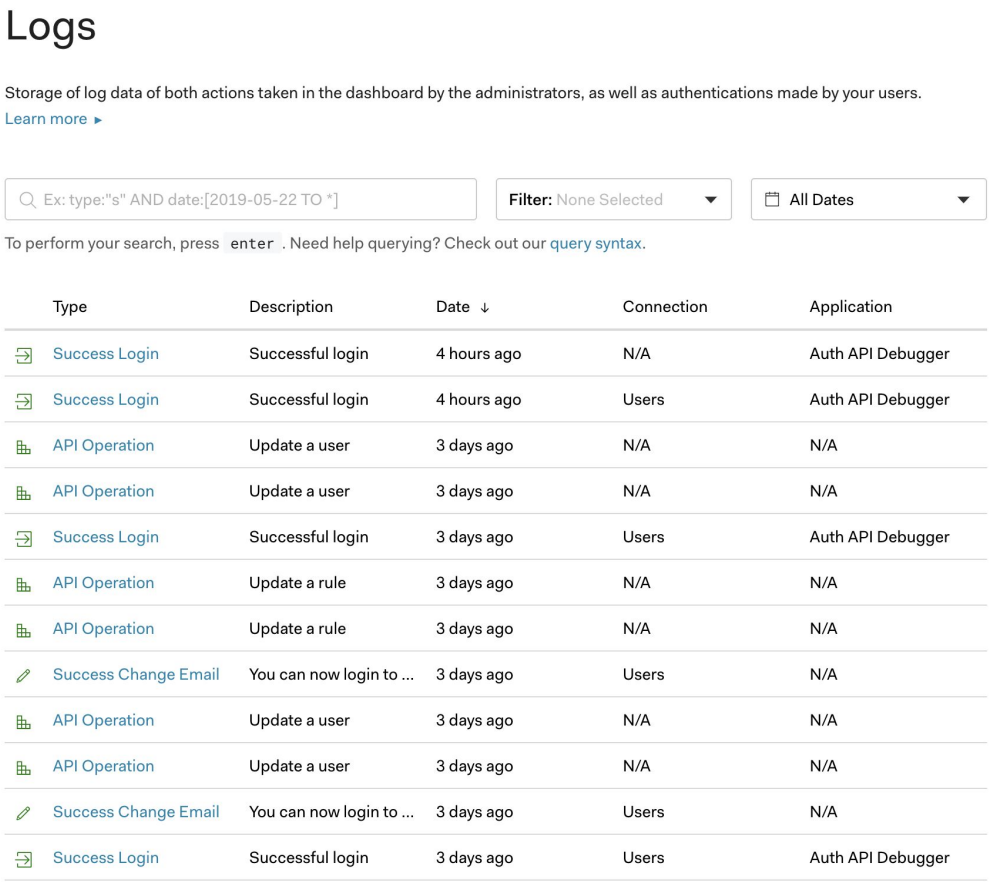
Logs export

Max log retention: 30 days

Logs can be exported via Management API or via extensions.

Exported logs provide:

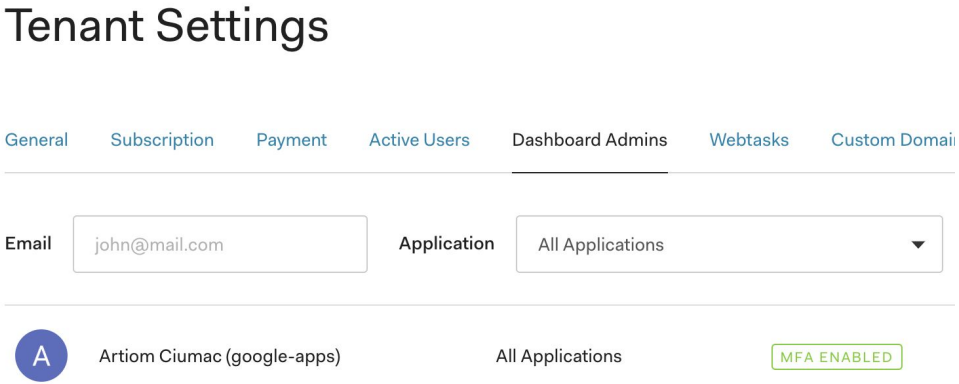
- Monitoring
- Reporting
- Integrations



Tenant administrators

A tenant administrator has full control over all setup, including ability to export user profiles.

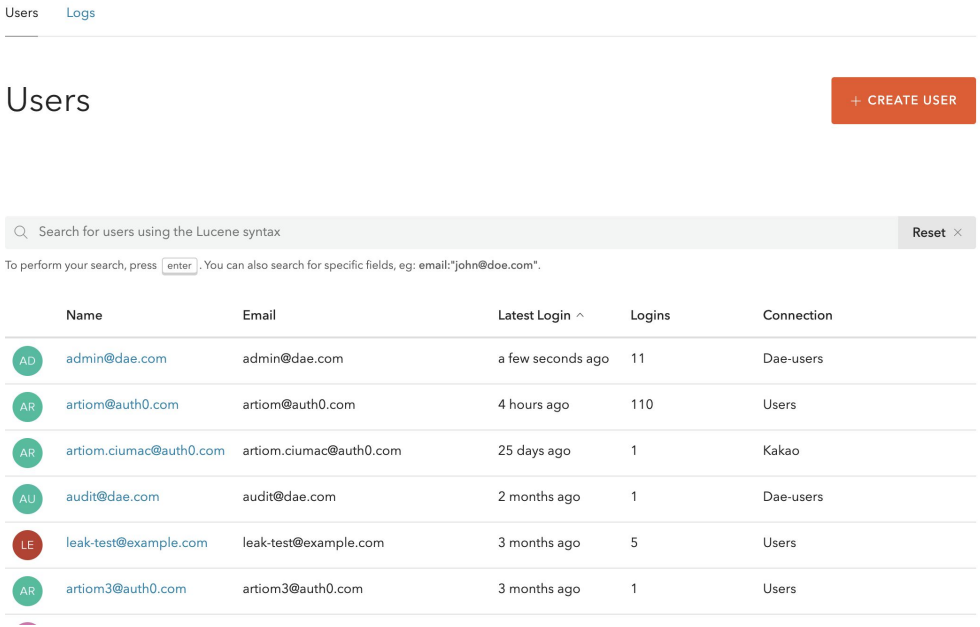
- Make sure all administrators have MFA enabled
- If needed - use the corporate IdP for tenant administrators login



User support service

Provide the tools for support agents to diagnose issues:

- “Delegated Administration” extension.
- Integrate via API
- Script repetitive tasks



Automated testing

- API Rate Limits can affect tests or other integrations
- Avoid creating users at the beginning of each test - this can quickly exhaust the MAU quota
- Isolate frequent tests from Auth0
- Use ROPG to obtain tokens programmatically
- Prefer to isolate Auth0 from frequent tests
- Use browser automation for end-to-end tests
    - DO NOT record and replay network requests

Auth0 Training

8.1. Monitoring

Collecting metrics and detecting unusual behaviour

Log types

Auth0 generates various log events and improves them over time. They can be broadly categorised as following:

- Notices - for example “depnote” - notify administrators, no immediate action is required.
- Success - “s*” - establish a baseline, monitor spikes
- Failures - “f*” - establish a baseline, monitor spikes
- Limits - “*limit*” + other anomaly detection - alert immediately

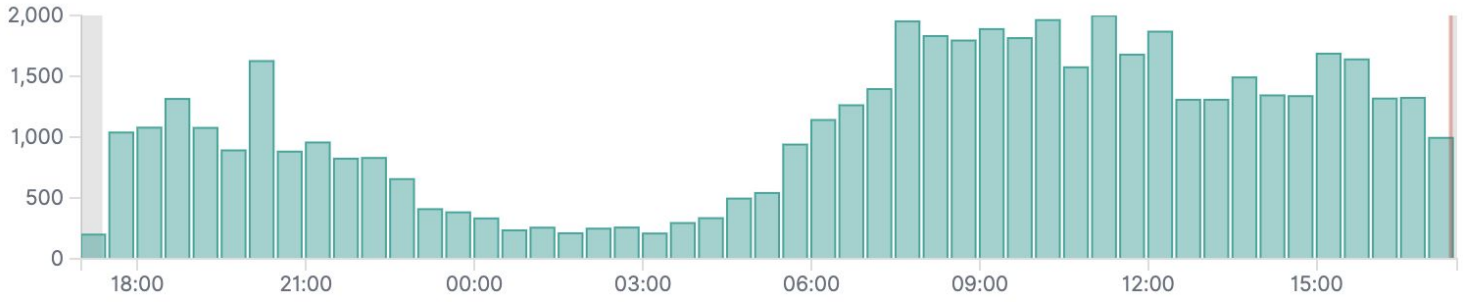
https://auth0.com/docs/logs/references/log-event-type-codes

Internal platform logs are not accessible for customers but can be used by Auth0 support to troubleshoot issues.

Monitoring log data

User type incorrect passwords all the time - it is ok to have a steady stream of errors.

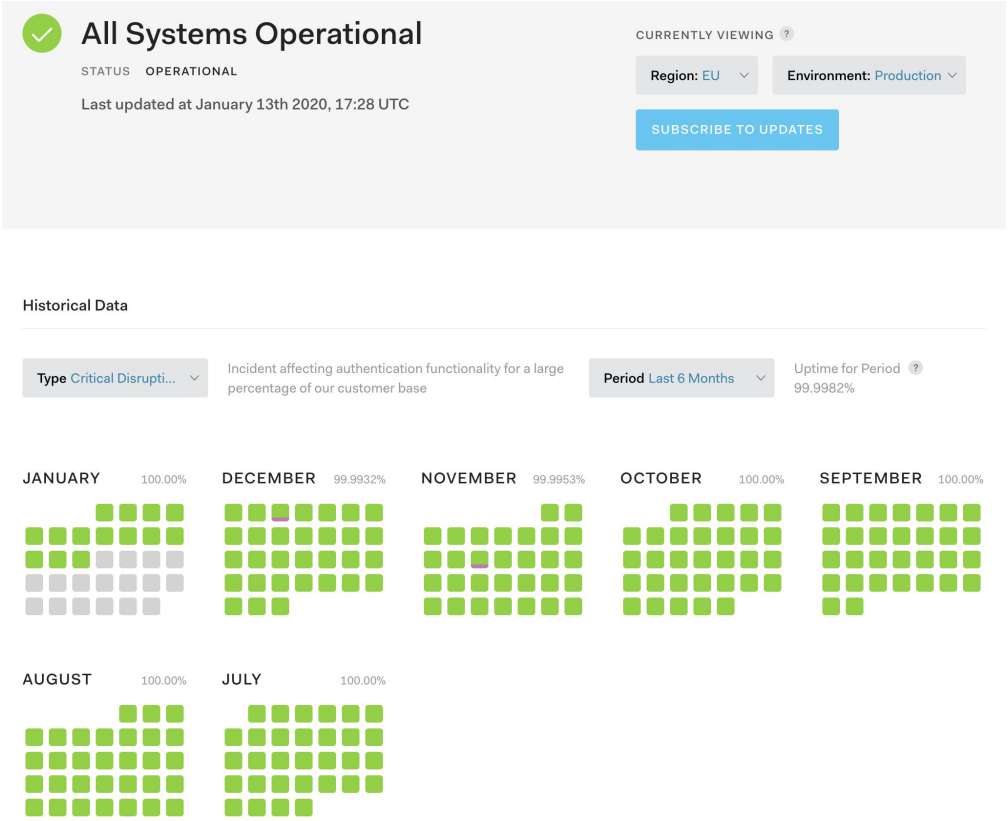
Establish a baseline and monitor for any spikes or unusual behaviour. The actual user behaviour depends on geographic distribution and business specifics. Monitor both success and failure events.



Monitoring Auth0 services

- Subscribe to Auth0 status page
- Periodically call “/test” and “/testall” endpoints
- Monitor external dependencies like APIs called from scripts or the email provider

https://auth0.com/docs/monitoring



Auth0 Training

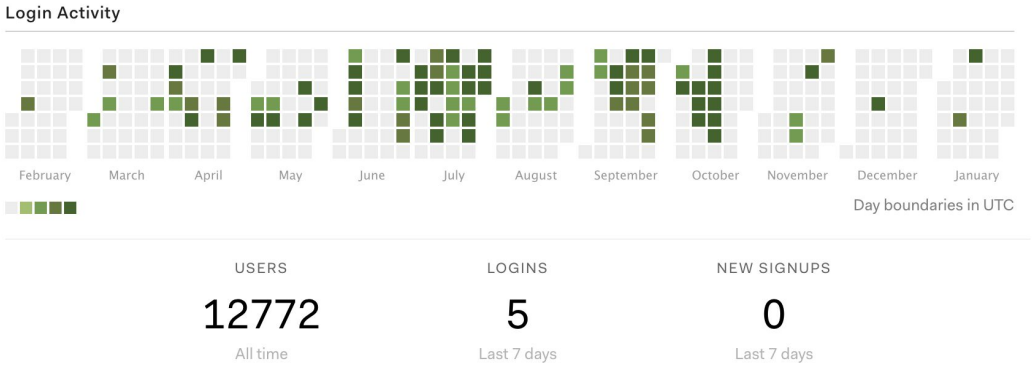
8.2. User DB maintenance

Maintenance of the user base

General maintenance

Not much to do here:

- Inactive accounts do not affect utilization quota
- DO NOT delete user databases
- Educate users for security best practices
- If possible - integrate MFA
- Periodic cleanup jobs can be implemented depending on business requirements



Data analysis

Use bulk export:

- Can be slow for big user bases - consider building “daily reports”
- Bulk import with “upsert=true” can be used for updates
- Combine user data with logs for detailed analysis


```
{
  "email": "admin@dae.com",
  "updated_at": "2020-01-13T16:43:14.780Z",
  "user_id": "auth0|5ccc386245b82c1101294d71",
  "identities": [
    {
      "user_id": "5ccc386245b82c1101294d71",
      "provider": "auth0",
      "connection": "Dae-users",
      "isSocial": false
    }
  ],
  "created_at": "2019-05-03T12:47:30.635Z",
  "last_password_reset": "2019-10-14T11:58:44.031Z",
  "app_metadata": {
    "roles": [
      "Delegated Admin - Administrator"
    ]
  },
  "last_ip": "109.193.224.177",
  "last_login": "2020-01-13T16:43:14.780Z",
  "logins_count": 11,
  ...
}
```

Auth0 Training

8.3. Troubleshooting

When things are not as expected

Initial diagnostics

Determine the scope of the issue:

1. Is only one user having the issue? Some? All of them?
2. Is it app-specific?
3. Are there clear steps to reproduce? Whenever possible - browser screenshots should include the URL of the loaded page.
4. If possible - find out more about user's environment (device details, OS, browser, etc)

https://auth0.com/docs/troubleshoot

User-specific issues

1. Check user profile state: Is it blocked? Is the profile in the expected state? Does it contain all required data?
2. Search for logs - when searching by user email - it must be lowercase.
3. If possible - try to reproduce the issue together with the user (via screen sharing).
4. A network request trace can help a lot in understanding what's going on (generated by browser dev tools).
5. If possible - check what browser the user is using and what is the environment. For example a corporate network may enforce a proxy which blocks cookies.

App-specific issues

1. Is the app correctly configured in Auth0? Check enabled connections, callbacks and protocols. Most of the time incorrect configuration will generate corresponding log messages.
2. Is the app sending the expected parameters? Check the initial call to /authorize and inspect all parameters.
3. Is the app handling the response correctly? If the app is not able to establish a user session - it can cause an infinite “authentication redirect” loop.
4. Is the app maintaining the state correctly? Lack of caching can cause too many requests and hit API rate limits.

Universal Login

1. Ensure there are no javascript errors - some browsers stop processing on first unhandled exception.
2. Verify the configuration - Custom Domains require additional configuration in Universal Login Classic.
3. Are the page scripts using some features unsupported in certain browsers?
4. During authentication, the Universal Login Classic performs several requests in a row - ensure they are not interrupted. Also incorrect implementation can cause the login form to be submitted multiple times.
5. Lack of user session will cause the silent authentication to fail.

Emails

1. Are the email templates correctly configured? Also some templates can be disabled (like welcome email).
2. Links like password reset and email verification are one-time use only.
3. Ensure there are log messages that the email was sent to the provider.
4. Inspect the email provider logs to ensure that the email was delivered to user's inbox.

Webtasks

If anything goes wrong:

- Check tenant logs for odd messages - unhandled exceptions end there.
- Use Realtime Webtask Logs extension to troubleshoot custom code.
- Make sure the callbacks are called exactly once.

```
function bad(user, context, callback) {
  // visible through Realtime Webtask Logs
  console.log("Start rule...");
  if (user.email === "blah@blah.com") {
    // bad, callback will be called twice
    callback(new UnauthorizedError("Access denied"));
  }
  callback(null, user, context);
}

function good(user, context, callback) {
  if (user.email === "blah@blah.com") {
    // callback called only once due to "return"
    return callback(new UnauthorizedError("Access denied"));
  }
  callback(null, user, context);
 }
```

APIs and Rate Limits

Any calls from rules/hooks/custom DB scripts to Auth0 APIs are still subject to API rate limits:

- If possible, minimize calls to Management API from rules.
- If user profile update is needed - make sure to combine multiple calls into one.
- Always plan in advance if there are expected spikes in user logins (events similar to Black Friday).
- When calling Auth0 APIs, inspect the HTTP Response headers for current rate limits - this is useful when implementing request throttling.

8.4. Security issues

Should not happen, but it doesn’t hurt to be prepared

You are not on your own

- Auth0 will always help reviewing any security concerns - being it a suspected security issue, security testing results or compliance questions.
- If the issue affects the users in production - raise a support ticket with the “urgent” priority so that the support engineers get notified immediately.
- Provide as much detail as possible and try to describe what is the goal that needs to be achieved, rather than the solution.
- Auth0 internal platform logs can help with the investigation.

Depending on the affected area, different options are possible - on next slides.

Individual user account

When an individual user account is affected:

- It can be disabled.
- User profile can be edited through Management Dashboard or API to correct its state.
- User-related logs will contain the information about the source IP and the User agent, which may provide additional insight about the user behaviour.
- Revoke user's refresh tokens (if any).

An application

When an application is affected:

- Rotate the secret for confidential clients.
- Disable the connections to stop users from logging in.
- Ensure there are no machine-to-machine clients with unneeded scopes for Management API.
- Inspect the tenant logs for any suspicious behaviour.

Extensibility - setup

Incorrect setup can cause data leaks or allow for MiTM attacks:

- Ensure only required values are added as custom claims. Avoid adding the entire app_metadata structure as a custom claim.
- All API calls must use HTTPS.
- External APIs must use at least some form of caller authentication - an API key or something similar.
- It is a good idea to have API rate limits implemented and to allow the calls only from Auth0-specific IP addresses.
- Assume users will always tamper with client-side code and never trust any parameters passing through user's browser.
- Do not forward external input from your API to Auth0 without validation.

Extensibility - mitigation

Extensibility can also be used to reduce the impact of the issues:

- A rule can stop users from logging in - it can be done by app, by user account or by any other criteria that can be computed from the available context in the rule. For example it can stop from logging in all users registered after a certain date.
- Client credentials exchange hook can inspect and block M2M calls.
- Pre-user registration hook can perform server-side data validation.
- A conditional “Redirect from rules” can enforce captcha for suspicious requests or enforce collection of additional data.

Auth0 Training

8.5. Automated settings deployment

CI/CD integration

Deploy Options

There are three options for updating your tenant other then the dashboard

- Use the management API directly.
- Source control extensions. We provide extensions that automatically update tenant configurations when called by its webhook. Deploys when a new commit is detected.
- Deploy Command Line Interface. This CLI tool can be called manually or programmatically and allows both importing and exporting settings from a tenant. Output can be either to a YAML file or directory structure.

Ultimately all options leverage the Management API under the covers so keep rate limits in mind.

Source control extensions

- Configured under the extensions area of the Auth0 dashboard. You will need to provide the extension information about your repository including url, base folder, and branch. You will be provided information to configure the source control system after saving these.
- In SCM you will have to configure a webhook with the information (url, content-type, secret) provided by the extension.
- Your repo should include artifacts as files in these folders: clients, grants, emails, resource-servers, connections, database-connections, rules-configs, rules, pages
- Any rules or database connection scripts that exist in Auth0 but not in your repository will be deleted. The extension can be configured to exempt certain rules from this.
- Extension allows for encrypting secrets in files.
- Extension provides a history of deployments successful or otherwise.

Auth0 Deploy CLI

- Auth0 supports integration into existing CI/CD pipelines using the open-source Deploy CLI tool.
- Tool allows import and export of Tenant settings, Rules (including secrets/settings), Connections, Custom databases, Clients/applications, Resource servers (APIs), Pages, Email templates and providers, and Guardian settings
- Tool can be called programmatically.
- Tenant settings can be import/exported to/from a YAML config file or a directory structure.
- This tool can be destructive to your Auth0 tenant. Please ensure you have read the documentation and tested the tool on a development tenant before using it in production.
- The recommended approach for utilizing the CLI is to incorporate it into your build system
    - Create a repository to store your deploy configuration, then create a set of configuration files for each environment.
    - On your continuous integration server, have a deploy build for each environment.
    - This deploy build should update a local copy of the deploy configuration repository, then run the CLI to deploy it to that environment.

Installing Auth0 Deploy CLI

- The deploy cli runs as a non-interactive application that uses a M2M token to call the Auth0 Management API. This application must be created in the Auth0 Dashboard.
- The application can be created manually and assigned the appropriate authorization to call the management api.
- An easier option is add the cli repo as an extension in the dashboard. This will automatically create the necessary application and authorize it for you.
- The CLI itself can be installed on the machines which will run it via npm

```
npm i -g auth0-deploy-cli
```

Calling Auth0 Deploy CLI

- `a0deploy import` - Imports tenant settings from YAML/Directory
- `a0deploy export` - Exports tenant settings from YAML/Directory
- Configuration and environment specific variables can be put in a config.json file

```
{
 "AUTH0_DOMAIN": "mattb-deploycli.auth0.com",
 "AUTH0_CLIENT_ID": "client id of application deploy cli runs as",
 "AUTH0_CLIENT_SECRET": "client secret of application deploy cli runs as, better practice is to pass
this in with -x parameter when calling cli",
 "AUTH0_KEYWORD_REPLACE_MAPPINGS": { "environment_specific_variable_1": "value_1" },
 "AUTH0_ALLOW_DELETE": false, //if true delete objects that don't exist in the deploy
 "AUTH0_EXCLUDED_RULES": [ "rule-1-name" ],
 "AUTH0_EXCLUDED_CLIENTS ": [ "client-1-name" ],
 "AUTH0_EXCLUDED_RESOURCE_SERVERS": [ "resourceServer-1-name" ]
}
```

- Environment variables can be injected as a literal value with `@@key@@` or via string interpolation with `##key##`.

Calling Auth0 Deploy CLI

- Export to YAML: `a0deploy export -c <your config json> -f yaml -o <your repo directory> -x <client secret>`
- Import from YAML: `a0deploy import -c <config.json> -i <tenant.yaml> -x <client secret>`
- Export to Directory: `a0deploy export -c <your config json> -f directory -o <your repo directory> -x <client secret>`
- Import from Directory: `a0deploy import -c <config.json> -i <path/to/files> -x <client secret>`
- Other Options
  - `--help` Show help [boolean]
  - `--version` Show version number [boolean]
  - `--verbose, -v` Dump extra debug information. [string] [default: false]
  - `--proxy_url, -p` A url for proxying requests, only set this if you are behind a proxy. [string]

Calling Auth0 Deploy CLI Programmatically

```
import { deploy, dump } from 'auth0-deploy-cli';

const config = {
  AUTH0_DOMAIN: process.env.AUTH0_DOMAIN,
  AUTH0_CLIENT_SECRET: process.env.AUTH0_CLIENT_SECRET,
  AUTH0_CLIENT_ID: process.env.AUTH0_CLIENT_ID,
  AUTH0_ALLOW_DELETE: false
};


// Export Tenant Config
dump({
  output_folder: 'path/to/yaml/or/directory',   // Input file for directory
  base_path: basePath,                          // Allow to override basepa
  config_file: configFile,                      // Option to a config json
  config: configObj,                            // Option to sent in json as
  strip,                                        // Strip the identifier fie
  secret                                        // Optionally pass in auth0
})
  .then(() => console.log('yey dump was successful'))
  .catch(err => console.log(`Oh no, something went wrong. Error: ${err}`));
```

```
// Import tenant config
deploy({
  input_file: 'path/to/yaml/or/directory',  // Input file for directory, cha
  base_path: basePath,                      // Allow to override basepath,
  config_file: configFile,                  // Option to a config json
  config: configObj,                        // Option to sent in json as obj
  env,                                      // Allow env variable mappings
  secret                                    // Optionally pass in auth0 cli
})
  .then(() => console.log('yey deploy was successful'))
  .catch(err => console.log(`Oh no, something went wrong. Error: ${err}`));
```

End of Chapter 8

9. Hands-on Labs

Let’s try it in practice!

Thanks.

auth0.com