Red Hat API Management

API Security with OpenID Connect

Presenter

Title

Date

CUSTOMER MOVING TO CONTAINERS?

Customer Benefits

- Breaking down the monolith means faster turnaround.
- Greater flexibility.
- Fully automated testing driving better QA.
- Free from single technology limits
- Makes customer agile, cloud native, reactive compliant, with bounded security and fully aggregate enabled!

REMEMBER SECURITY

Everything Has To Be Super Secure

- Independent services that are no longer bound to one monolith i.e. just one security context on one cluster or multiple ones ?
- So in microservices - does each service have to store identity?
- Ownership of identity data is a risk if not done properly.

MORDAC, THE PREVENTER OF INFORMATION SERVICES.
SECURITY IS MORE IMPORTANT THAN USABILITY.
www.dilbert.com
scottadams@aol.com
IN A PERFECT WORLD, NO ONE WOULD BE ABLE TO USE ANYTHING.
©2007 Scott Adams, Inc./Dist. by UFS, Inc.
11-6-07
To complete the log-in procedure, stare directly at the sun.

CSO IS ON THE SPOTLIGHT

Recent News On Data Breaches Is Not A Good Start

77,000,000 PSN RECORDS STOLEN

Sony

143,000,000 ACCOUNTS

Equifax

117,000,000 RECORDS STOLEN

LinkedIn

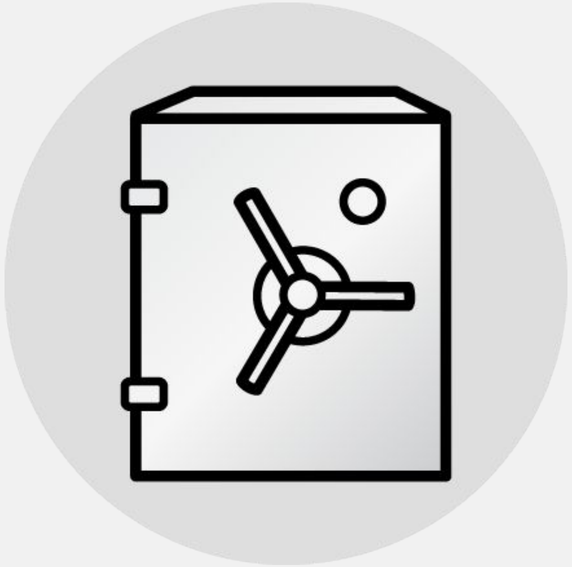
DEVELOPERS HEADACHE

Homemade Security Is Always A Recipe For Disaster

- The bar for appdev security has been raised substantially.
- Most devs don’t really ‘do’ security.
    - E.g. Storing user information - should be as salted hash.
- The hand rolled solutions of the past are not viable going forwards.
- Are you still sure you want to store ‘identity’ in your App?

PROBLEM WAS SOLVED WHILE AGO

Banks

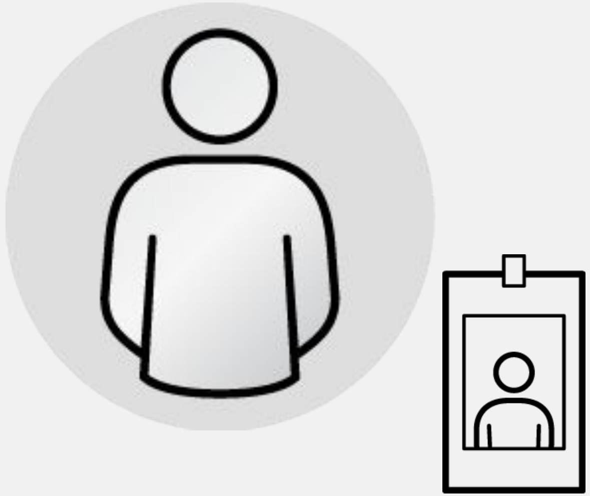


- Most people do not keep their savings under their mattress for the same security reasons.
- They store them in a trusted environment - i.e. the bank.
- Then use a separate means to authorise / authenticate payments.

API SECURITY WORKS IN THE SAME WAY

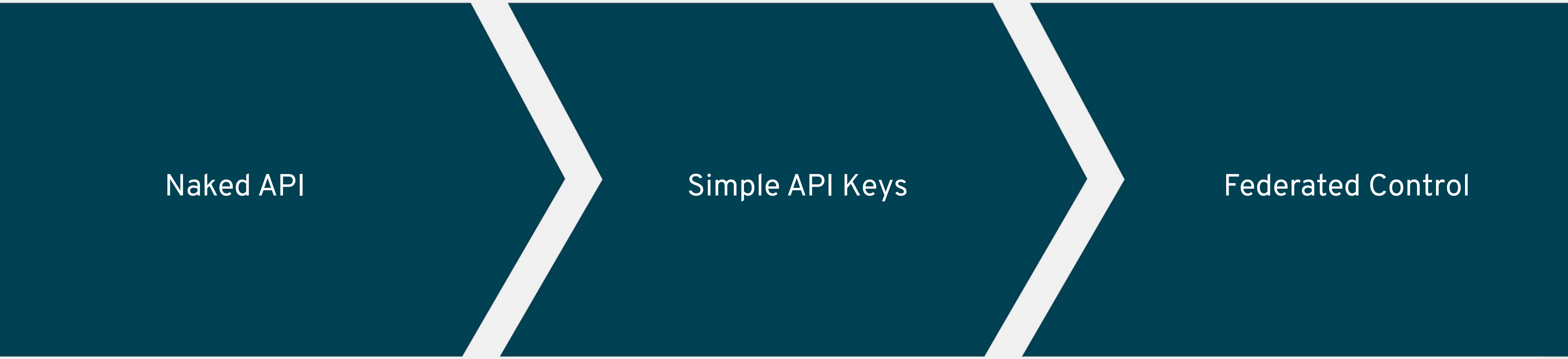
Trusted Identity Authority

- Identity is abstracted away to a single ‘fortress’.
- Tokens are issued as a means of allowing access to services and resources.
- So now your application services no longer have to store sensitive data!



API SECURITY

Evolution of API Security



The Authentication Granddaddy - Basic Auth

API SECURITY

Top Schemes

Most API Management platforms supports the following security schemes:

- **API Key** single token string
- **APP ID/APP Key (Basic Auth)** two token strings i.e. username, password
- **OpenID Connect (OIDC)** simple identity layer on top of OAuth framework

OAUTH

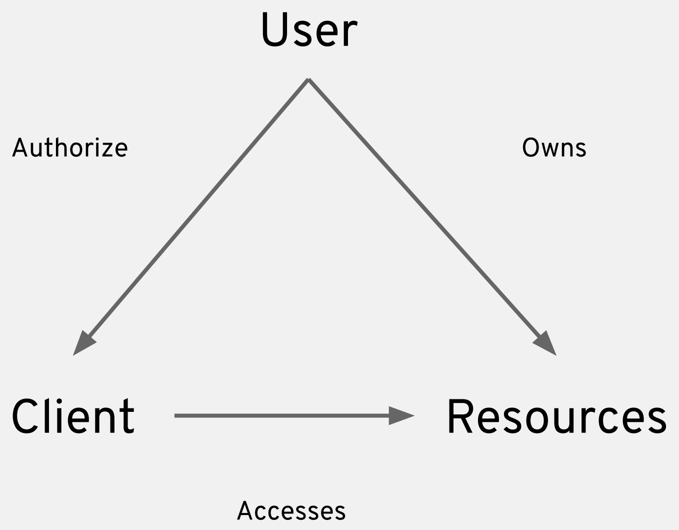
OAUTH 2.0

From 20,000 FT

OAuth (Open Authorization) is an open standard for access delegation:

- One service can request access to resources on another service on the behalf of the user.

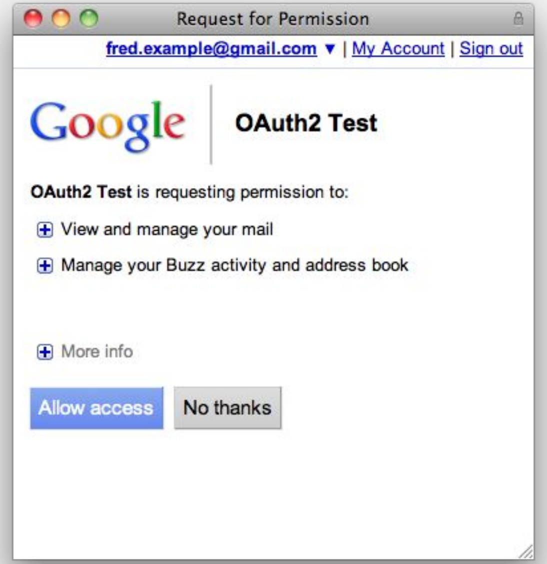
Published October 2012



OAUTH 2.0

Delegation

OAuth enables users to grant third-party access to their web resources without sharing their passwords.


Request for Permission

fred.example@gmail.com ▼ | My Account | Sign out

Google | OAuth2 Test

OAuth2 Test is requesting permission to:

- View and manage your mail
- Manage your Buzz activity and address book

More info

OAUTH 2.0

Terminology

- **Resource Owner:** generally yourself.
- **Resource Server:** server hosting protected data (for example Google hosting your profile).
- **Client:** application requesting access to a resource server (i.e. a mobile application).
- **Authorization Server:** server issuing token to the client. This token will be used for the client to request the resource server.

OAUTH 2.0

Grant / Flow Types

Authorization Code Flow

The most secure and used where a user logs into Identity server and grants access to Application to retrieve their data

Client Credentials Flow

Only Application data is passed in a single request for an Access Token

Implicit Flow

User logs in but secret is not passed

Resource Owner Password Flow

Application, username and password data is passed in a single request for an Access Token

OPENID CONNECT

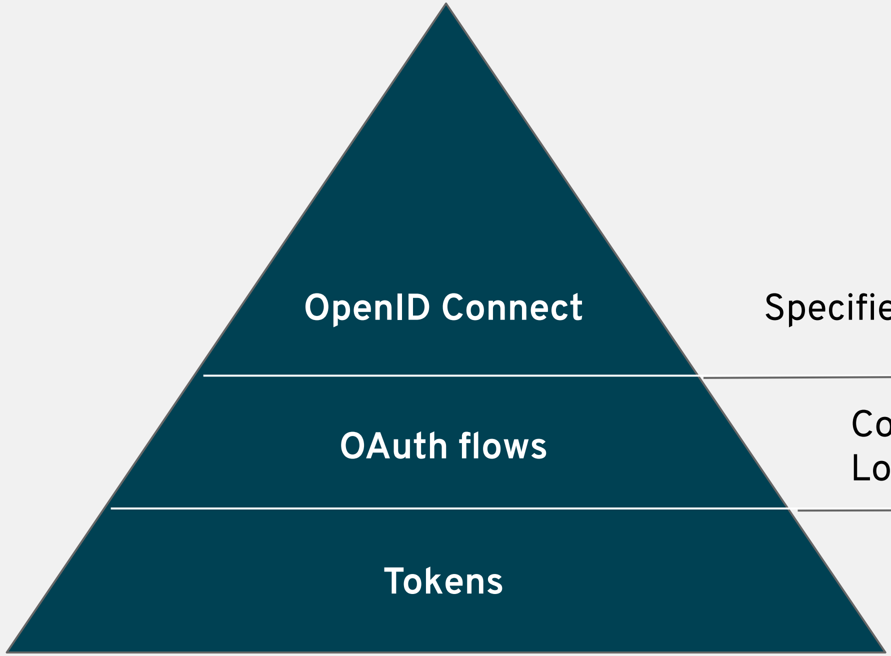
OPENID CONNECT

Overview

- Built on top of the OAuth 2.0 protocol
- Allows clients to verify the identity of an end user
- Obtains basic profile information about the end user
- RESTful HTTP API, using JSON as a data format
- Like SAML - but not just webpage centric, easier to implement.

OPENID CONNECT

Layered Security Standards



Specifies JWT for token, and some extensions

Core delegation flows.
Lots of flexibility (too much?)

Wide variety of token and encryption standards

OPENID CONNECT

Vs OAuth 2.0

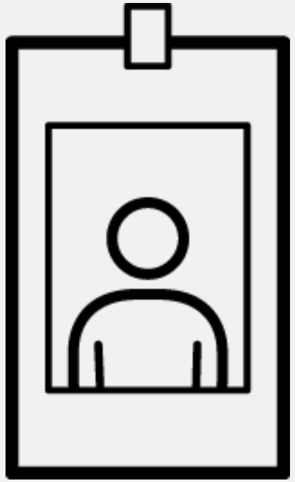
OpenID is an open standard for authentication. A user must obtain an OpenID account through an OpenID identity provider (for example, Google). The user will then use that account to sign into any website (the relying party) that accepts OpenID authentication.

OAuth2 is an open standard for authorization. Confusingly, OAuth2 is also the basis for OpenID Connect. OAuth2 provides secure delegated access, meaning that an application, called a client, can take actions or access resources on a resource server on the behalf of a user, without the user sharing their credentials with the application.

OPENID CONNECT

ID Token

- Provides identity information to the application from the Authority Server
- Base64 encoded - easy to work with.



Name: John Doe

Type: Employee

Expired by: Company

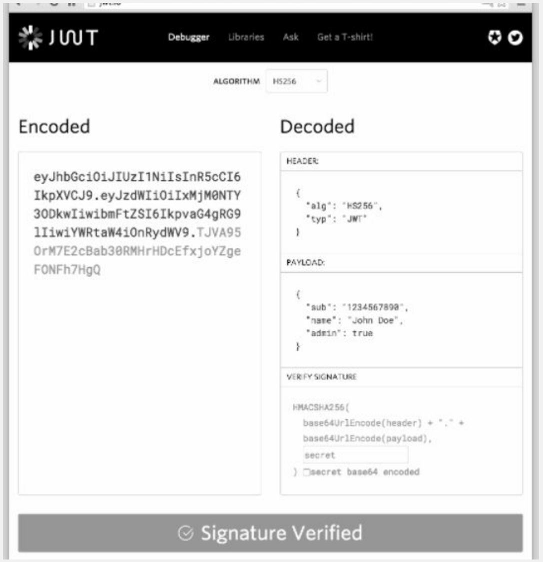
Expiration:

02-06-2019

JWT (“JOT”)

To The Rescue

- Signed by algo and verified by only correct key
- Contains user identity in form of claims (Private, public, reserved)
- For OIDC purpose, SSO is widely adopted in consumer/enterprise apps
- Eliminates the need to look up against a central access control list



ALGORITHM HS256

Encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6
IkpXVCJ9.eyJzdWIiOiIxMjM0NTY
3ODkwIiwibmFtZSI6IkpvaG4gRG9
lIiwiYWRtaW4iOnRydWV9.TJVA95
OrM7E2cBab30RMHrHDcEfxjoYZge
FONFh7HgQ
```

Decoded

HEADER:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

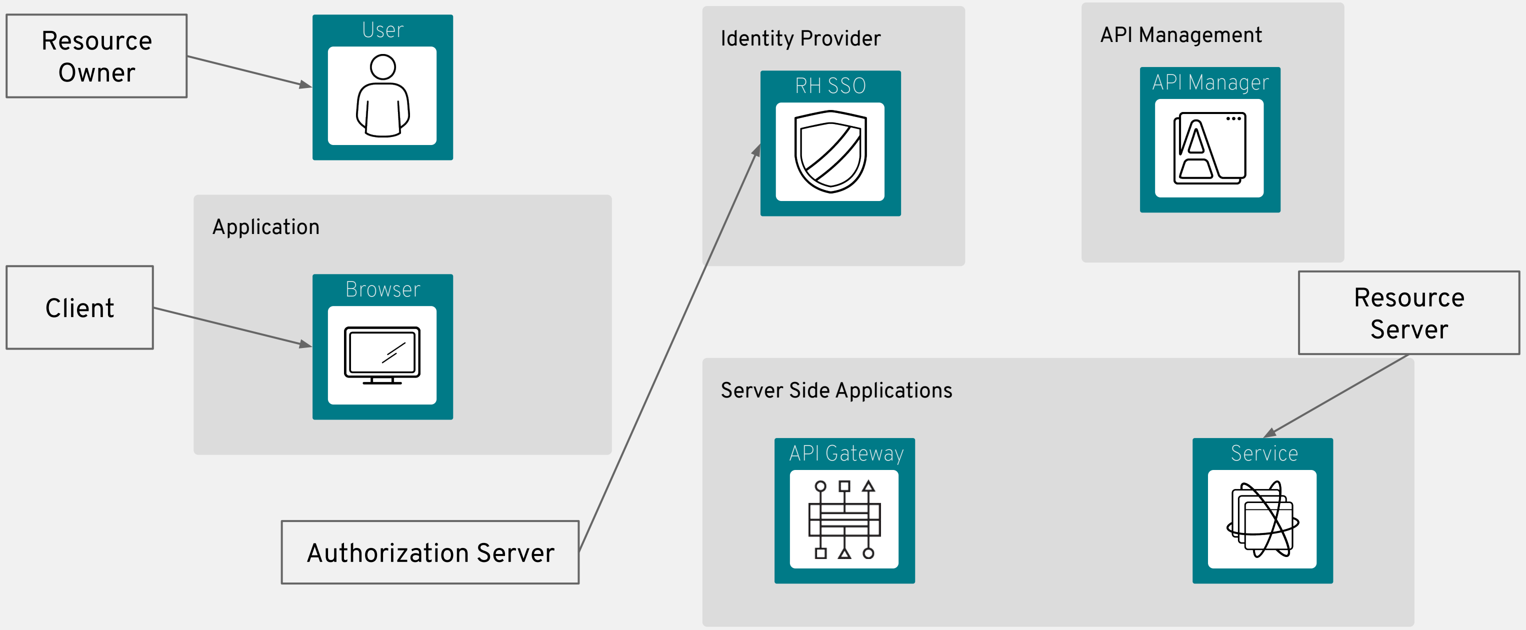
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) ☐secret base64 encoded
```

Signature Verified

AUTHORIZATION CODE FLOW COMPLETE EXCHANGE

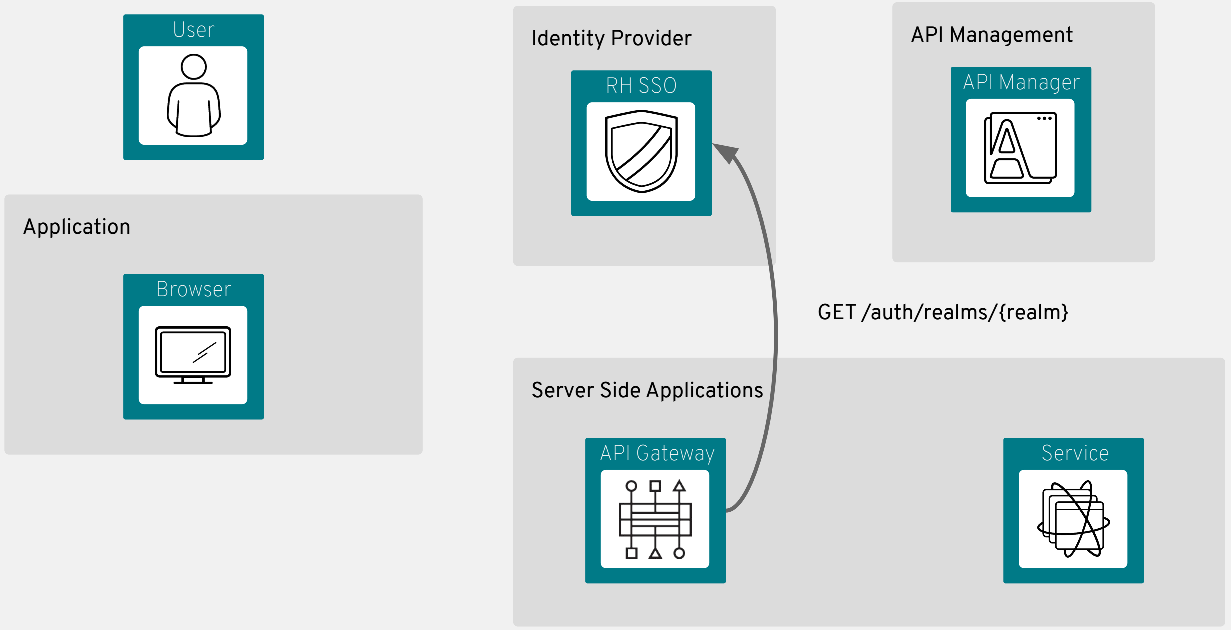
AUTHORIZATION CODE FLOW

An Orientation



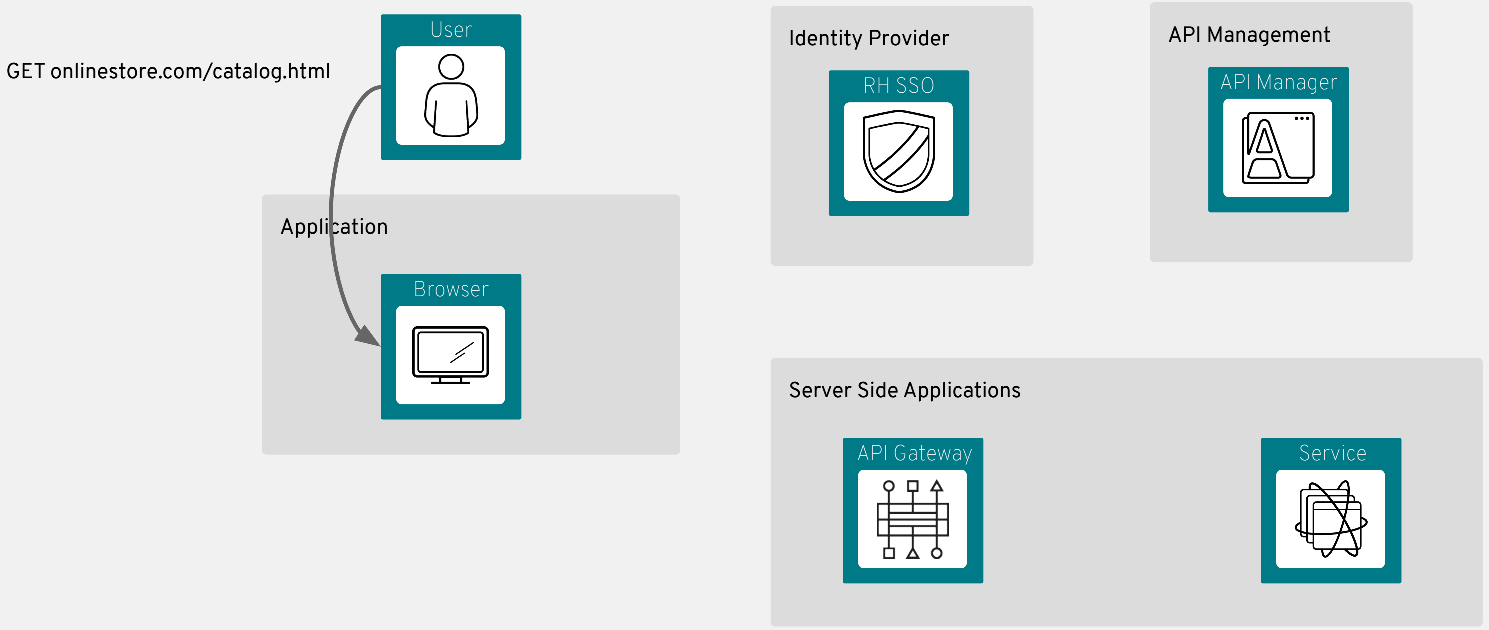
AUTHORIZATION CODE FLOW

#0 - 3scale API Gateway Gets RH SSO Public Key On Configuration Load



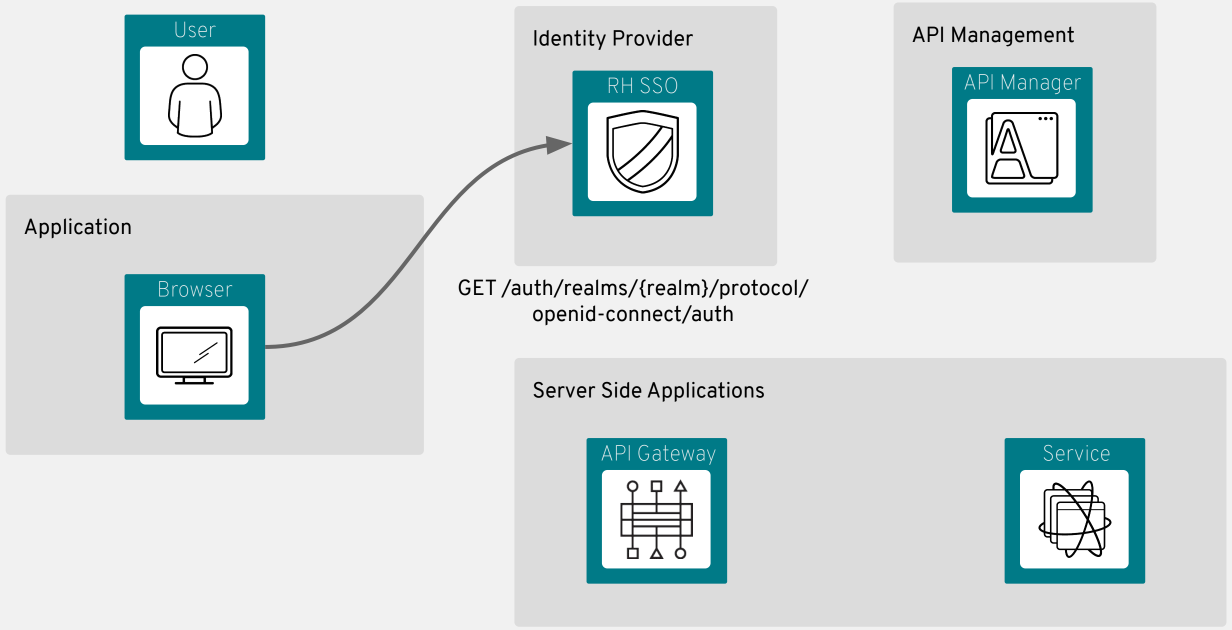
AUTHORIZATION CODE FLOW

#1 - User Starts Using The Web App



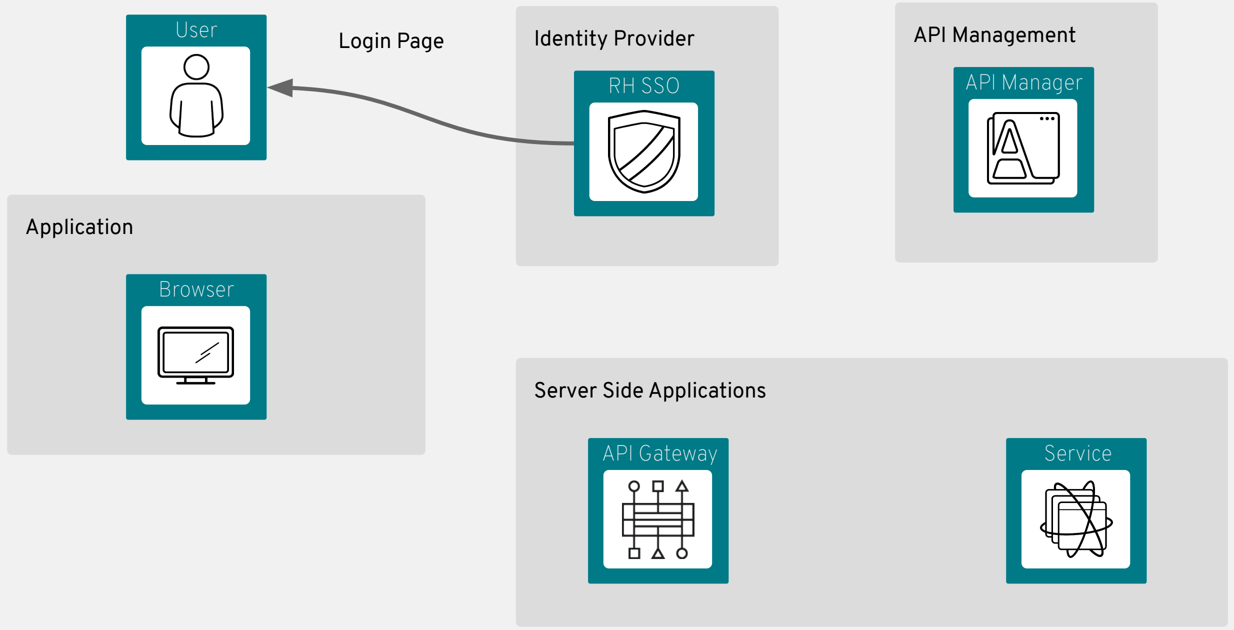
AUTHORIZATION CODE FLOW

#2 - The Application Introduces RH SSO



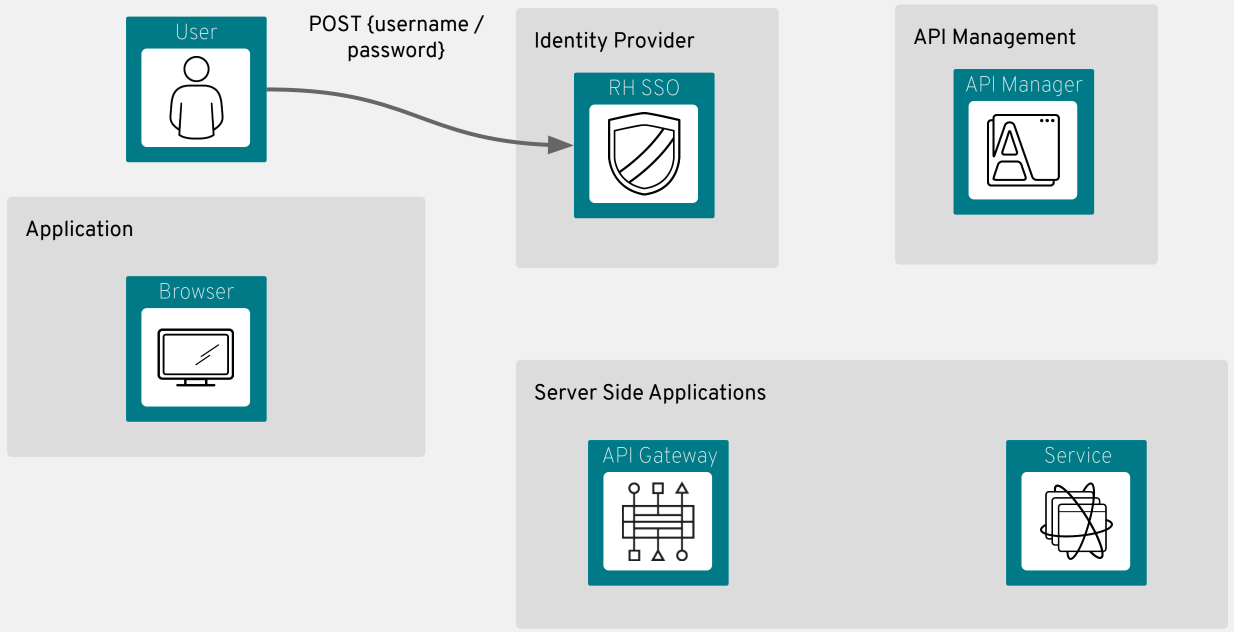
AUTHORIZATION CODE FLOW

#3 - RH SSO Forwards To Login Form



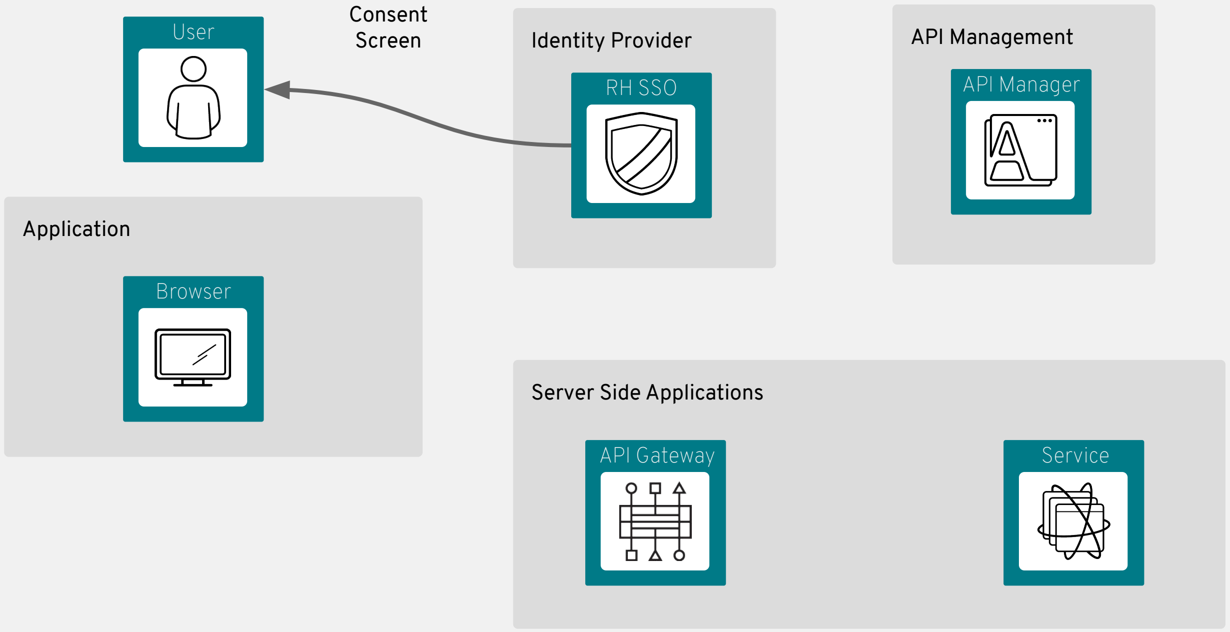
AUTHORIZATION CODE FLOW

#4 - The User Logs Into RH SSO



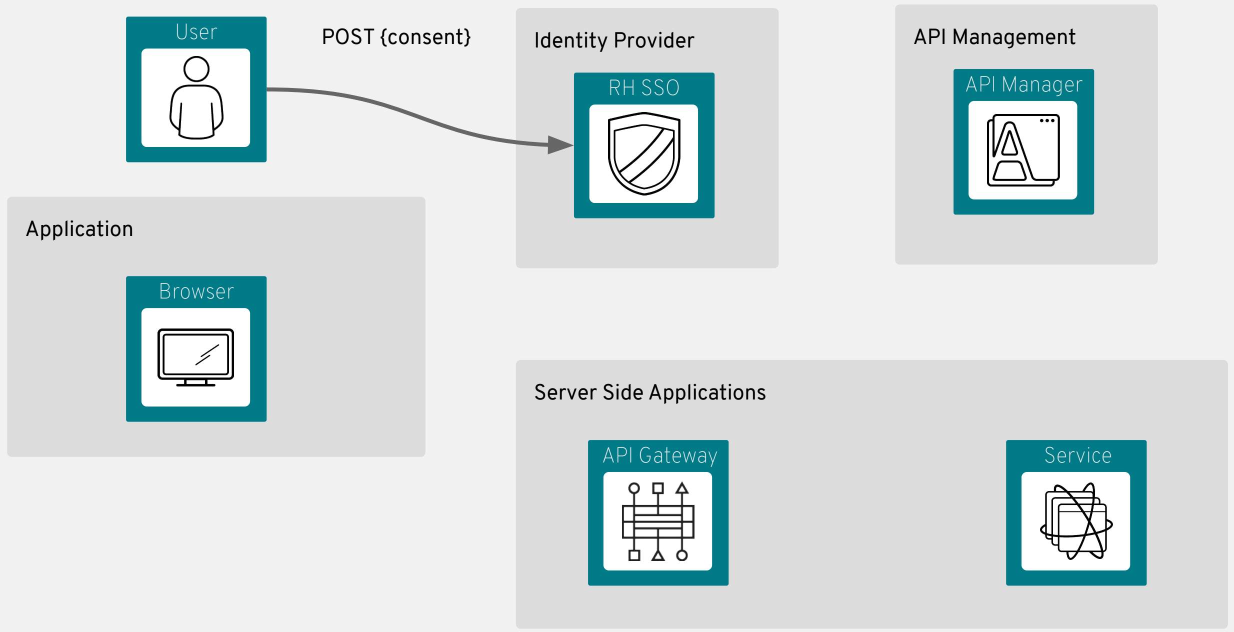
AUTHORIZATION CODE FLOW

#5 - RH SSO Forwards To Consent Page



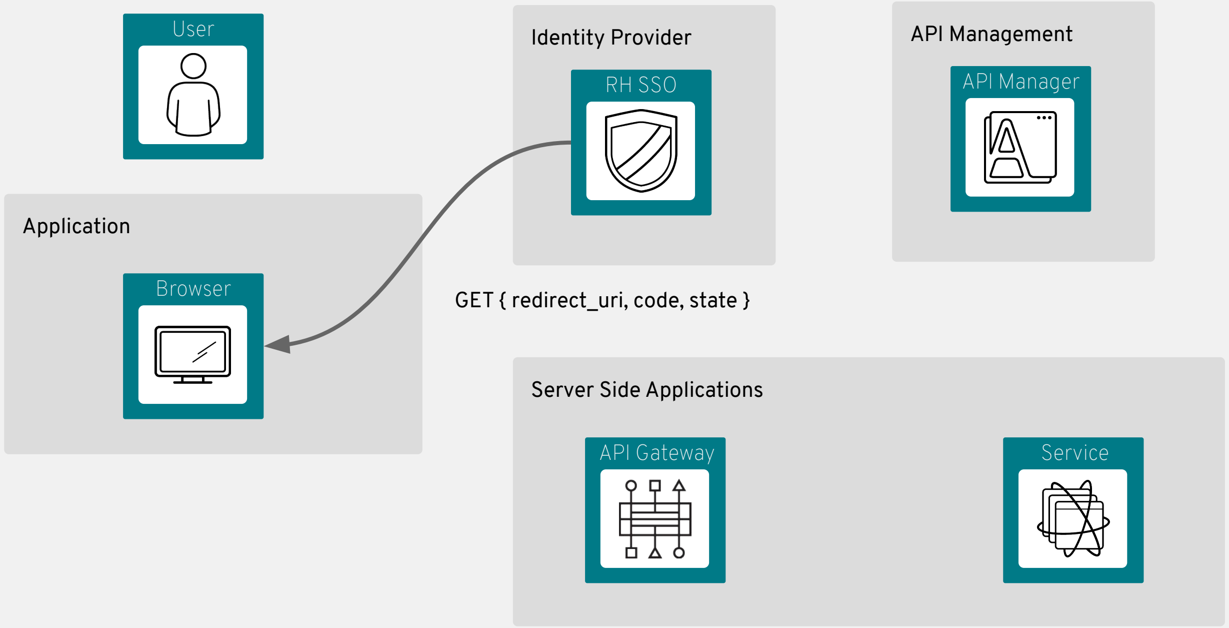
AUTHORIZATION CODE FLOW

#6 - The User Consents



AUTHORIZATION CODE FLOW

#7 - RH SSO Redirects To Application And Sends An Auth Code



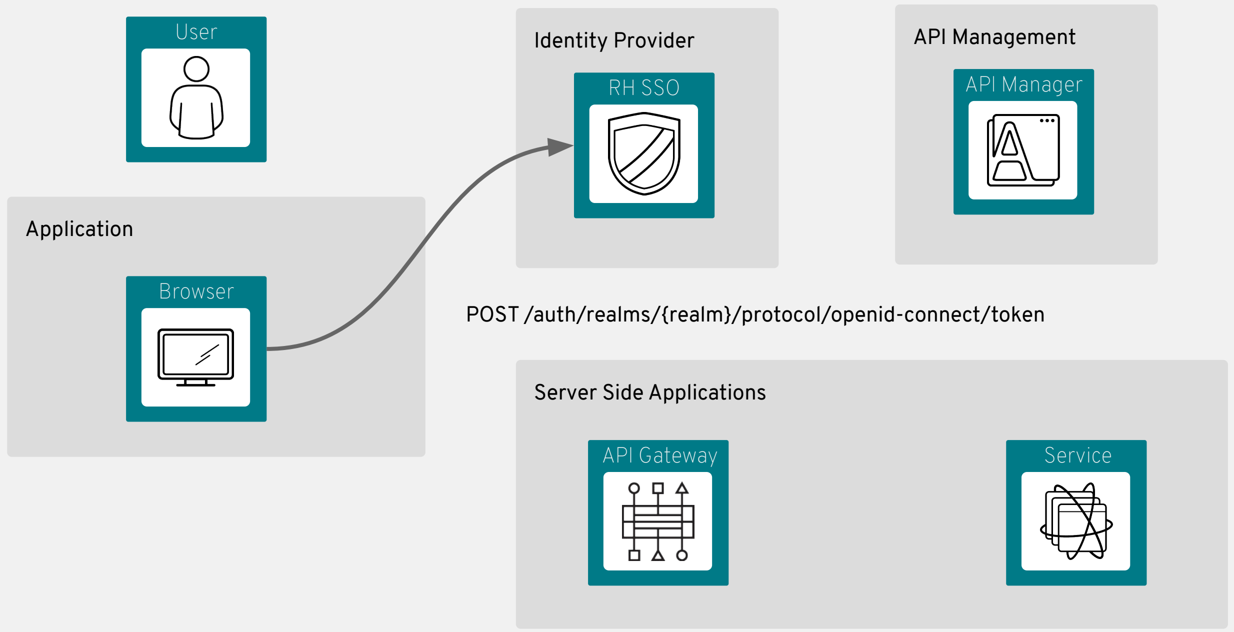
AUTHORIZATION CODE FLOW

#7.1 - The Temp Auth Code

- Is used to acquire an access code.
- Think of this as being a cloakroom ticket - this can be used once only to acquire a bearer token.

AUTHORIZATION CODE FLOW

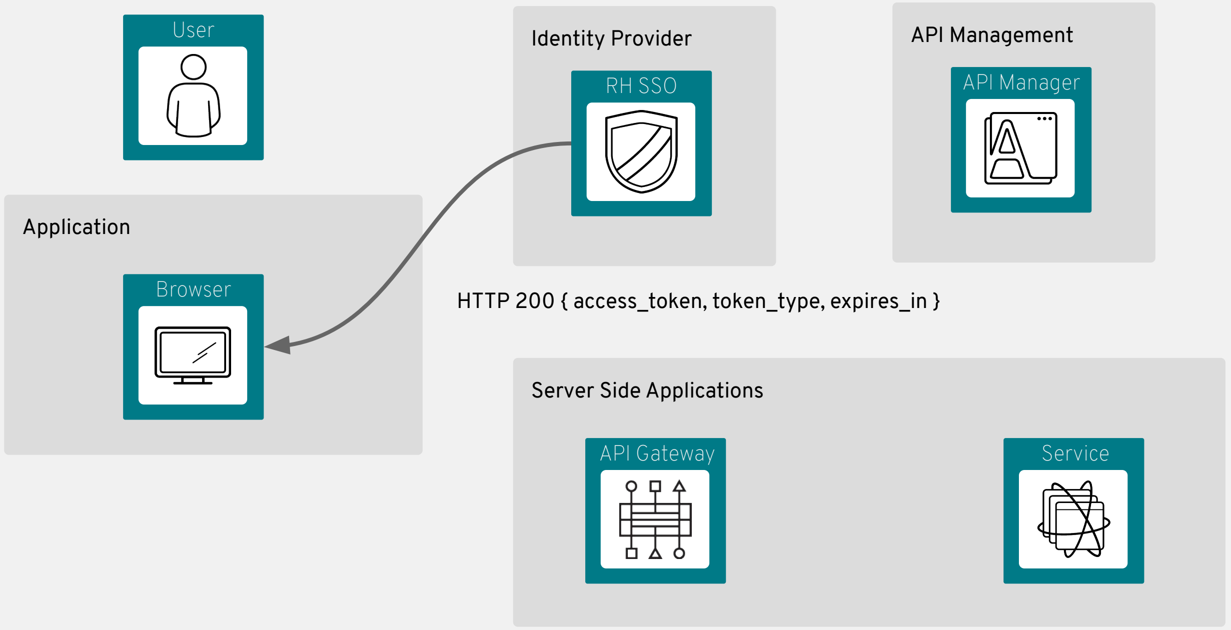
#8 - The Web App Calls The Token Endpoint



POST /auth/realms/{realm}/protocol/openid-connect/token

AUTHORIZATION CODE FLOW

#9 - RH SSO Sends A Valid Bearer Token



AUTHORIZATION CODE FLOW

#9.1 - The Bearer Token

"A security token with the property that any party in possession of the token (a "bearer") can use the token in any way that any other party in possession of it can"

AUTHORIZATION CODE FLOW

#9.2 - The Bearer Token

```
Authorization: Bearer
QXV0aG9yaXphdGlvbjogQmVhcmVyIA0Kew0KICJqdGkiOiAiYmNiMTFmNDktZTZhZS00NGNhLWIwNzctMzc5MjU5NGYwZDk4IiwN
CiAiZXhwIjogMTQ5NTI3MjczOSwNCiAibmJmIjogMCwNCiAiaWF0IjogMTQ5NDMyMjMzOSwNCiAiaXNzIjogImh0dHA6Ly8wOTY2
ZWExZi5uZ3Jvay5pby9hdXRoL3JlYWxtcy9mb3VybWFya3MiLA0KICJhdWQiOiAiNGQ2NTI0MDYiLA0KICJzdWIiOiAiZDIwZGM0
MTUtNzUyZi00YTc5LWEzYTgtNTJlOTVlYTZkZWM2IiwNCiAidHlwIjogIkJlYXJlciIsDQogImF6cCI6ICI0ZDY1MjQwNiIsDQog
InNlc3Npb25fc3RhdGUiOiAiNTVhODQzMjktY2Y2ZC00YjliLWJhOGYtYWJhMDM3NjRjMjFjIiwNCiAiY2xpZW50X3Nlc3Npb24i
OiAiYmYxYTA3MzktYTM5Yy00NTE1LTljMDAtNzhlMTgyNmI4ZDM2IiwNCiAiYWxsb3dlZC1vcmlnaW5zIjogWw0KICAiaHR0cHM6
Ly93d3cuZ2V0cG9zdG1hbi5jb20iDQogXSwNCiAicmVhbG1fYWNjZXNzIjogew0KICAicm9sZXMiOiBbDQogICAiYWNjZXNzX215
X3Jlc291cmNlIg0KICBdDQogfSwNCiAicmVzb3VyY2VfYWNjZXNzIjogew0KICAiYWNjb3VudCI6IHsNCiAgICJyb2xlcyI6IFsN
CiAgICAibWFuYWdlLWFjY291bnQiLA0KICAgICJ2aWV3LXByb2ZpbGUiDQogICBdDQogIH0NCiB9LA0KICJuYW1lIjogInRlc3Qg
dXNlciIsDQogInByZWZlcnJlZF91c2VybmFtZSI6ICJ0ZXN0dXNlciIsDQogImdpdmVuX25hbWUiOiAidGVzdCIsDQogImZhbWls
eV9uYW1lIjogInVzZXIiLA0KICJlbWFpbCI6ICJ0ZXN0QGJsYWguY29tIg0KfQ0K
Accept: */*
Postman-Token: 86b86d4a-8369-40af-8612-9f0d3589fdfb
Cf-Ray: 35c3a94bb1ac35ae-LHR
X-3Scale-Proxy-Secret-Token: Shared_secret_sent_from_proxy_to_API_backend_169ad455fe40801e
```

What does a bearer token look like?

AUTHORIZATION CODE FLOW

#9.3 - The Bearer Token

if you base64 decrypt you get:

notice the role information

the token is a JWT.

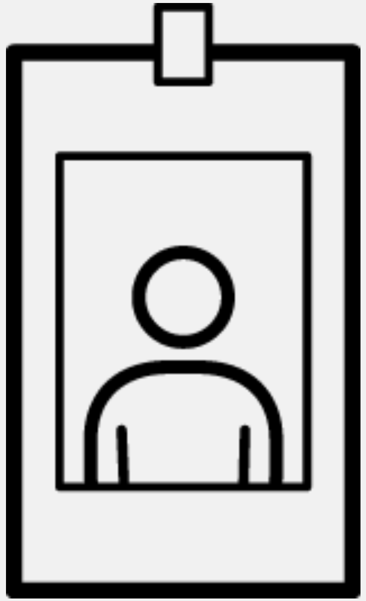
```
Authorization: Bearer
{
 "jti": "bcb11f49-e6ae-44ca-b077-3792594f0d98",
 "exp": 1495272739,
 "nbf": 0,
 "iat": 1494322339,
 "iss": "http://0966ea1f.ngrok.io/auth/realms/fourmarks",
 "aud": "4d652406",
 "sub": "d20dc415-752f-4a79-a3a8-52e95ea6dec6",
 "typ": "Bearer",
 "azp": "4d652406",
 "session_state": "55a84329-cf6d-4b9b-ba8f-aba03764c21c",
 "client_session": "bf1a0739-a39c-4515-9c00-78e1826b8d36",
 "allowed-origins": [
  "https://www.getpostman.com"
 ],
 "realm_access": {
  "roles": [
   "access_my_resource"
  ]
 },
 "resource_access": {
  "account": {
   "roles": [
    "manage-account",
    "view-profile"
   ]
  }
 },
 "name": "test user",
 "preferred_username": "testuser",
 "given_name": "test",
 "family_name": "user",
 "email": "test@blah.com"
}
```

AUTHORIZATION CODE FLOW

#9.4 - The Bearer Token

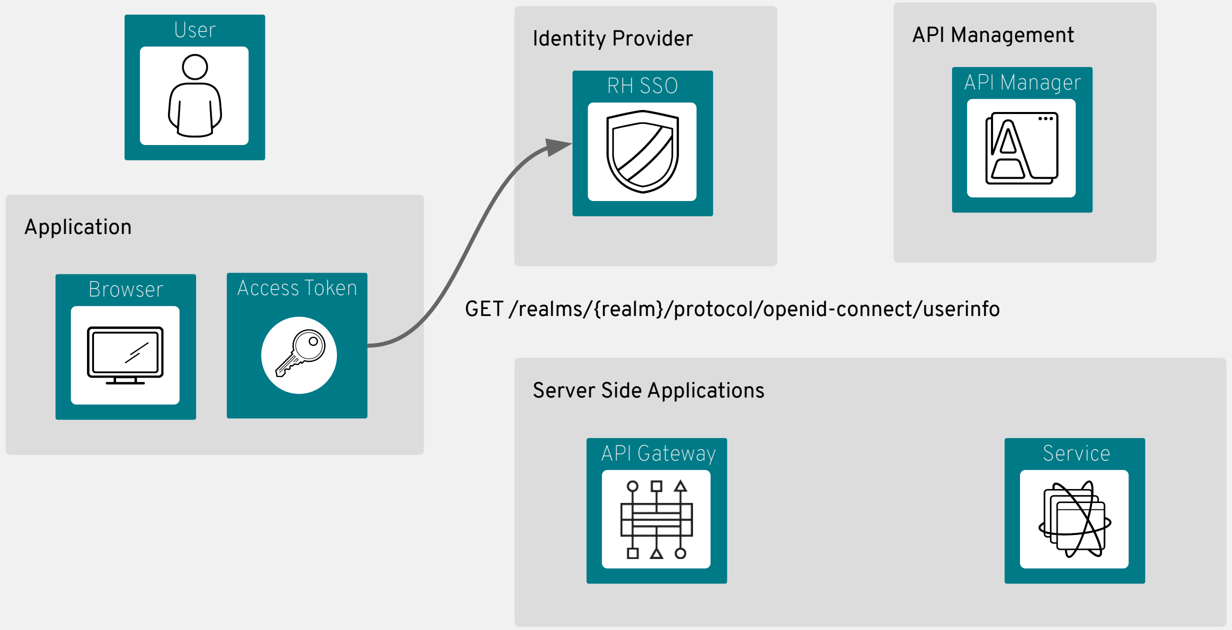
- Digitally signed by the Auth Server.
- A Standardised Identity token.
- Contains the username and roles, but

  can also add custom claims.



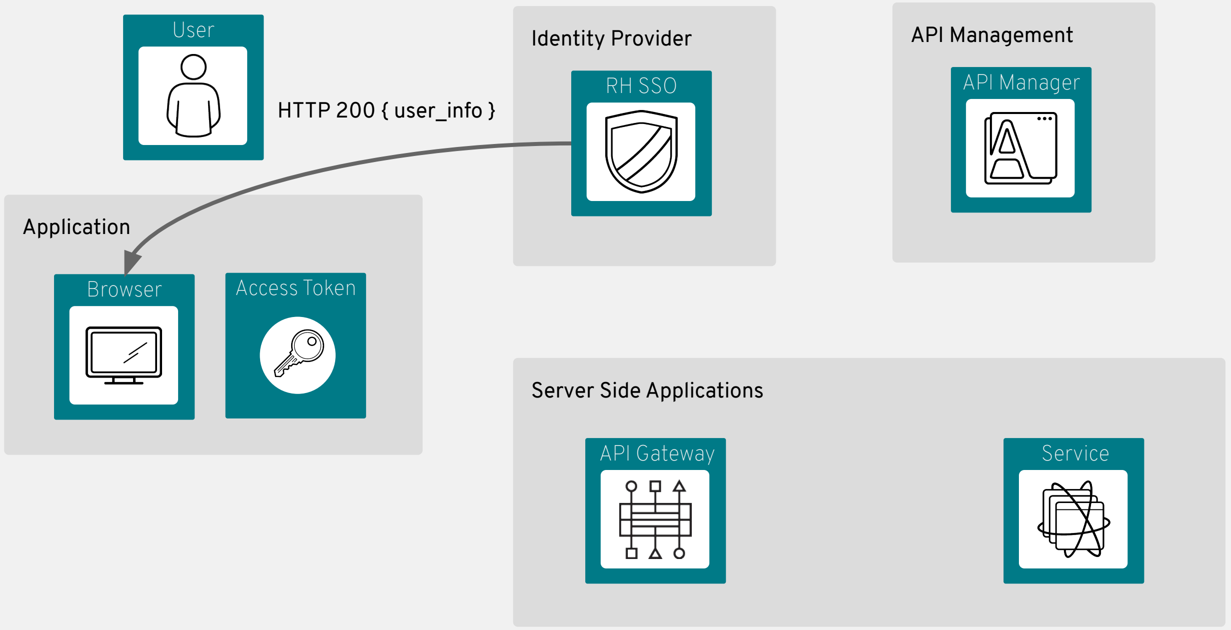
AUTHORIZATION CODE FLOW

#9.5 - Web App Submits The Access Token To Get User Info (Optional)



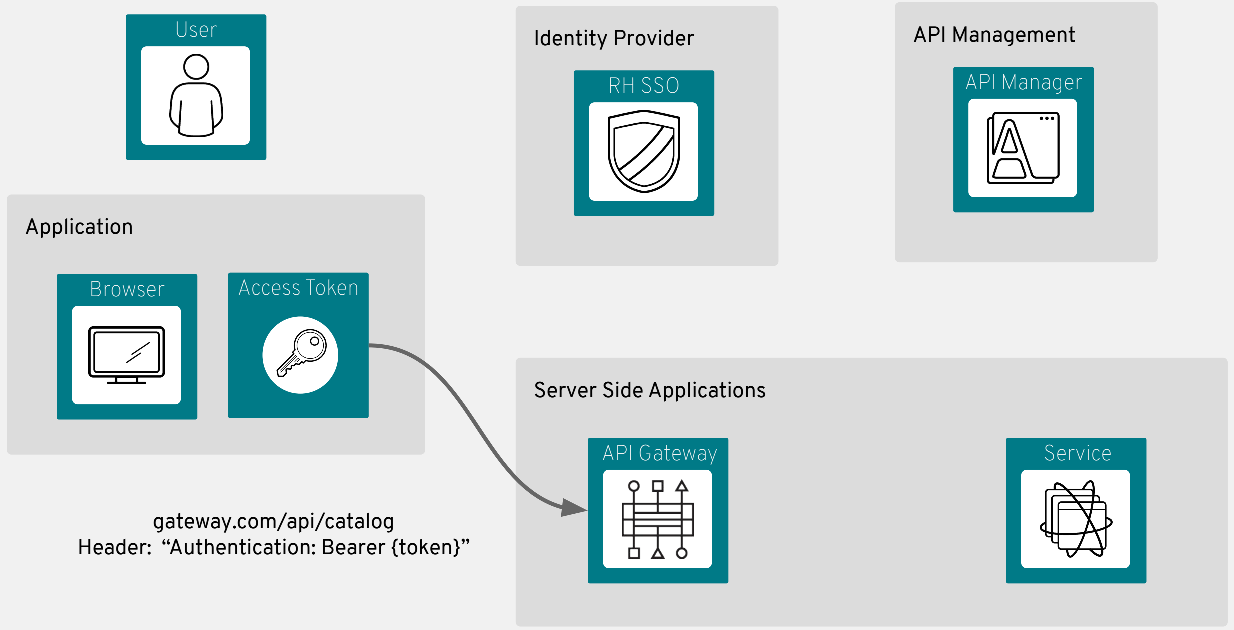
AUTHORIZATION CODE FLOW

#9.6 - Web App Receives UserInfo



AUTHORIZATION CODE FLOW

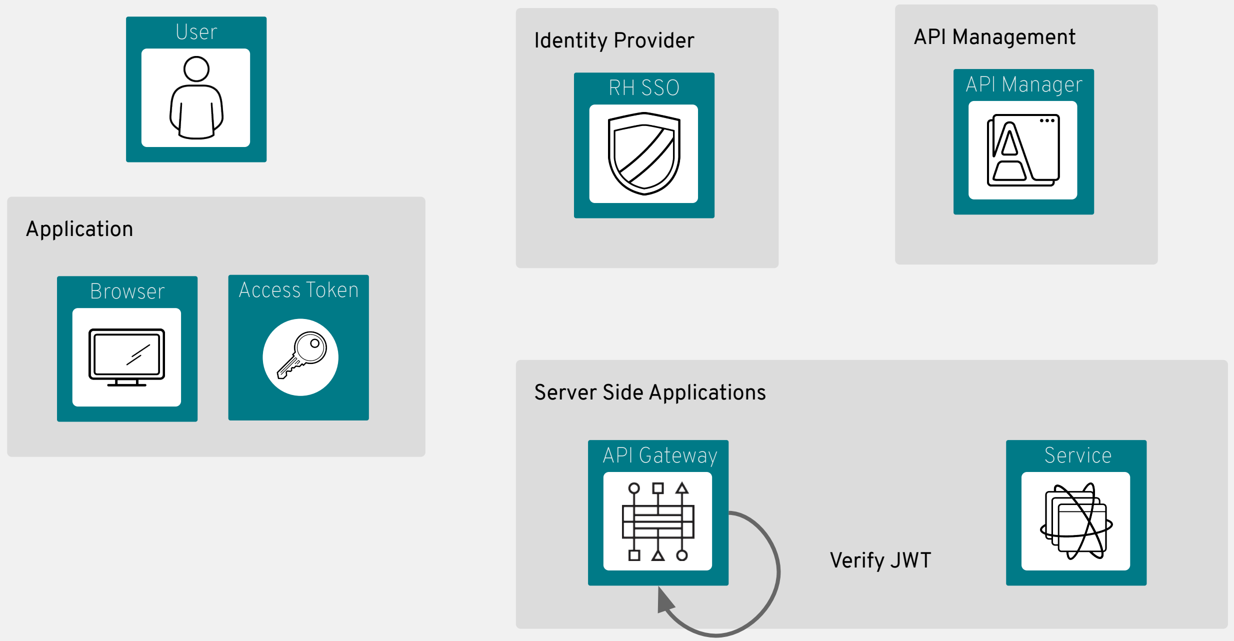
#10 - Web App Submits The Bearer Token



gateway.com/api/catalog
Header: “Authentication: Bearer {token}”

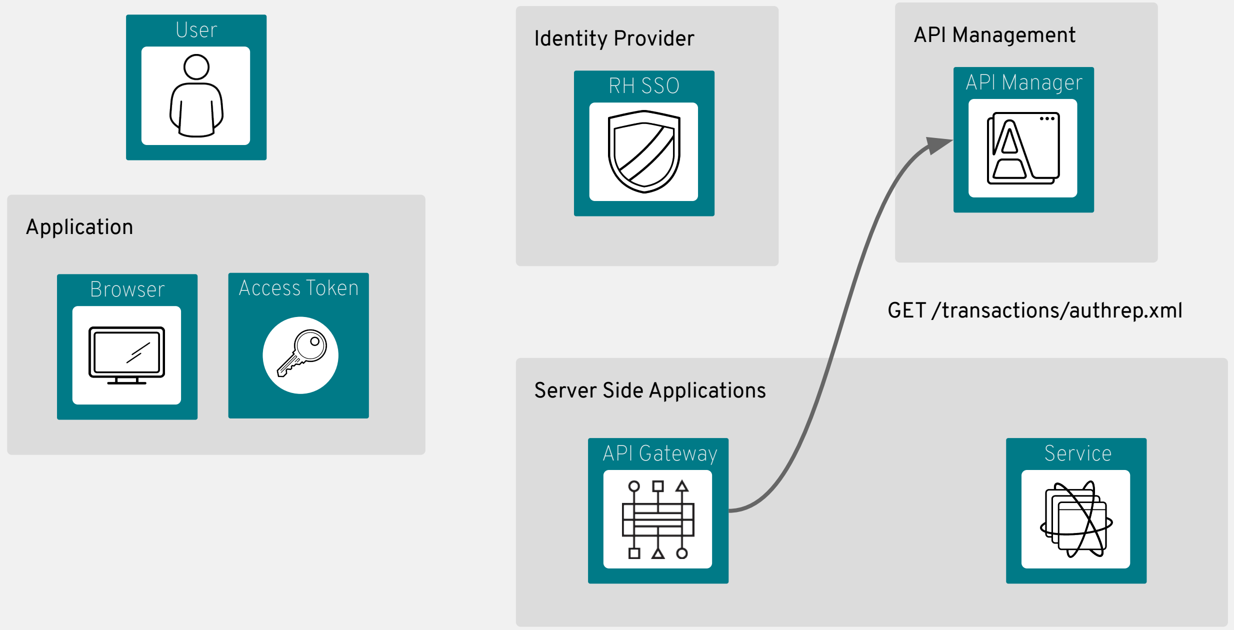
AUTHORIZATION CODE FLOW

#10.1 - Gateway Verifies Token



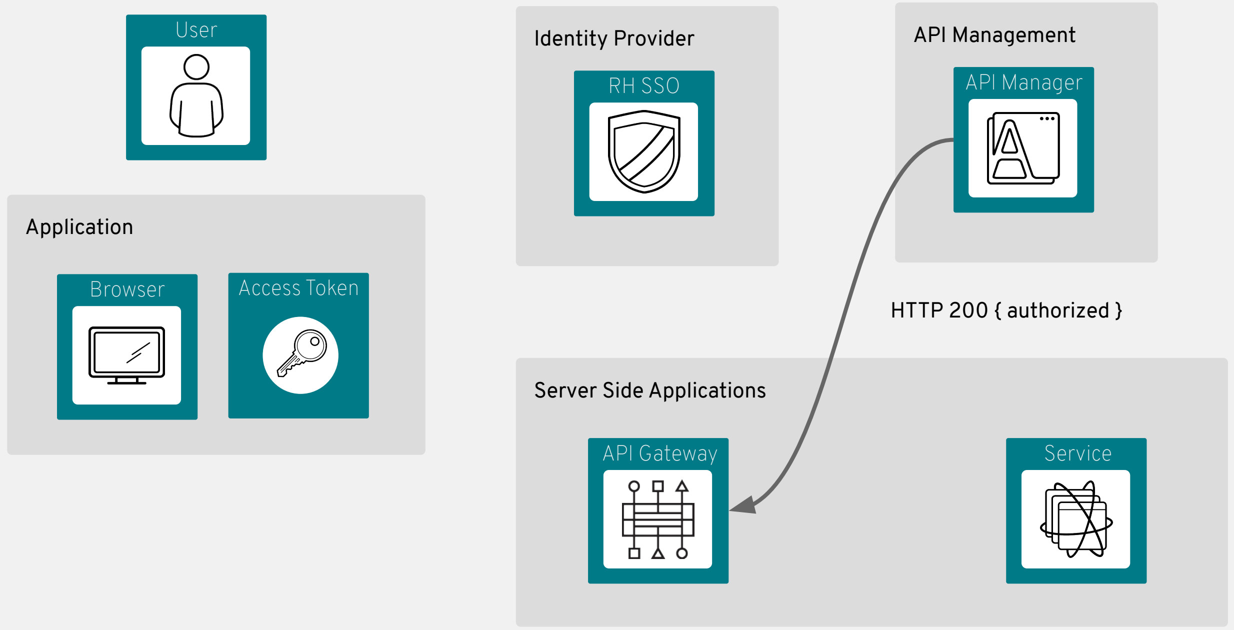
AUTHORIZATION CODE FLOW

#10.2 - Gateway Requests Auth To API Manager



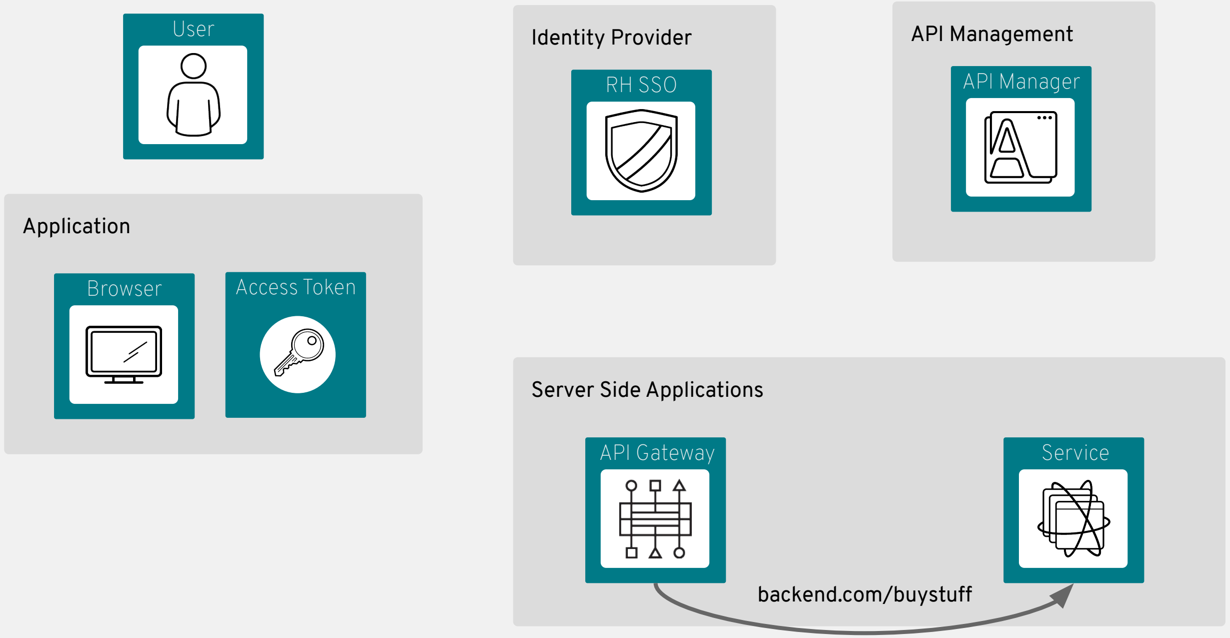
AUTHORIZATION CODE FLOW

#10.3 - API Manager Response “Authorized”



AUTHORIZATION CODE FLOW

#10.3 - Gateway Calls Backend API



RED HAT 3SCALE API MANAGEMENT

RED HAT 3SCALE API MANAGEMENT

System Architecture

The 3scale API Management architecture consists of :

- The **API Manager** which manages the API, Developers and Applications
- The **Traffic Manager** (API Gateways) that enforce the policies from the API Manager and delegate authorization to 3rd party IDPs
- The **Identity Provider** (IDP) identity hub that supports many authentication using various protocols
- The **API Backend** the API. i.e. the API Provider

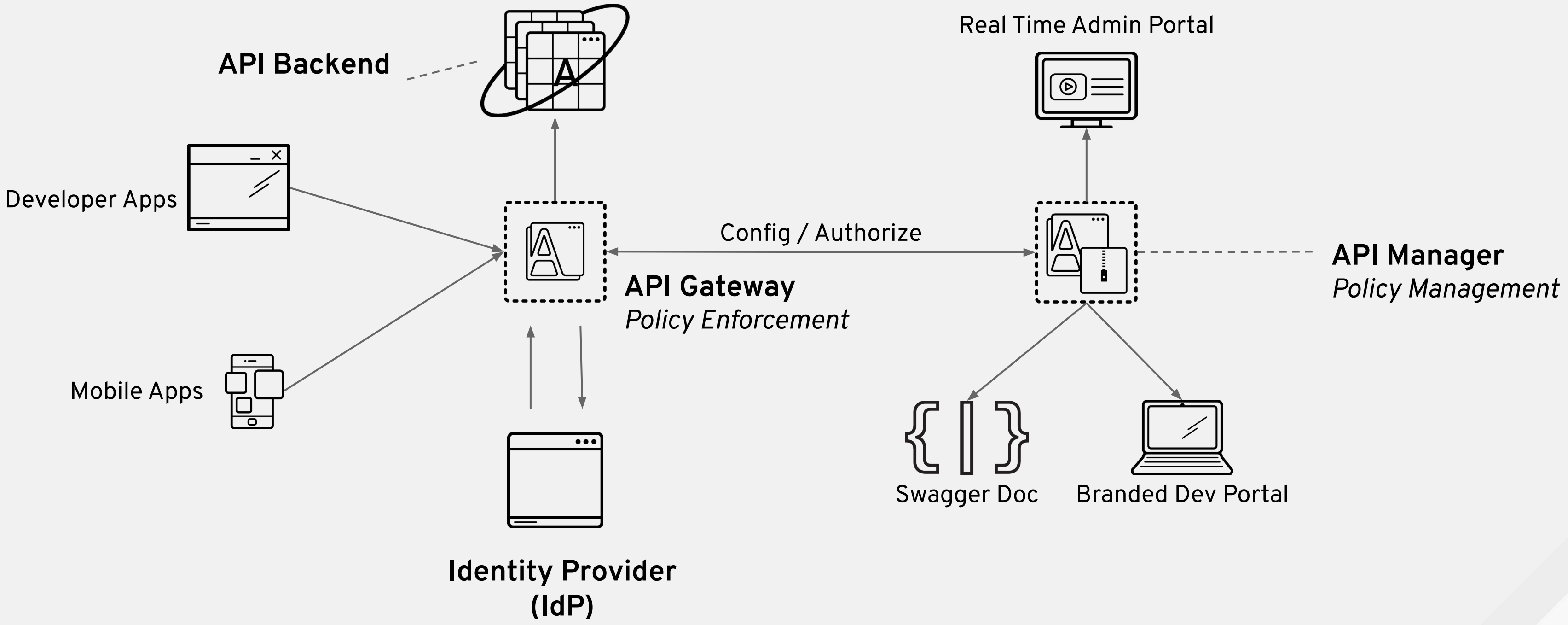
RED HAT API MANAGEMENT

Gateway Operations

- Checks the timestamp for ‘expired’ token.
- Checks the client_id is still valid
- Performs a check on the signature of the JWT using RH SSO public key

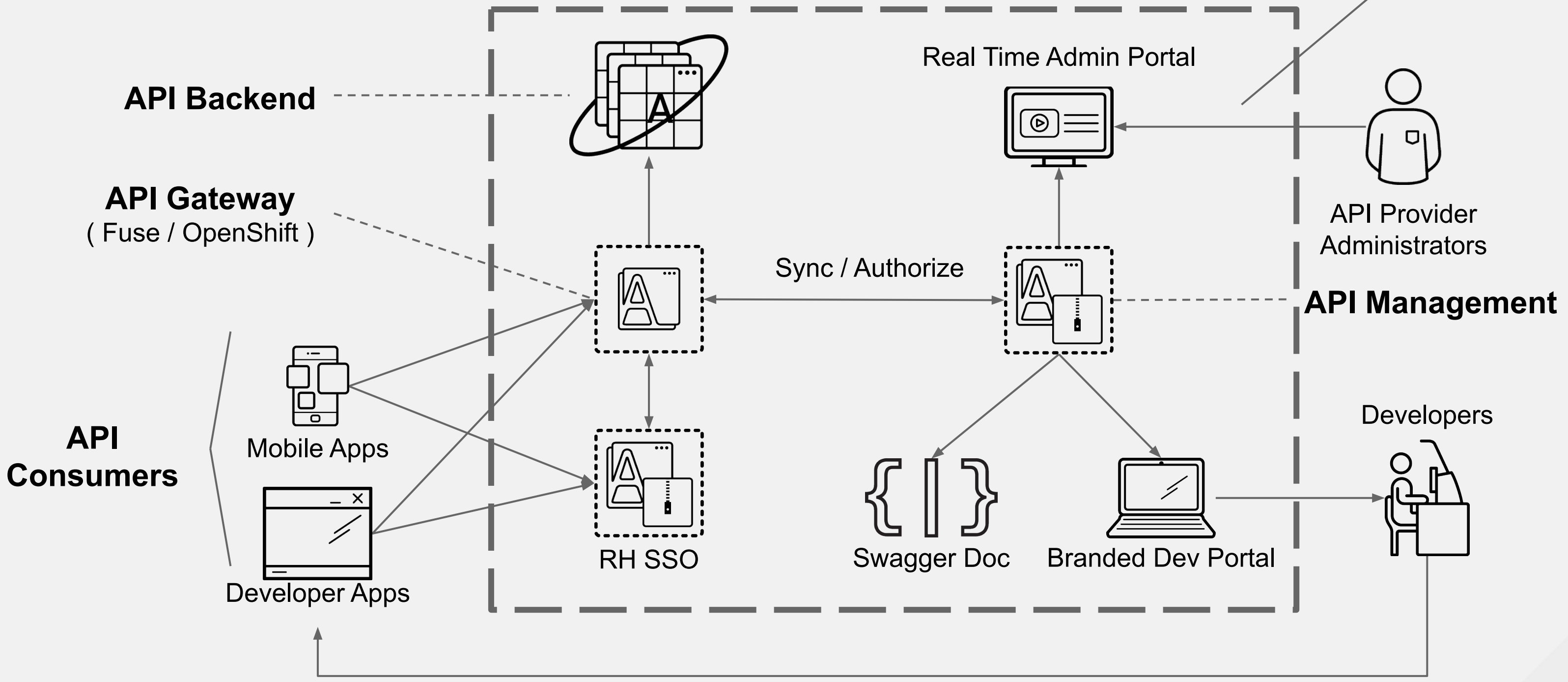
RED HAT API MANAGEMENT

System Architecture



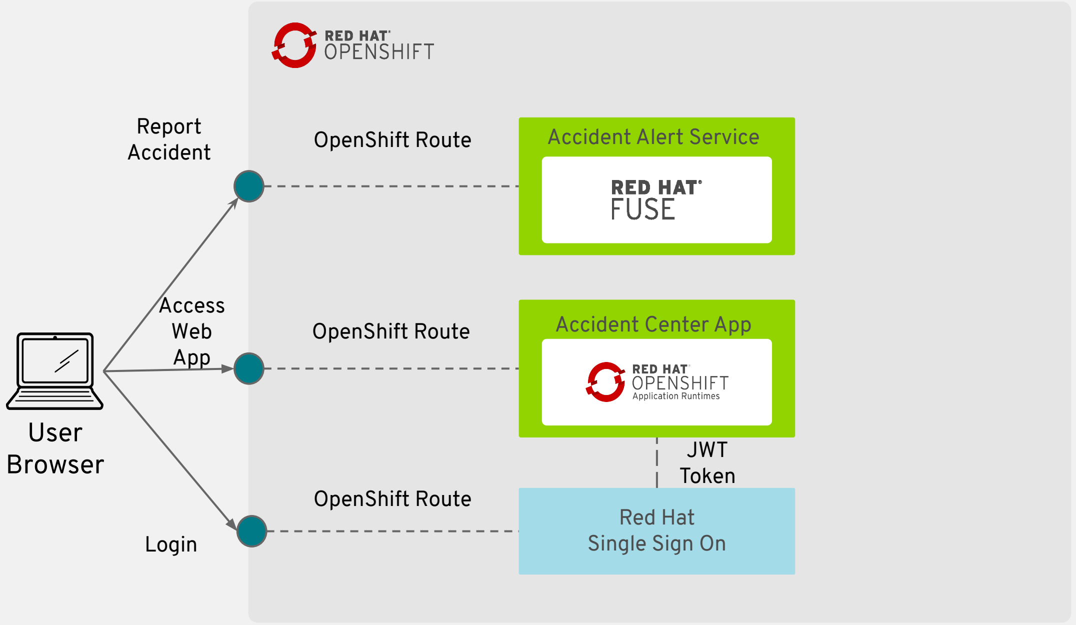
RED HAT API MANAGEMENT

2.1: Openshift On-Premises + OpenID Connect

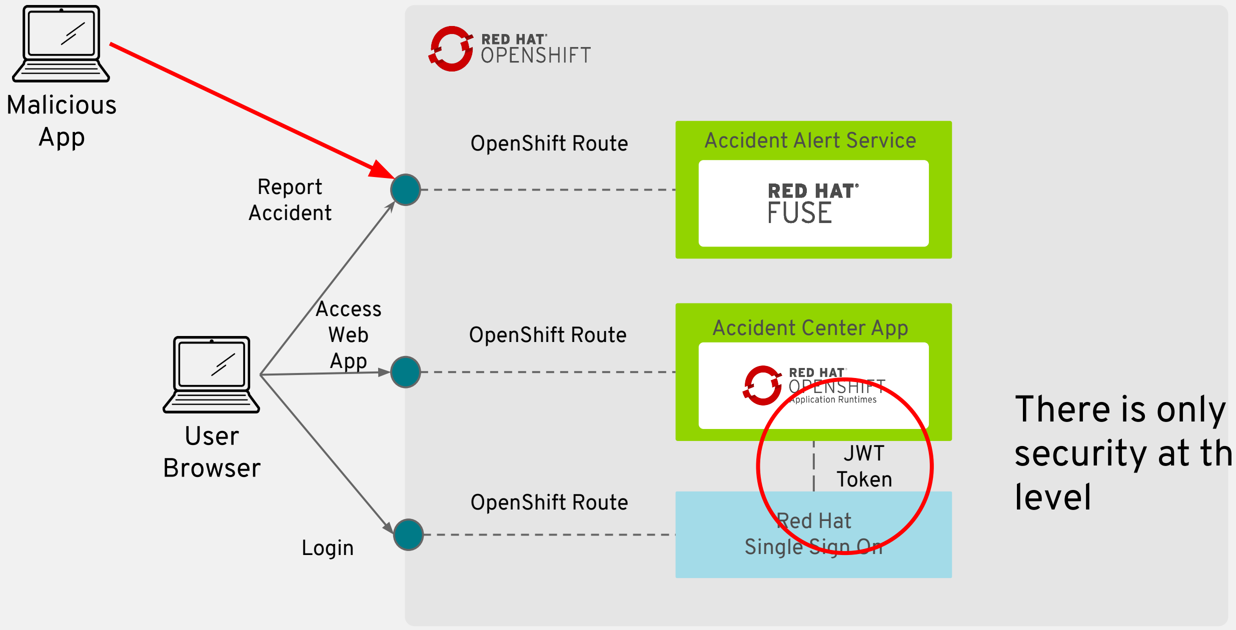


DEMO

INITIAL SCENARIO

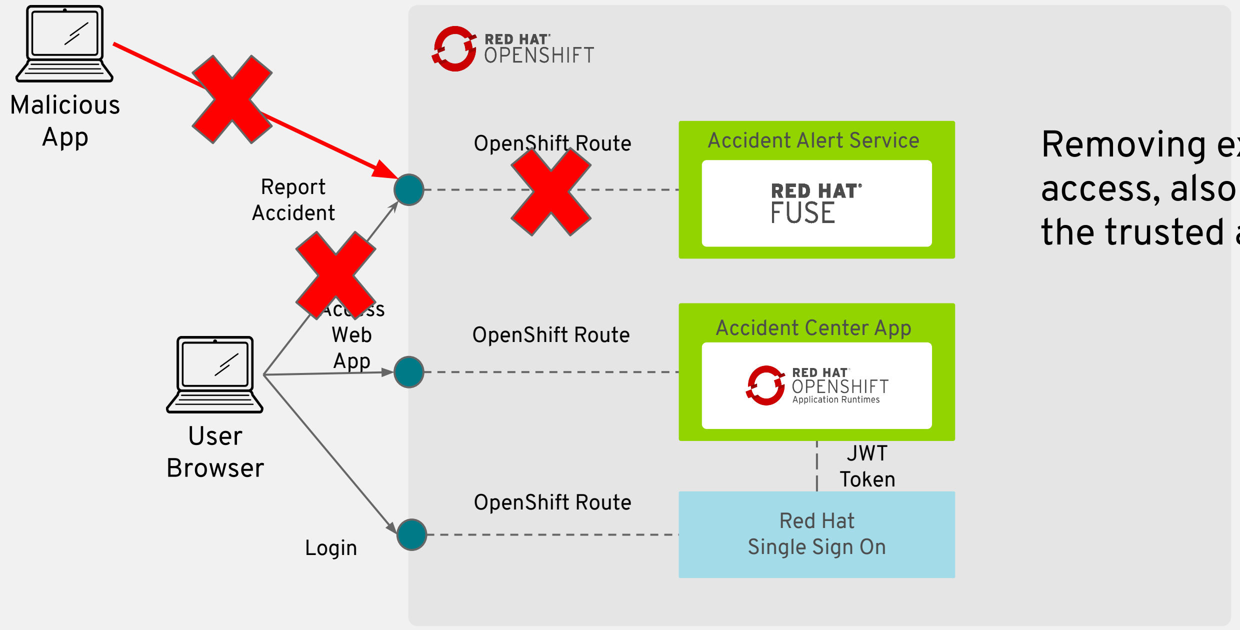


INSECURE SCENARIO



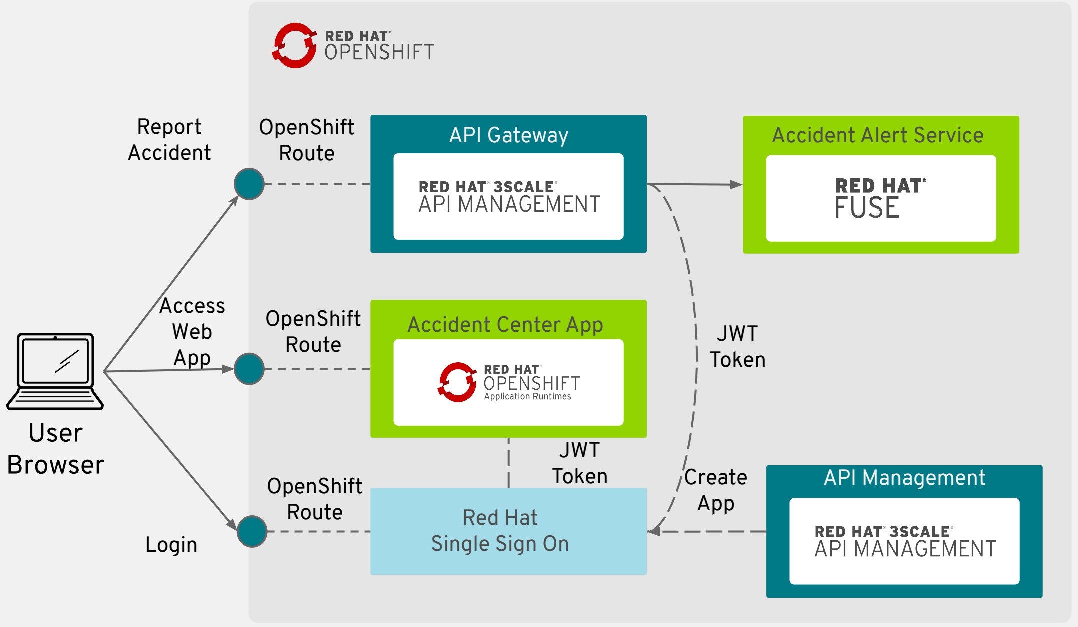
There is only security at the UI level

ISOLATE BACKEND API

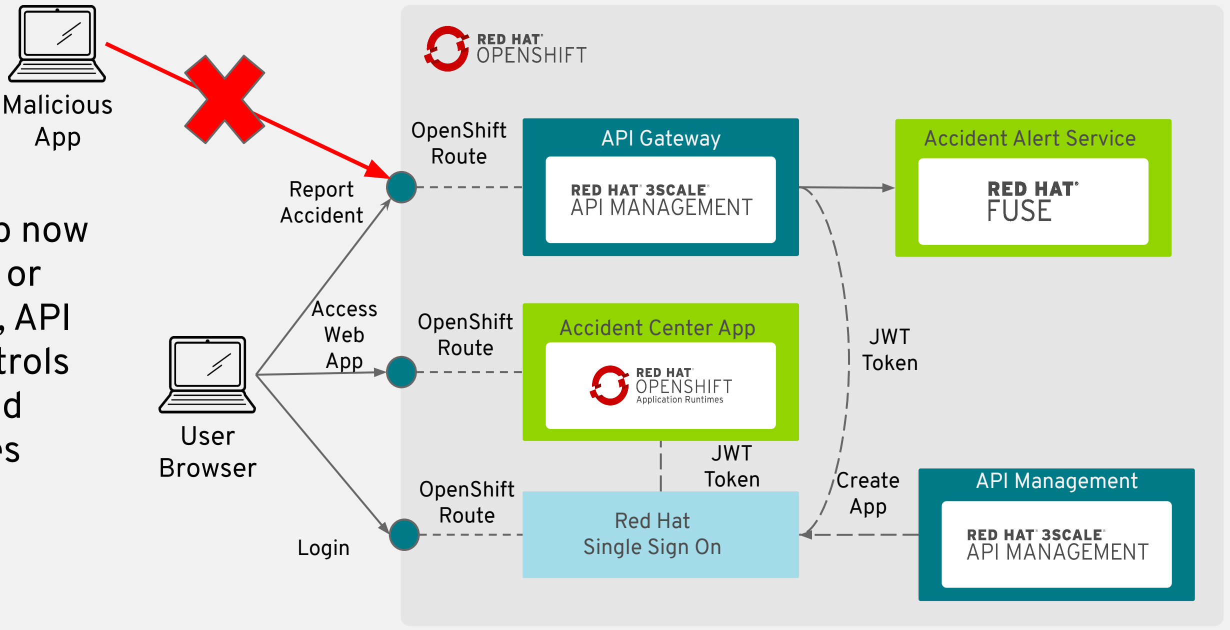


Removing external access, also breaks the trusted app

SECURING THE SERVICE

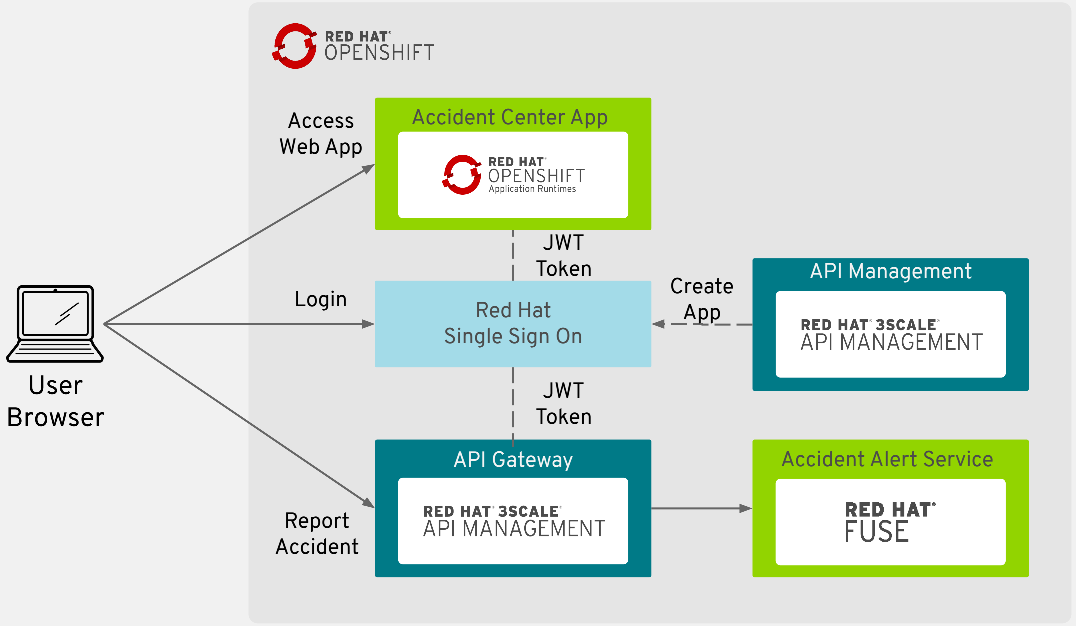


SECURING THE SERVICE



Malicious app now has no token or invalid token, API gateway controls rate limits and mapping rules

FINAL DEPLOYMENT



redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos