Software Architecture Summit

RESTful Web API Design

Rainer Stropek

Software Architecture Summit 2015

RESTful

Web API Design

Rainer Stropek

software architects gmbh

Web	http://www.timecockpit.com
Mail	rainer@timecockpit.com
Twitter	@rstropek

time cockpit

Saves the day.

Agenda

RESTful Web APIs have become an integral part of modern software packages. They are important for integration scenarios in enterprises and in the cloud. This workshop is dedicated to designing RESTful Web APIs. Rainer Stropek, himself founder a SaaS-focused company, will guide you through the world of RESTful APIs. In particular, Rainer will speak about the following topics:

- Short recap of the basic principles of RESTful Web APIs
- Real-world RESTful API design (e.g. addressing in multi-tenant systems, versioning, long-running operations, etc.)
- Authentication and authorization with OAuth2 and OpenID Connect
- The OData standard for RESTful APIs
- The role of metadata using the examples of http://swagger.io/ and OData
- Securing and operating RESTful APIs using the example of Azure API Management
- Code samples using Node.js with JavaScript and .NET with C#

Attendees of this workshop should have some understanding of http and cloud computing. Practical experience regarding RESTful API design or development is not necessary.

RESTful Web APIs

Short recap of the basic principles of RESTful Web APIs

What is „REST“?

Representational State Transfer (REST)

Architecture style, not a standard

HTTP

Request-response protocol in client-server systems

HTTP methods („verbs“)

- GET – retrieve data, no side effects (except logging, caching, etc.)
- HEAD – like get but without response body, useful to retrieve metadata
- POST – submit new data
- PUT – update or create
- PATCH – partial update
- DELETE
- TRACE – echo
- OPTIONS – query verbs that the server supports for a given URL

What is „REST“?

HTTP

- Idempotent requests
  - GET, HEAD, OPTIONS, TRACE
  - PUT, DELETE
- Non idempotent requests
  - POST
- Status Codes (complete list of status codes), examples:
  - 200 OK
  - 201 Created
  - 301 Moved permanently
  - 400 Bad request
  - 401 Unauthorized
  - 403 Forbidden (authorization will not help)
  - 404 Not found
  - 405 Method not allowed (wrong verb)
  - 500 Internal server error

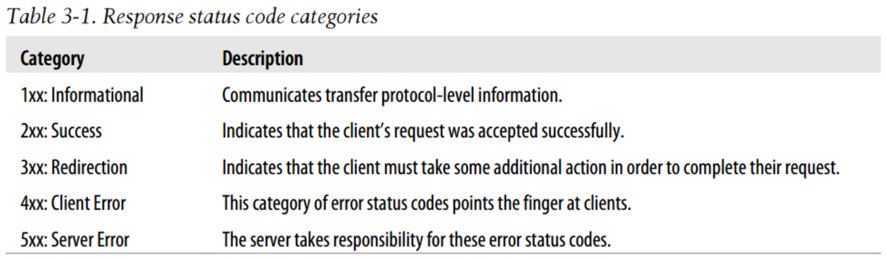


Table 3-1. Response status code categories

Category	Description
1xx: Informational	Communicates transfer protocol-level information.
2xx: Success	Indicates that the client's request was accepted successfully.
3xx: Redirection	Indicates that the client must take some additional action in order to complete their request.
4xx: Client Error	This category of error status codes points the finger at clients.
5xx: Server Error	The server takes responsibility for these error status codes.

Source of Table: Mark Massé, REST API Design Rulebook, O'Reilly

What is „REST“?

HTTP

Header fields (list of header fields), examples:

- Accept – e.g. application/json
- Authorization – authentication credentials
- Cache-Control
- Cookie
- Content-Type
- If-Match, If-Modified-Since, If-Unmodified-Since
- X-... - non-standard fields

- ETag – identifier for a specific version of a resource
- Last-Modified
- Set-Cookie

What is „REST"?

Important REST principles

Stateless

- No client context stored on the server, each request is complete

Cacheable

- Responses explicitly indicate their cacheability

Layered System

- Client cannot tell if connected directly to the server (e.g. reverse proxies)

URIs

- Resources are identified using *Uniform Resource Identifiers* (URIs)

Resource representation

- XML, JSON, Atom – today mostly JSON

RESTful Web API

Interacting with a RESTful web api

Tools

- Azure Mobile Service
- Fiddler
- Postman

Demo

RESTful Web API

Demoscript

```
Create Azure Mobile Service
   Show REST API documentation

Create table, allow all requests anonymously

Show POST, GET in Fiddler

Show POST, PATCH, DELETE, GET in Postman

Show table content in SQL Management Studio

Change access policy to API key
  Get API key
  Show GET with API key in X-ZUMO-APPLICATION header
```

API Design

Real-world RESTful API design

Design Rules

Do use HTTPS

No-brainer on public networks
Recommended on company/home network, too

Do use a consistent naming schema

Prefer hyphens ("-") instead of underscores ("_") in URIs
Do not mix languages
Prefer lowercase letters in URIs
Prefer camel casing for resource representation (e.g. in JSON)
Singular noun for documents, plural noun for collections, verb for controller names

Design Rules

Do carefully model URI paths

URIs should reflect the API's resource model

- E.g. *https://api.myservice.com/customers/ALFKI/orders*
- Bad example: *https://api.myservice.com/afe7f2cb-8e71-4472-a53b-1f8e3712dffc/orders*

Don't forget controller resources

Consider identity values for variable URI path segments

E.g. *https://api.myservice.com/customers/ALFKI/orders*

Do use HTTP verbs as they were intended to

Also for controller resources (e.g. POST for controller that creates data)

Consider firewall problems with PUT and sometimes even DELETE

Avoid using controller names instead of HTTP verbs

- Bad example: https://api/myservice.com/customers/deleteCustomer?id=ALFKI

RESTful Web API

Controller resources

Demo

```
exports.post = function(request, response) {
    if (!request.body || !request.body.rows) {
        response.status(400).end();
    }
    else {
        var customerTable =
            request.service.tables.getTable('customers');
        for (var i = 0; i < request.body.rows; i++) {
            var customer = {
                firstName: "Test " + i.toString(),
                lastName: "Test " + i.toString(),
                revenueYTD: i * 1000
            };
            customerTable.insert(customer);
        }

        response.status(201).end();
    }
};
```

RESTful Web API

Demoscript

Custom API

Design Rules

Do use standard response codes as they were intended to

200 for success
201 if somethings has been created (specify URI of new resource in *Location* header)
202 if controller started an async operation
204 if not response was sent back intentionally (PUT, POST, DELETE)
401 if something is wrong with authorization
404 if no resource is present at given URI
406/415 if requested/given Content-Type is not supported
500 represents a server error (not the client's fault)

Consider returning additional error information in body

Use response code 4xx and error information in response body
Don't expose security-critical data in error messages (especially for server errors)
- Use properly protected logs instead

Demo

API Design

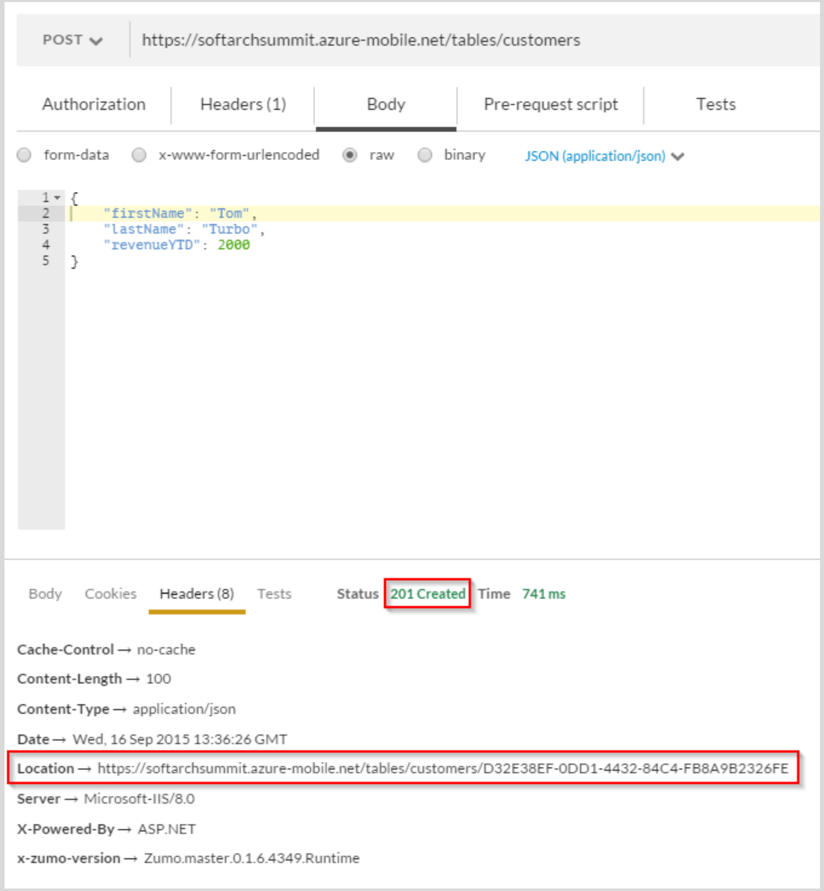
Location header with POST

Additional error data in case of 4xx error

RESTful Web API

Demoscript

Location header



POST https://softarchsummit.azure-mobile.net/tables/customers

Authorization | Headers (1) | Body | Pre-request script | Tests

form-data | x-www-form-urlencoded | [✓ raw] | binary | JSON (application/json)

```
{
    "firstName": "Tom",
    "lastName": "Turbo",
    "revenueYTD": 2000
}
```

Body | Cookies | Headers (8) | Tests

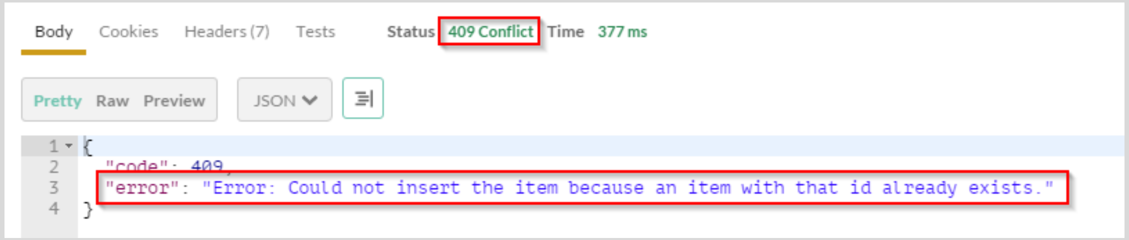
Status 201 Created Time 741 ms

```
Cache-Control → no-cache
Content-Length → 100
Content-Type → application/json
Date → Wed, 16 Sep 2015 13:36:26 GMT
Location → https://softarchsummit.azure-mobile.net/tables/customers/D32E38EF-0DD1-4432-84C4-FB8A9B2326FE
Server → Microsoft-IIS/8.0
X-Powered-By → ASP.NET
x-zumo-version → Zumo.master.0.1.6.4349.Runtime
```

RESTful Web API

Demoscript

Additional error data



Status 409 Conflict Time 377 ms

```
{
  "code": 409,
  "error": "Error: Could not insert the item because an item with that id already exists."
}
```

Design Rules

Don't use GET + query for controller actions that write

Use proper HTTP verbs and parameters in the request body instead

Do use query for ad hoc filtering, sorting, paging, etc.

Examples:

https://api.myservice.com/customers?$filter=name eq 'ALFKI'
https://api.myservice.com/customers?$top=10
https://api.myservice.com/customers?$orderby=name
http://petstore.swagger.io/v2/pet/findByStatus?status=sold

See also *OData* (more details later)

Consider allowing correlation identifier in custom header

Stored in server-side logs
Can be used to correlate client- and server-side activities

Design Rules

Consider support for batching of operations

Performance considerations (latency reduction)

Execute in server-side transactions

Example: Entity Group Transactions in *Azure Table Storage*

Consider using Multipart MIME messages

Example: OData Batch Requests

Consider allowing the client to specify a server timeout

Do define a maximum server timeout to protect from over-usage of server resources

Consider progress reporting for long running requests

Examples: Polling API, Message bus, SignalR

Design Rules

Consider using *Etag* and *If-None-Match* to save bandwidth

Consider using *If-Match* or *If-Unmodified-Since* for optimistic concurrency

Consider allowing to suppress response echo on POST

Typically, POST returns created document

Consider a header with which the client can suppress this echo to save bandwith

API Design

Location header with POST

Additional error data in case of 4xx error

Building Web API with Node.js

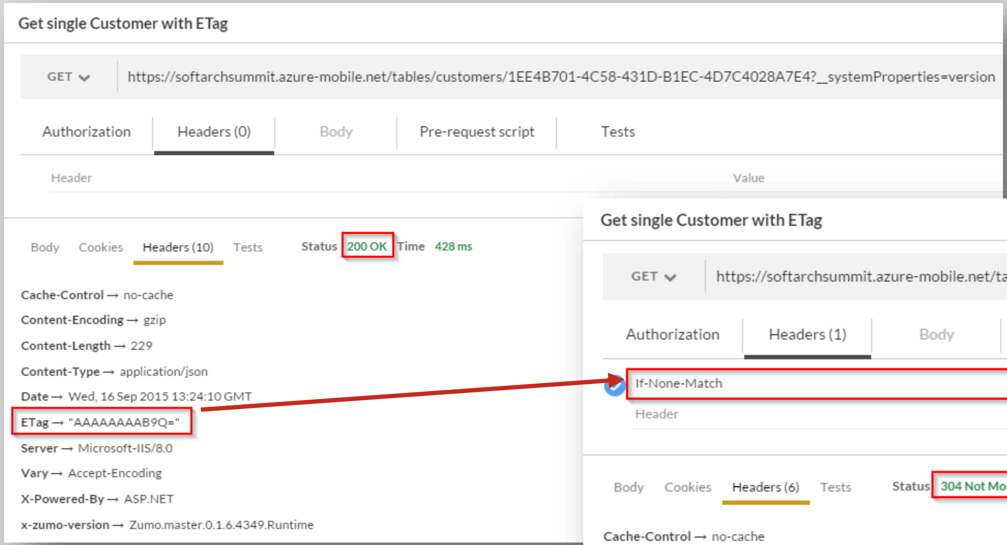
Prefer header in *Azure Table Storage*

Demo

RESTful Web API

Demoscript

ETag and *If-None-Match*



Get single Customer with ETag

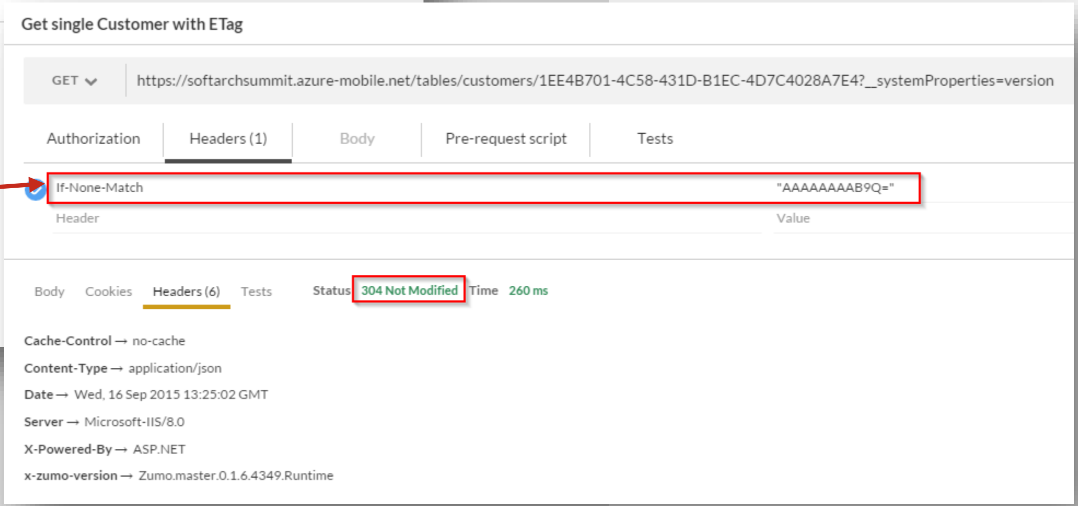
GET https://softarchsummit.azure-mobile.net/tables/customers/1EE4B701-4C58-431D-B1EC-4D7C4028A7E4?__systemProperties=version

Authorization | Headers (0) | Body | Pre-request script | Tests

Header | Value

Body | Cookies | Headers (10) | Tests — Status 200 OK Time 428 ms

```
Cache-Control → no-cache
Content-Encoding → gzip
Content-Length → 229
Content-Type → application/json
Date → Wed, 16 Sep 2015 13:24:10 GMT
ETag → "AAAAAAAAB9Q="
Server → Microsoft-IIS/8.0
Vary → Accept-Encoding
X-Powered-By → ASP.NET
x-zumo-version → Zumo.master.0.1.6.4349.Runtime
```



Get single Customer with ETag

GET https://softarchsummit.azure-mobile.net/tables/customers/1EE4B701-4C58-431D-B1EC-4D7C4028A7E4?__systemProperties=version

Authorization | Headers (1) | Body | Pre-request script | Tests

Header	Value
If-None-Match	"AAAAAAAAB9Q="

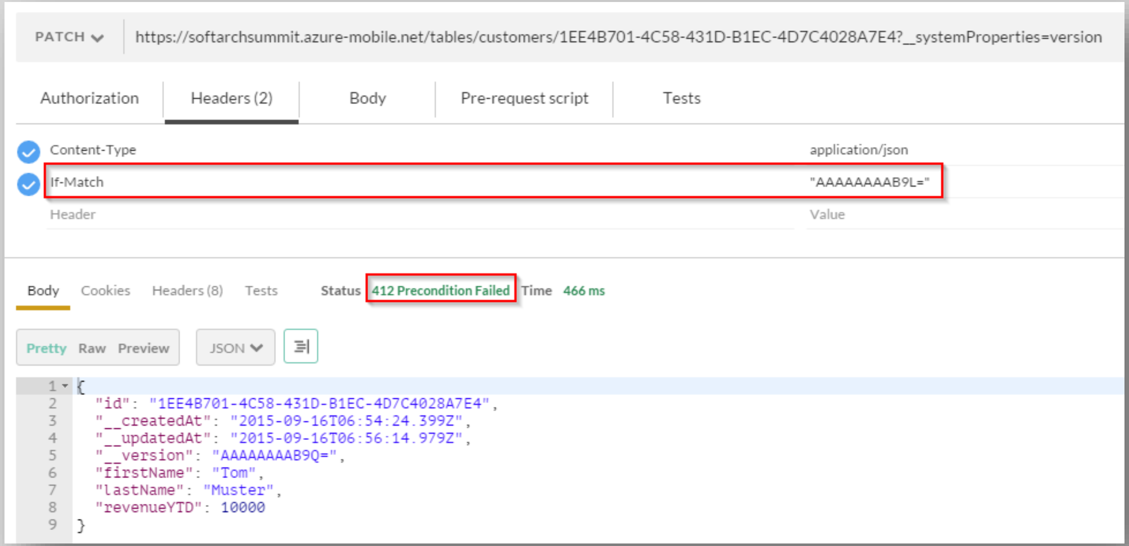
Body | Cookies | Headers (6) | Tests — Status 304 Not Modified Time 260 ms

```
Cache-Control → no-cache
Content-Type → application/json
Date → Wed, 16 Sep 2015 13:25:02 GMT
Server → Microsoft-IIS/8.0
X-Powered-By → ASP.NET
x-zumo-version → Zumo.master.0.1.6.4349.Runtime
```

RESTful Web API

Demoscript

If-Match and optimistic concurrency



PATCH https://softarchsummit.azure-mobile.net/tables/customers/1EE4B701-4C58-431D-B1EC-4D7C4028A7E4?__systemProperties=version

Header	Value
Content-Type	application/json
If-Match	"AAAAAAAAB9L="

Status 412 Precondition Failed Time 466 ms

```
{
  "id": "1EE4B701-4C58-431D-B1EC-4D7C4028A7E4",
  "__createdAt": "2015-09-16T06:54:24.399Z",
  "__updatedAt": "2015-09-16T06:56:14.979Z",
  "__version": "AAAAAAAAB9Q=",
  "firstName": "Tom",
  "lastName": "Muster",
  "revenueYTD": 10000
}
```

Design Rules

Do support JSON for resource representation

application/json

Consider other resource representation if needed

E.g. *application/xml*

Consider adding links

Programmatically process connections between resources

Consider publishing schema information

For details see *OData* and *Swagger*

API Design

Links for entities in OData

- XOData

Demo

Design Rules

Consider configuring CORS to enable broad web API usage

Don't solely rely on CORS for protecting your resources

Avoid JSONP (JSON with padding)

Work around same origin policy by injecting <*script*> tags at runtime

Do use *OAuth2* and *OpenID Connect* to protect resources

See also *Protecting Resource* section later for more details

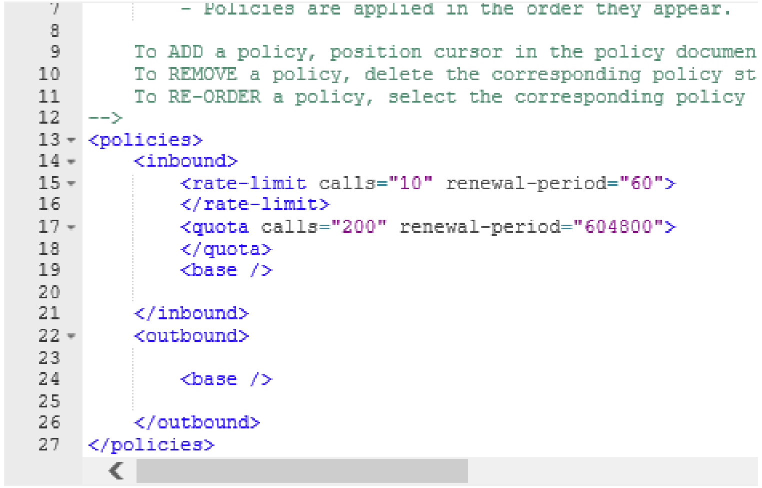
Design Rules

Do limit server resource usage in multi-tenant systems

Examples:

- Query timeout and pagination in *Azure Table Storage*
- API rate limits in Azure API Management

Policy definition



```
        - Policies are applied in the order they appear.

    To ADD a policy, position cursor in the policy documen
    To REMOVE a policy, delete the corresponding policy st
    To RE-ORDER a policy, select the corresponding policy
-->
<policies>
    <inbound>
        <rate-limit calls="10" renewal-period="60">
        </rate-limit>
        <quota calls="200" renewal-period="604800">
        </quota>
        <base />

    </inbound>
    <outbound>

        <base />

    </outbound>
</policies>
```

Design Rules

Do plan for versioning your web API

Consider using a custom header for API version to enable complex versioning scenarios

Examples

- *x-ms-version* in Azure Table Storage
- *OData-MaxVersion* and *OData-Version* headers in Odata

Consider using version-specific URIs for simple versioning scenarios and major versions

Protecting Resources

CORS – Cross-Origin Resource Sharing

What is CORS?

XMLHttpReqest limits cross-domain web API calls

Same origin policy: Script can only make HTTP requests to the domain it came from

CORS is a W3C spec to allow cross-domain calls

See http://enable-cors.org/client.html for browser support

Server specifies allowed calling domains in special response headers

See Mozilla Docs for technical details about CORS

https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

How CORS works

Simple requests

GET, *HEAD* or *POST*

- If *POST*, only content types *application/x-www-form-urlencoded*, *multipart/form-data*, or *text/plain*

No custom headers in the request

Browser sends *Origin* header

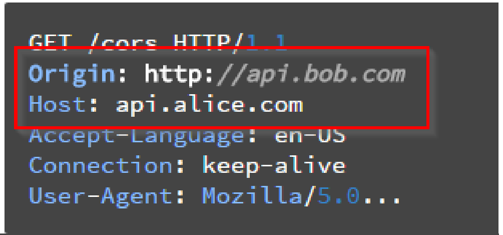
Server returns error if Origin in not allowed to do API calls

Access-Control headers

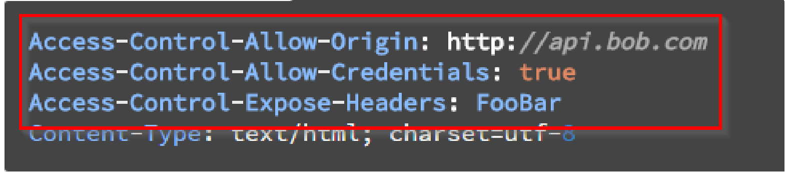
Allow-Origin: * or *Origin*

Allow-Credentials: Cookies included?

Expose-Headers: Non-simple headers available to the client



```
GET /cors HTTP/1.1
Origin: http://api.bob.com
Host: api.alice.com
Accept-Language: en-US
Connection: keep-alive
User-Agent: Mozilla/5.0...
```



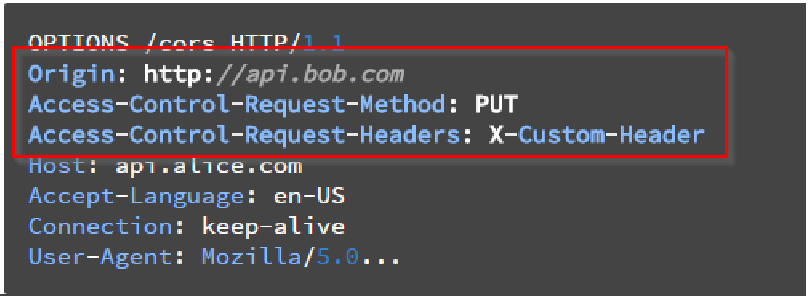
```
Access-Control-Allow-Origin: http://api.bob.com
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: FooBar
Content-Type: text/html; charset=utf-8
```

How CORS works

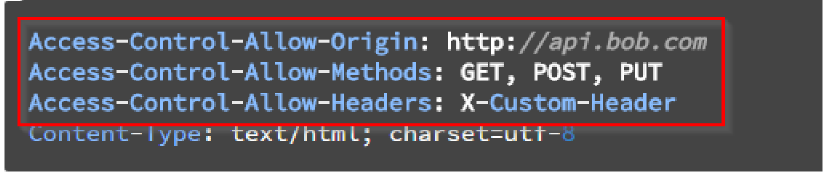
Non-simple requests

Preflight request

Client asks for permissions
Server must support OPTIONS
Performance implications
Server returns no CORS headers if not allowed



```
OPTIONS /cors HTTP/1.1
Origin: http://api.bob.com
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: X-Custom-Header
Host: api.alice.com
Accept-Language: en-US
Connection: keep-alive
User-Agent: Mozilla/5.0...
```



```
Access-Control-Allow-Origin: http://api.bob.com
Access-Control-Allow-Methods: GET, POST, PUT
Access-Control-Allow-Headers: X-Custom-Header
Content-Type: text/html; charset=utf-8
```

Actual request follows successful preflight request

Demo

CORS

Adding CORS support to ASP.NET Web API

RESTful Web API

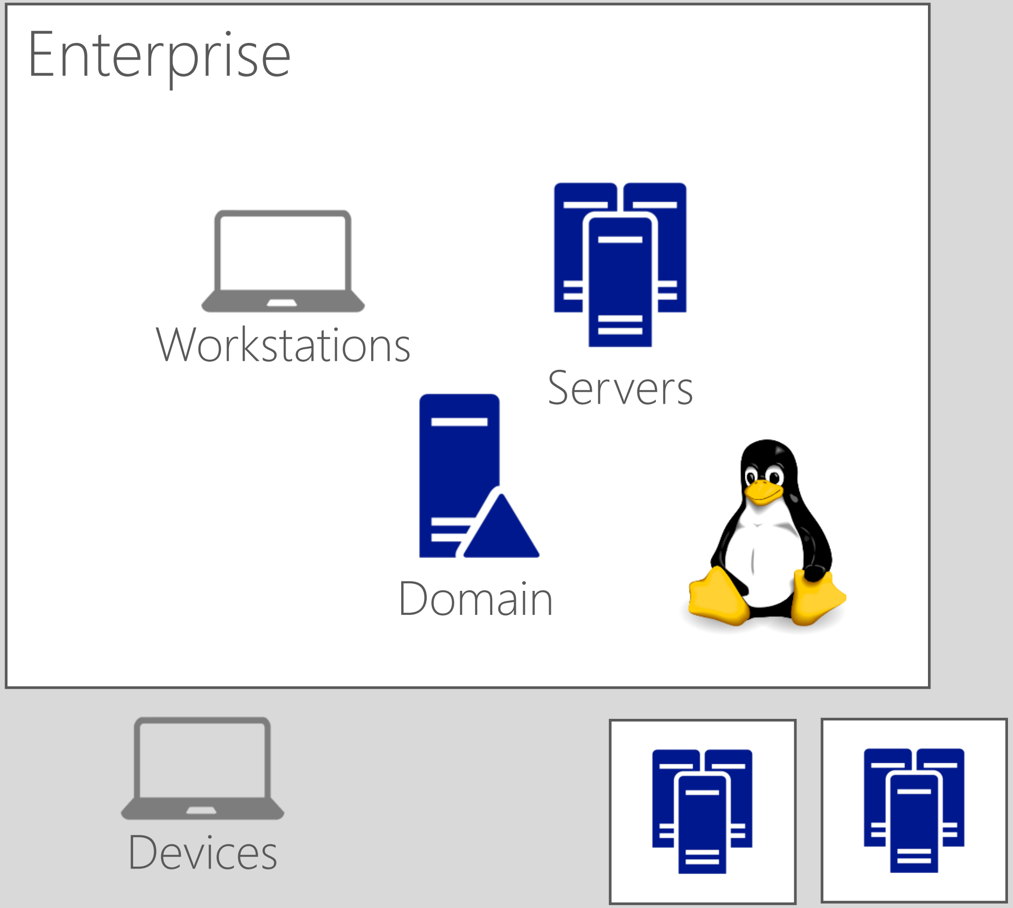
Demoscript

Add NuGet package Microsoft.AspNet.WebApi.Cors

```
public static void Register(HttpConfiguration config)
{
    // New code
    config.EnableCors();
}


--- or ---

[EnableCors(origins: "http://example.com",
  headers: "*", methods: "*")]
public class TestController : ApiController
{
    // Controller methods not shown...
}
```

Protecting Resources

Auth with OAuth2 and OpenID Connect



Local Auth

Auth inside of the enterprise

Single, integrated domain

All devices belong to the enterprise

Everything is Windows

Problems

- External devices
- External services
- Non-Windows environments

OAuth2

Successor of OAuth1 and OAuth WRAP

Standard for delegating authorization for accessing resources via HTTP(S)

- Not a standard for authentication
- Not a standard for authorization

Very common in the internet today

- Many different flavors as the standard leaves many decisions up to the developer
- Example: https://oauth.io/

Important Terms

OAuth Provider

Aka OAuth Server, Authorization Server

Examples: AD FS, Google, Twitter, Microsoft AAD

Resource Provider

Aka Resource Server

In our case: A REST Web API

Resource Owner

In our case: The end user, the organization

Client

Application accessing a protected resource

In our case: Native app, server-based web app, SPA, mobile app

OAuth Endpoints

Authorization Endpoint (aka OAuth-A)

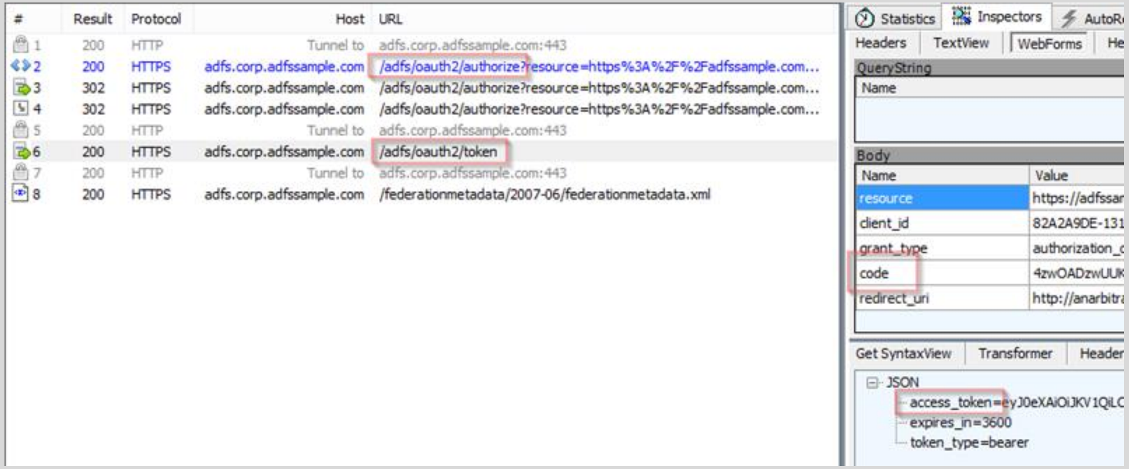
Authenticates the resource owner (e.g. user/password)
Asks for consent
Sends confirmation (access code) to redirect endpoint

Redirect Endpoint

Offered by the client
Called via redirecting the user-agent (HTTP redirect 302)
Receives code (there are other options, too) and fetches token from token endpoint

Token Endpoint (aka OAuth-T)

Creates tokens for access codes, refresh tokens, etc.
Can validate the client using a client secret



#	Result	Protocol	Host	URL
1	200	HTTP	Tunnel to	adfs.corp.adfssample.com:443
2	200	HTTPS	adfs.corp.adfssample.com	/adfs/oauth2/authorize?resource=https%3A%2F%2Fadfssample.com...
3	302	HTTPS	adfs.corp.adfssample.com	/adfs/oauth2/authorize?resource=https%3A%2F%2Fadfssample.com...
4	302	HTTPS	adfs.corp.adfssample.com	/adfs/oauth2/authorize?resource=https%3A%2F%2Fadfssample.com...
5	200	HTTP	Tunnel to	adfs.corp.adfssample.com:443
6	200	HTTPS	adfs.corp.adfssample.com	/adfs/oauth2/token
7	200	HTTP	Tunnel to	adfs.corp.adfssample.com:443
8	200	HTTPS	adfs.corp.adfssample.com	/federationmetadata/2007-06/federationmetadata.xml

Name	Value
resource	https://adfssar
client_id	82A2A9DE-131
grant_type	authorization_
code	4zwOADzwUUK
redirect_uri	http://anarbitra

```
JSON
  access_token=eyJ0eXAiOiJKV1QiLC
  expires_in=3600
  token_type=bearer
```

OAuth Tokens

Authorization Code

Access Token

Refresh Token

OAuth Flows

Authorization Code Flow

Aka 3-legged OAuth
Client must be capable of storing secrets

Implicit Flow

Less secure
No refresh tokens
For clients that cannot store secrets (e.g. SPA written in JavaScript)

Resource Owner Password Flow

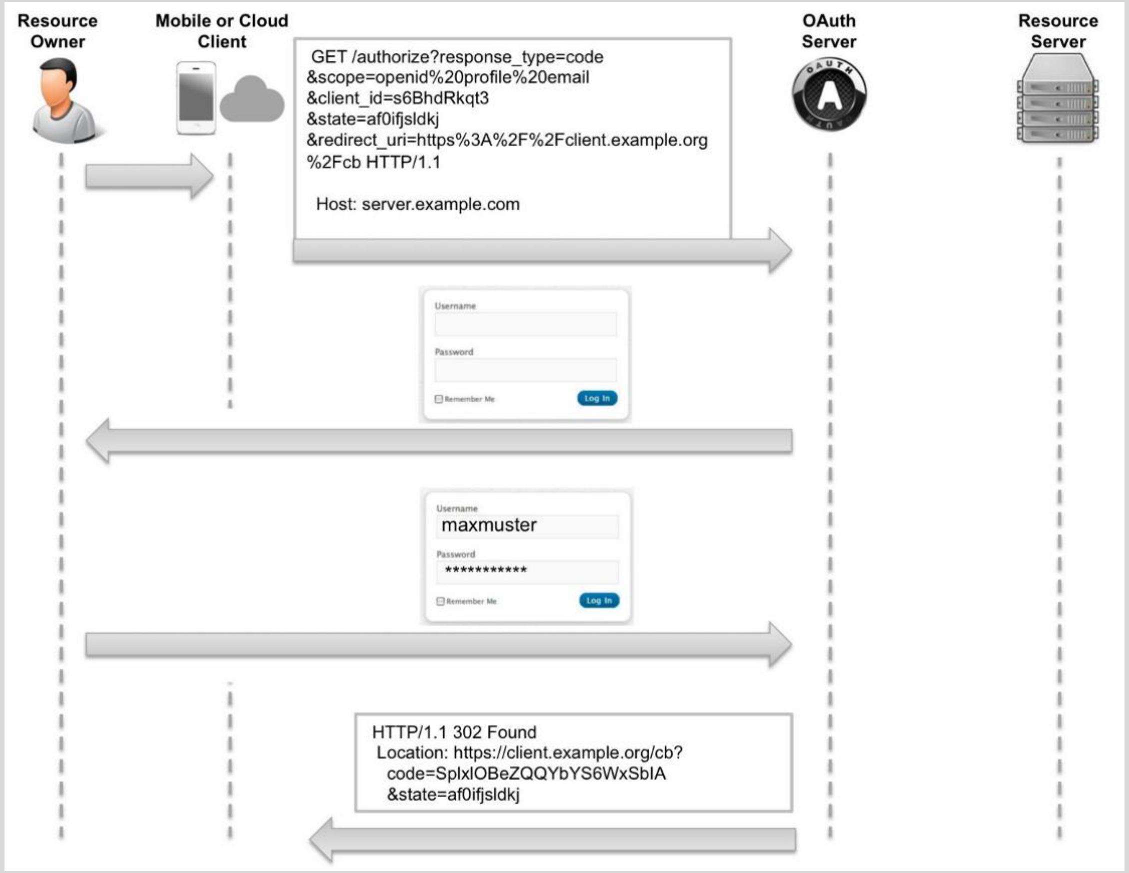
For trusted clients

Client Credential Flow

Aka 2-legged OAuth
Client is also the resource owner

Authorization Code Flow

Getting the auth code



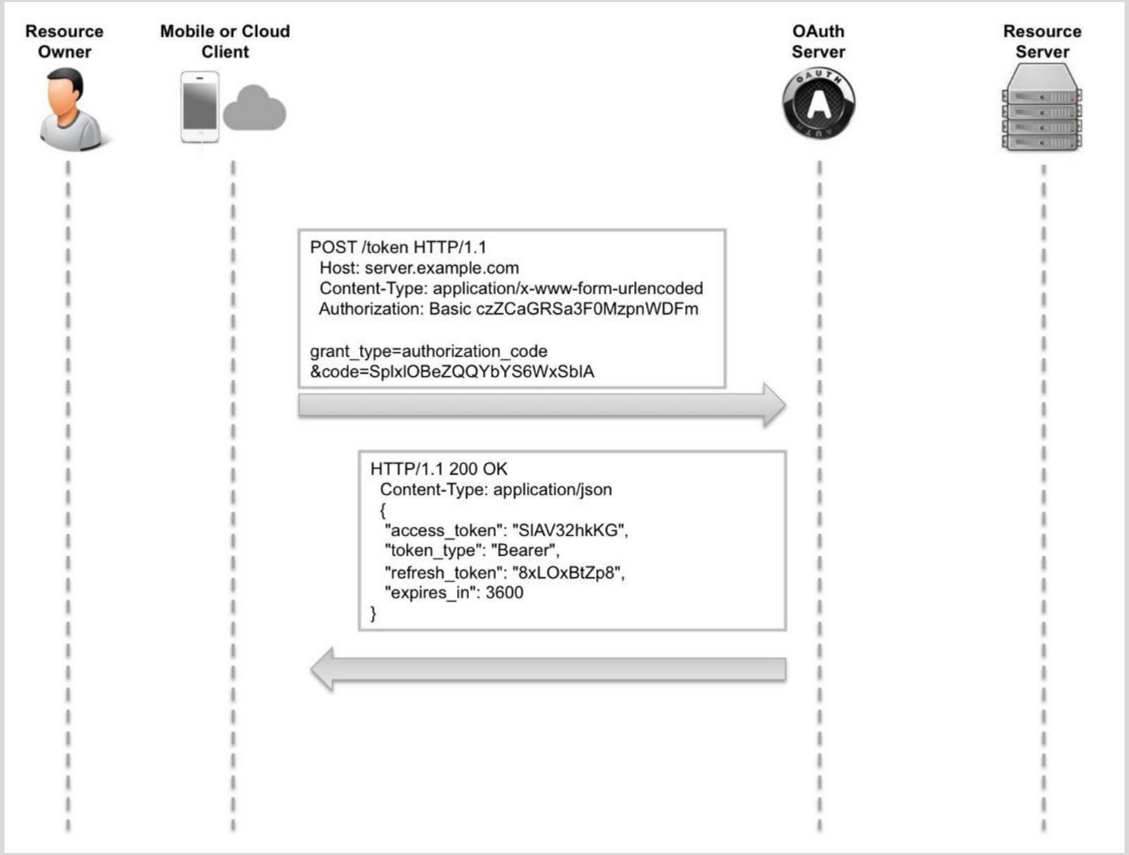
```
GET /authorize?response_type=code
&scope=openid%20profile%20email
&client_id=s6BhdRkqt3
&state=af0ifjsldkj
&redirect_uri=https%3A%2F%2Fclient.example.org
%2Fcb HTTP/1.1

 Host: server.example.com
```

```
HTTP/1.1 302 Found
 Location: https://client.example.org/cb?
   code=SplxlOBeZQQYbYS6WxSbIA
   &state=af0ifjsldkj
```

Source: Biehl, Matthias (2014-11-14). OAuth 2.0: Getting Started in API Security (API-University Series Book 1)

Authorization Code Flow

Getting the token



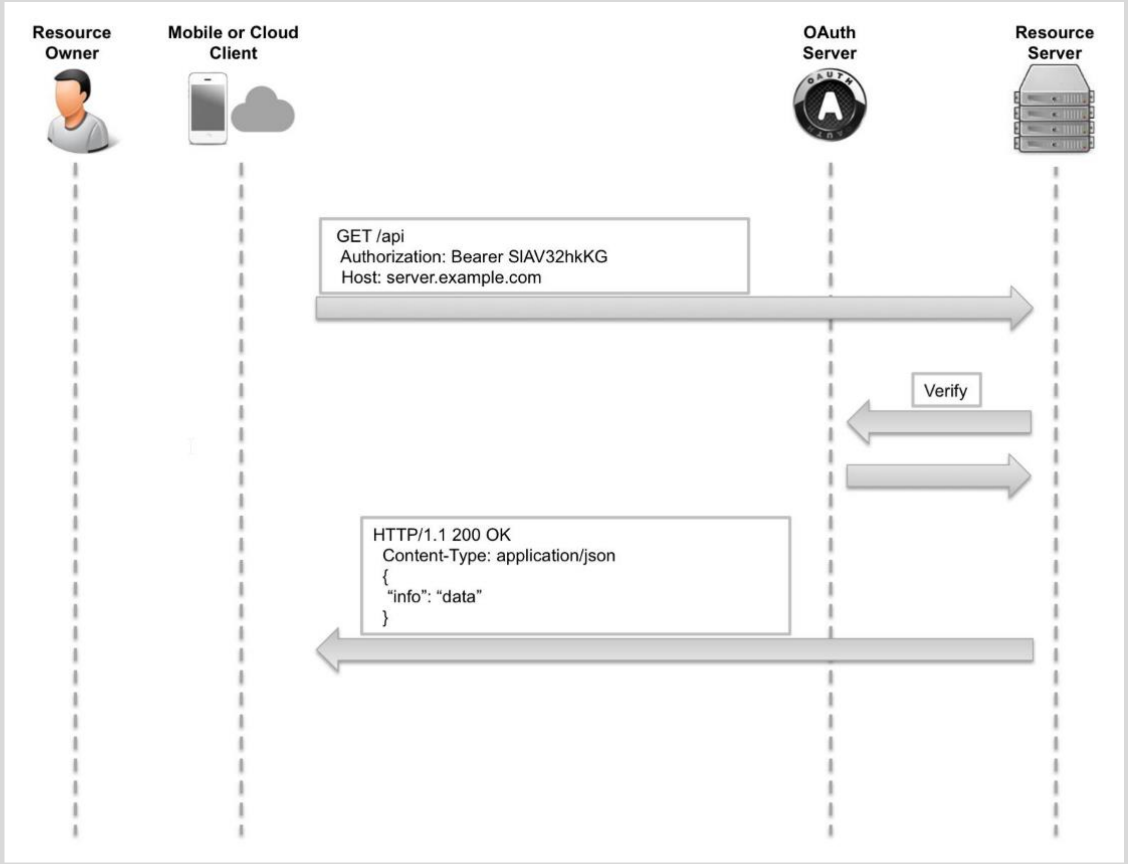
```
POST /token HTTP/1.1
  Host: server.example.com
  Content-Type: application/x-www-form-urlencoded
  Authorization: Basic czZCaGRSa3F0MzpnWDFm

grant_type=authorization_code
&code=SplxlOBeZQQYbYS6WxSbIA
```

```
HTTP/1.1 200 OK
  Content-Type: application/json
  {
   "access_token": "SlAV32hkKG",
   "token_type": "Bearer",
   "refresh_token": "8xLOxBtZp8",
   "expires_in": 3600
}
```

Source: Biehl, Matthias (2014-11-14). OAuth 2.0: Getting Started in API Security (API-University Series Book 1)

Authorization Code Flow

Accessing the resource



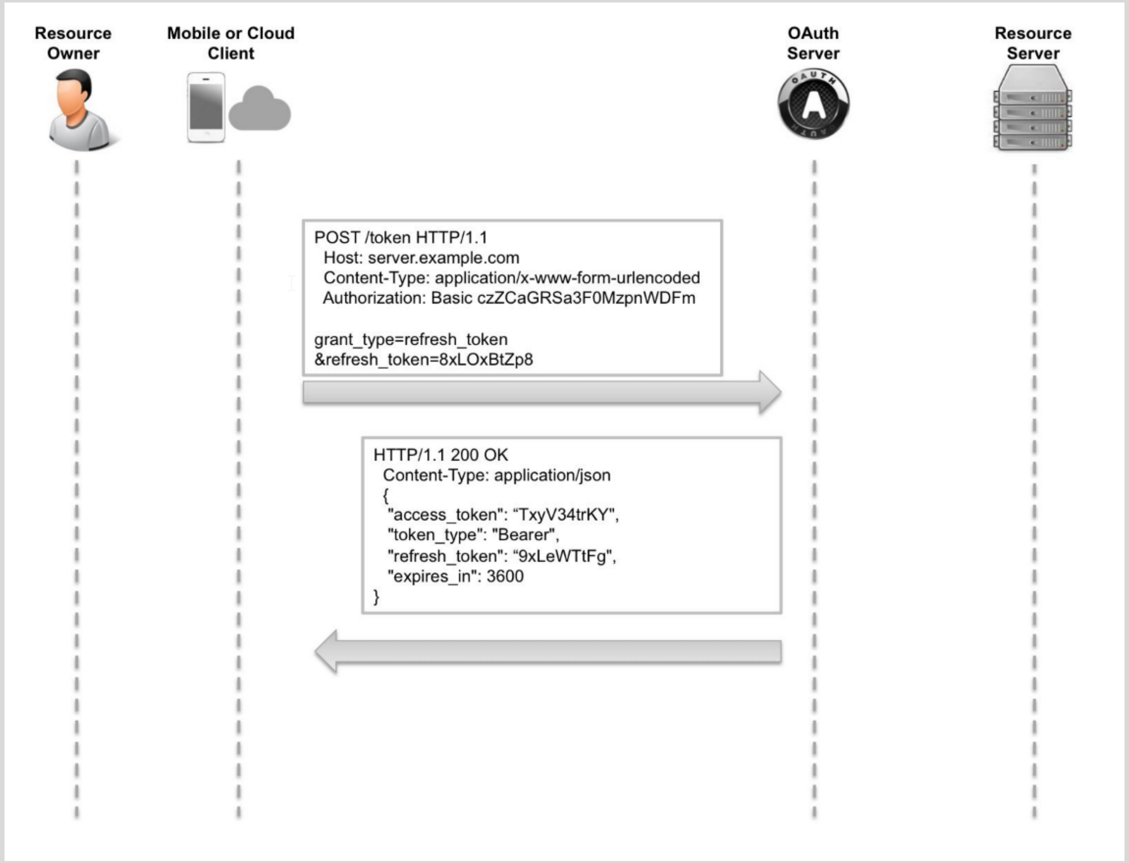
```
GET /api
 Authorization: Bearer SIAV32hkKG
 Host: server.example.com
```

```
HTTP/1.1 200 OK
  Content-Type: application/json
  {
   "info": "data"
  }
```

Source: Biehl, Matthias (2014-11-14). OAuth 2.0: Getting Started in API Security (API-University Series Book 1)

Authorization Code Flow

Refreshing the token



```
POST /token HTTP/1.1
  Host: server.example.com
  Content-Type: application/x-www-form-urlencoded
  Authorization: Basic czZCaGRSa3F0MzpnWDFm

grant_type=refresh_token
&refresh_token=8xLOxBtZp8
```

```
HTTP/1.1 200 OK
  Content-Type: application/json
  {
   "access_token": “TxyV34trKY",
   "token_type": "Bearer",
   "refresh_token": “9xLeWTtFg",
   "expires_in": 3600
}
```

Source: Biehl, Matthias (2014-11-14). OAuth 2.0: Getting Started in API Security (API-University Series Book 1)

Problems with OAuth2

Many different implementations

Not compatible

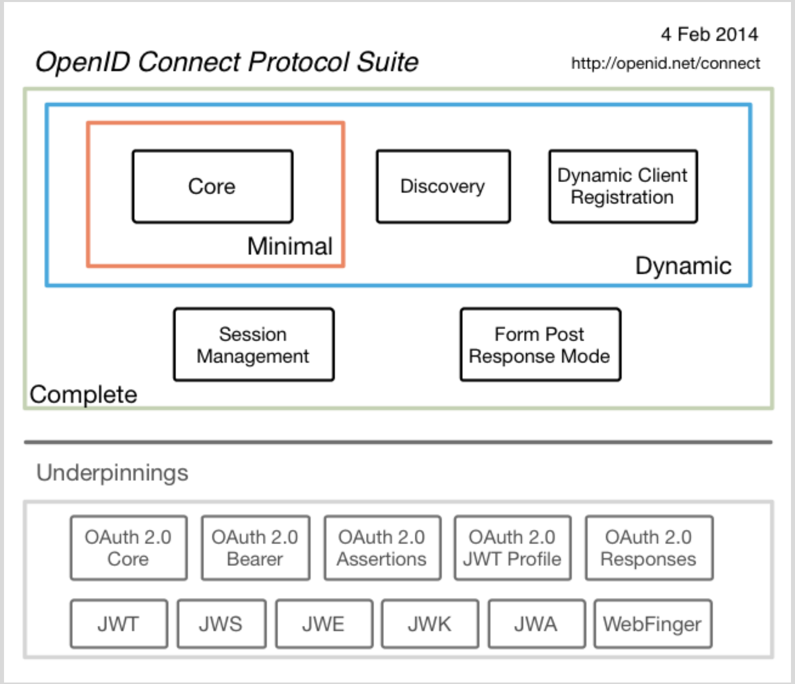
Limited scope

No specified token formats, crypto algorithms, etc.
No standard for authN, session management, etc.
No specification for token validation

Open ID Connect fills many of the gaps

Standardized way to get the resource owner's profile data
Introduces an ID-Token
Standardized token format and crypto: JWT (JSON Web Token)

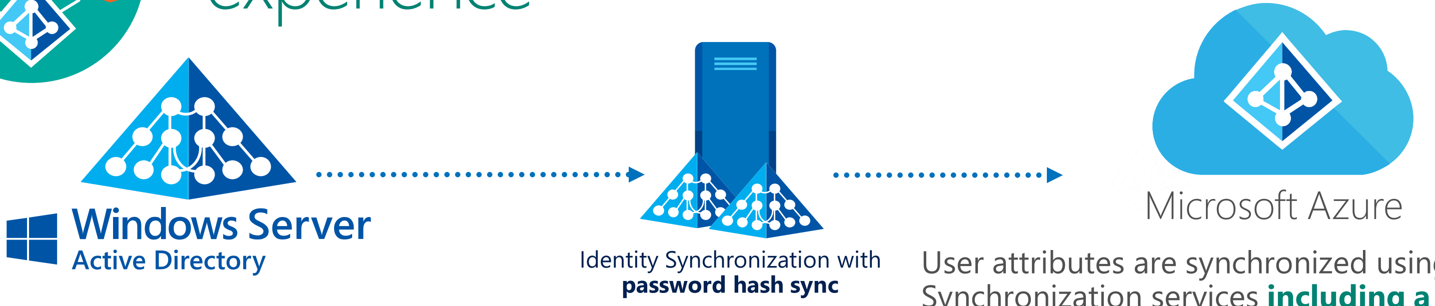
OIC Protocol



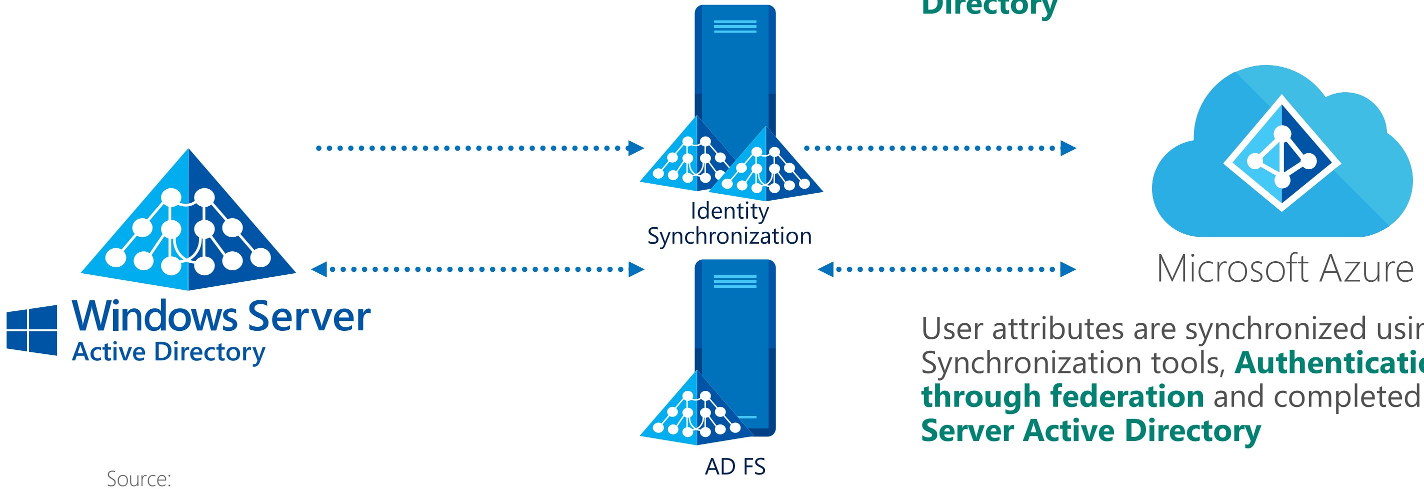
- OpenID Connect extends OAuth2
- Although rather new, OIC is already very popular
  - Libraries and products:
    - http://openid.net/developers/libraries/

Source: http://openid.net/connect/

Delivering a seamless user authentication experience



User attributes are synchronized using Identity Synchronization services **including a password hash,** Authentication is completed against **Azure Active Directory**



User attributes are synchronized using Identity Synchronization tools, **Authentication is passed back through federation** and completed against **Windows Server Active Directory**

Source:
http://channel9.msdn.com/events/TechEd/Europe/2014/CDP-B210

Standards based integrations

Custom LOB applications that integrate with Azure Active Directory

Sign in to Active Directory-integrated applications with cloud identities

Active Directory-integrated applications can access Office 365 and other web APIs

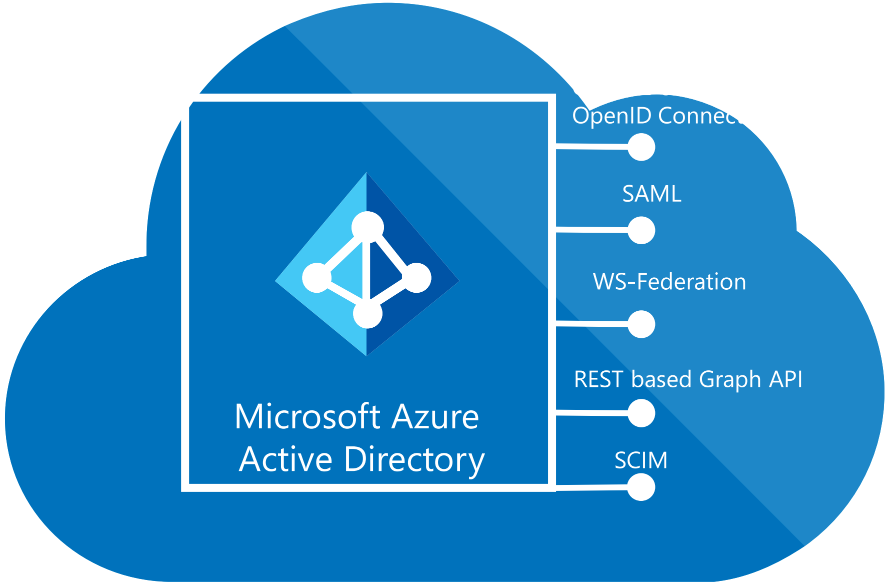
Applications can extend Azure Active Directory schema

Cross-platform support

iOS, Android, and Windows

Open Standards

SAML, OAuth 2.0, OpenID Connect, OData



Source: http://channel9.msdn.com/events/TechEd/Europe/2014/CDP-B210

Web API Metadata

The role of metadata using the examples of http://swagger.io/ and OData

Why Metadata?

Humans *and computers* discover and understand services

Less need to read documentation or source code

Enables tools for the API creator

Write less documentation manually

Make consuming the API easier → raises adoption

Enables tools for the API consumer

Build generic service consumer

  Examples: BI tools like PowerBI, workflow engines like Azure Logic Apps

Auto-generate client code/libraries

Swagger

http://swagger.io

Tools for API creators

Swagger Editor (http://editor.swagger.io/) for top-down approach
Auto-generate Swagger definition from server-side implementation
    Example: https://github.com/domaindrivendev/Swashbuckle

Tools for API consumers

Swagger UI (http://petstore.swagger.io/)
Code generators (http://swagger.io/getting-started/swagger-codegen)

Demo

Swagger

Swagger editor

Swagger code generator
(AngularJS)

OData – Much More than Metadata

http://www.odata.org

Common Schema Definition Language (CSDL)

OASIS standard

Extensible

http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part3-csdl.html

Libraries for API creators and consumers

http://www.odata.org/libraries/

Widely used at Microsoft and SAP

Examples: *Microsoft Azure, PowerBI, Visual Studio*

OData – Much More than Metadata

CRUD operations

RESTful web API

Standardized query language using URIs

```
https://api.myserver.com/odata/Customers?
  $filter=CustomerID eq 15&
  $top=10&
  $select=FirstName,LastName
```

http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html

Standardized document representation

XML (Atom), JSON

http://docs.oasis-open.org/odata/odata-json-format/v4.0/odata-json-format-v4.0.html

OData

Demo

Implementing an OData service in .NET

OData consumption
- XOData
- Power BI

Software Architecture Summit 2015

Q&A

Thank your for coming!

Rainer Stropek

software architects gmbh

Mail	rainer@timecockpit.com
Web	http://www.timecockpit.com
Twitter	@rstropek

time cockpit
Saves the day.

time cockpit

time cockpit is the leading time tracking solution for knowledge workers. Graphical time tracking calendar, automatic tracking of your work using signal trackers, high level of extensibility and customizability, full support to work offline, and SaaS deployment model make it the optimal choice especially in the IT consulting business.

Try time cockpit for free and without any risk. You can get your trial account at http://www.timecockpit.com. After the trial period you can use time cockpit for only 0,25€ per user and day without a minimal subscription time and without a minimal number of users.

time cockpit

time cockpit ist die führende Projektzeiterfassung für Knowledge Worker. Grafischer Zeitbuchungskalender, automatische Tätigkeitsaufzeichnung über Signal Tracker, umfassende Erweiterbarkeit und Anpassbarkeit, volle Offlinefähigkeit und einfachste Verwendung durch SaaS machen es zur Optimalen Lösung auch speziell im IT-Umfeld.

Probieren Sie time cockpit kostenlos und ohne Risiko einfach aus. Einen Testzugang erhalten Sie unter http://www.timecockpit.com. Danach nutzen Sie time cockpit um nur 0,25€ pro Benutzer und Tag ohne Mindestdauer und ohne Mindestbenutzeranzahl.