Article
Rob Tweed · Jul 31, 2017 7m read

JSON Web Tokens are now Supported by QEWD.js

I've written previous IDC articles about QEWD and how it provides a very high-performance, yet simple-to-use, way of integrating Node.js with Cache

I've just pushed out build 2.15 of QEWD which adds a major new capability: full support for JSON Web Tokens (JWT). I'd like to provide some background to this new development, and explain how to make use of JWTs for your Cache applications.

The key thing about JWTs is that you can use them to create completely stateless applications, with the application's state information securely held within the JWT rather than in the more "traditional" way, where you use a session structure held in a back-end database (such as Cache).

If you're not familiar with JWTs, here's a good place to start:

https://jwt.io/introduction/

Essentially, JWTs provide a secure way of maintaining user authentication and, optionally user-specific session data within a token that is held in the browser or client, not the server.

Before explaining in detail how to use JWTs in QEWD, you're probably wondering what the big deal is about them, and why you might use them in preference to the standard server-side Sessions supported by QEWD. Essentially there are two advantages:

1. a QEWD system can be scaled out to use a farm of QEWD servers. Provided they all share the same JWT secret string (see below), then it doesn't matter which QEWD server handles an incoming request - all the authentication and state information needed to handle the request is defined securely in the JWT attached to the request. So there's no need for "sticky" sessions where a user's requests can only be handled by the QEWD system that registered the session in the first place.
2. they make it possible to add proper micro-service support to QEWD. Build 2.15 includes Micro-service support, which I'll explain in a later posting. Micro-services provides another way of modularising and further scaling out your QEWD systems.
3. You can afford to give your sessions very long expiry times if this is something you want to do. Since the JWT, and therefore the session information, resides in the client/browser, there's no impact in terms of back-end database storage.

Use of JWTs is optional in QEWD. However, if you want it to be available to your applications, you need to add the following configuration parameter to your QEWD startup file:

```
jwt: {
  secret: 'myJWTSecretString'
}
```

It's up to you what your JWT secret text string contains - it's a good idea to make it a string that's not easily guessable. The secret will be used to decrypt, encrypt and sign your JWTs. Note that your existing non-JWT QEWD applications will continue to work as before, even if you enable JWTs.

You'll also need to update one other module:

- ewd-client (to build 1.17 or later)

If you're building a "traditional" web application where you load all your JavaScript manually at run-time using <script> tags, you'll need to copy the ewd-client,js file to your QEWD web server root path (eg /qewd/www)

You'll also need to load a JWT JavaScript client library into the browser, in order to access data within the JWT. Probably the most well-known one is:

https://github.com/auth0/jwt-decode

Install this locally into your QEWD web server root directory and load it into the browser, eg:

```
<script src="jwt-decode.js"></script>
```

In your application's browser-side JavaScript, you start up the application (ie once everything is loaded into the browser) using:

```
EWD.start({
  application: 'myJWTApp',
  jwt: true,
  jwt_decode: jwt_decode,
  io: io
?});
```

In other words, you need to tell the EWD client that you're using JWTs for this application, and that it should use the jwtdecode library to decode and access data within the JWT

Having done that, you just build out the client/browser-side of your QEWD application exactly as before, and the ewd-client will do the rest for you. Send your messages using EWD.send as usual. ewd-client will automatically send and maintain the JWT for you behind the scenes.

The one thing you'll need to know is how to access information in the JWT that has been sent from the back-end: the JWT payload is automatically available via the object EWD.jwt

eg to discover the application name:

```
var appName = EWD.jwt.application;
```

The JWT is updated after every request to the back-end. At the very least, the expiry time will be updated. However, the back-end QEWD message handler function may add values into the JWT that you can access in the browser.

So how about the server-side of your applications?

Again, there's not a great deal of difference from the developer's point of view. The incoming JWT will be automatically authenticated and decrypted, and its contents presented to you as the session object that you'd normally have access to, eg:

```
myRequestType: function(messageObj, session, send, finished) {
  // session is an object containing the decrypted contents of the JWT attached to th
e incoming request
}
```

Unlike when using QEWD's server-side sessions, you access properties directly within the session object, eg:

```
var application = session.application;
```

To add a property (which may be a string, numeric, boolean, object or array) to the JWT, just add it to the session object, eg:

```
session.currentDate = new Date().getTime();
```

when the JWT is returned to the browser, this property can be viewed using:

```
var currentTime = EWD.jwt.currentDate
```

A key feature of JWTs is that values (aka claims) within it cannot be arbitrarily changed within the client, as this will invalidate the signature that accompanies the JWT. This means that data that is for later use at the back-end can be safely made available to the browser. Nevertheless, there may be data that you want to store in the JWT for subsequent use at the back-end (ie when handling some subsequent request from the browser) but which you feel is too sensitive to be viewable in the browser (eg by someone using the browser's JavaScript console). For such data, you can use QEWD's session.makeSecret() function, eg:

```
session.myPrivateValue = 'a sensitive piece of info'l;
session.makeSecret('myPrivateValue');
```

All session properties that are made secret in this way are encrypted using the JWT secret key and, although sent to the browser, cannot be used or decrypted (the browser doesn't have access to the JWT secret key). All you'll see in the browser is a JWT claim named "qewd" that has an unusable encrypted value.

However, when the JWT is returned to the back-end in some subsequent request, the secret values are decrypted automatically for you, so you simply do this in a later message handler function:

```
var myPrivateValue = session.myPrivateValue;   // it's automatically decrypted and ma
de available to you!
```

User authentication is handled similarly to "standard" QEWD, ie, initially, on registration of the application, a secret JWT claim - authenticated - is set to false. You can set this to true at the back-end in your login handler method, eg:

```
session.authenticated = true;
session.timeout = 3600; // increase the session / JWT timeout to 1 hour
```

That's pretty much all there is to creating QEWD JWT Applications

Example

I've included a worked example of a simple QEWD JWT application within the QEWD repository - look in the /examples/jwt directory, eg:

https://github.com/robtweed/qewd/tree/master/example/jwt

The browser-side example code is in index.html - for ease of demonstration, all the application's JavaScript logic is inline within this file

An example startup file (qewd-jwt.js) can be found in:

https://github.com/robtweed/qewd/tree/master/example/jwt/startupfile

This startup file also shows you how to configure a micro-service - see my next posting for details.

The back-end module for the example application can be found in:

https://github.com/robtweed/qewd/tree/master/example/jwt/nodemodules

You'll see that the handler function logic is very similar to a standard QEWD application.

Conclusion

That's about it, in terms of how and why to use JWTs for your QEWD applications. JWTs are a pretty hot topic these days, and are very much seen as the modern way forward for applications. QEWD makes it simple for you to use them - I've done all the hard work for you!

The one drawback I can see for JWTs is if you need to use large volumes of state information - conveying a lot of state data between the browser and back-end on every request and response would quickly become very inefficient. If you're in that situation, you have 2 choices:

- revert to using standard back-end database-based QEWD sessions
- store your own pointer in the JWT and use that to point to the user's state data in a database of your choosing (eg Cache, using QEWD's abstraction of the cache.node APIs). The downside of this is that you'll need to figure out the maintenance and garbage-collection logic for such a database record.

But for most applications, you probably don't need that much state data, in which case JWTs are an interesting and very powerful alternative to the older server-side techniques for authentication and session management.

Have fun with JWTs, QEWD and Cache!

#Microservices #JSON #Frontend #JavaScript #Caché

Source URL:https://community.intersystems.com/post/json-web-tokens-are-now-supported-qewdjs