NodeJsRuntime

Robin De Silva Jayasinghe edited this page on 8 Sep 2015 · 11 revisions

Node.js Runtime

As part of XS2, one can also build application on plain Node.js. Typically, you might want to use Express (a "fast, unopinionated, minimalist web framework for Node.js") and testing utilities such as Mocha, ShouldJS, and Sinon JS. XS2 makes no assumption about the frameworks you are using whatsoever, however, it helps connecting to and using the HDI container and validating the JWT tokens.

Links:

- WIKI
- HANA / CloudFoundry connection tool
- Security Library
- Samples: Hello World

Using npm with SAP node.js modules

During development of the XS2 platform it was common to declare dependencies as internal github URLs in the package.json descriptor. While package.json is still the right place the dependencies have to be declared with proper npm versions. Initially it was planned to add @sap scope to all SAP modules and let users/customers setup a dedicated npm registry for that scope. As this registry will not be ready until dev close for SPS11 we decided to postpone the @sap scoping and deliver the modules with unscoped names. However, unscoped modules can only be loaded from the globally configured npm registry which is in most cases the official npmjs registry where SAP cannot publish its modules for obvious reasons. ;)

Unless SAP exposes a npm registry that supports scoped modules external users and customers have to load zipped node_modules from SMP and can then extract them manually to the node_modules folder of their application.

Example - xsenv

- Download xsenv from SMP
- Extract the archive to $your_nodejs_app_dir/node_moduels/xsenv
- Add the dependency 'xsenv' in the correct version (find it in the package.json of the downloaded module) to the packge.json of your application.

NPM handling of manually added modules

If npm detects that a declared module is already present (has been added manually) the module is ommited/ignored during processing of "npm install". That means that such modules should be at best self-contained (come with all their dependencies). As an alternative such sap-modules can

Pages 40

Table of Content

Links

- SAP HANA XS Advanced Corporate WIKI

Clone this wiki locally

https://github.wdf.sap.corp

Clone in Desktop

just be shipped with their (dependent) sap-modules in node_modules. In order to install all remaining OSS dependencies one has to call "npm install" in the directory of the given sap-module.

Deployment of applications with pre-installed node_modules

If such an application with pre-installed node_modules is pushed to the runtime environment it is important that "node_modules" is not listed in the .cfignore file of the node-backend application. Otherwise the carefully hand-crafted node_modules will be ignored and the runtime environment will fail to load the modules from the configured registry.

Important: NPM caches already processed modules. If you are in a local development scenario and want to **force** npm to use your local version and not the cached version the npm cache has to be disabled. Please set the following variable in your runtime environment (XS2 OnPremise or Cloud Foundry):

```
NODE_MODULES_CACHE=false
```

HANA HDI Container

In order to establish a connection to the database, an application must lookup the connection properties from the bound services and create a hdb client. For your convenience, you can use a simplified setup.

Add to the package.json the dependency to xs-hdb-connection:

```
{
  ...
  "dependencies": {
    "xs-hdb-connection": "git://github.wdf.sap.corp/xs2/node-xs-hdb-connection.git"
  }
}
```

Then create a connection. Internally, it looks for a service tagged as 'hana' and uses those properties to connect and set the schema correctly.

```
var xsConnection = require("xs-hdb-connection");
var hdbclient = xsConnection.createConnection(function(error) { ... });
```

Alternatively, you can create a connection pool or use an Express middlewar provided by HANA / node-xs-hdb-connection

Security

For Node.js, the client security library is a npm module called xs2sec. Add it via your package.json:

```
{
  ...
  "dependencies": {
    "xs2sec": "git://github.wdf.sap.corp/xs2/node-xs2sec.git"
  }
}
```

Check the xs2sec module for a detailed description how to use it. For the February Beta, you must use the following configuration to make it work in conjunction with the Application Router:

```
passport.use('JWT', new xs2sec.JWTStrategy({
    uaaBaseUrl : 'http://xs2-login.cf.sap-cf.com',
    containerUser : 'xs2.node',
    containerUserPassword : 'nodeclientsecret',
}));

...

app.use('/', passport.authenticate('JWT'), routes);
```

The user object is structured according to User Profile convention in Passport. It is consistent with other Passport strategies.

Outbound connectivity

In case the application code in Node.js needs to call external applications or services the only legal and technically possible way to do so is performing HTTP requests to those services. They can either be located in the same system (XS2 on-premise or Cloud Foundry) or somewhere out in the internet. In case of node.js applications we recommend the usage of the request module (https://github.com/request/request).

There are two things to be considered:

- In case your applications runs behind an HTTP proxy the proxy connection information needs to be provided to the request module. Request can pick up this information from the applications environment (you can set environment variables in manifest.yml of your application) or directly provide it in the options object of the HTTP request. The documentation for the request module should be sufficient. :)
- If the application needs to call a service that performs authentication via SAML and JOT and the calling service does so as well the SAML token can be simply taken from the incoming request and be attached to the outgoing request.

```
var options = {
    url: remoteURL,
    headers: {
        'authorization': req.headers.authorization
    }
};
```

The only boundary condition for this so called principal propagation is that both the calling and the called application are bound to the same UAA service. Otherwise the SAML token attached to the request will not be recognized and the request cannot be authenticated.

XS ✌ | Overview | Samples | 📘 SAP HANA XS Advanced Corporate WIKI