IBM SDK for Node.js - z/OS, V6.0

User's Guide

Version 6 Release 0

SC27-9054-00

IBM SDK for Node.js - z/OS, V6.0

User's Guide

Version 6 Release 0

SC27-9054-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 25.

This edition applies to Version 6 of IBM SDK for Node.js - z/OS (5655-SDK) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: November 27, 2018

Contents

Chapter 1. Overview of IBM SDK for Node.js - z/OS

The IBM® SDK for Node.js - z/OS® is an extended implementation of the Node.js runtime. Node.js extends the JavaScript programming language with a set of useful server-side APIs to provide a programming platform that allows efficient development of real-time, scalable server-side network applications.

Node.js is a runtime framework that enables you to write server applications that use the familiar syntax and structures of JavaScript. Node.js extends JavaScript by providing features such as:

- The creation, modification, and deletion of local (server-side) files
- Event-driven behavior
- Concurrency
- A very small "footprint" on the server

The SDK is built from the community source Node.js and provides the following extensions:

- Support to IBM platforms
- Additional Reliability, Availability, and Serviceability (RAS) features

These extensions do not alter any of the core Node.js APIs or function.

Chapter 2. Understanding IBM SDK for Node.js - z/OS

The IBM SDK for Node.js - z/OS is a server-side JavaScript runtime environment. You can create applications and tools by using the JavaScript language and Node.js APIs, then use the SDK to deploy them onto your z/OS system.

JavaScript is a pillar of web applications, and a familiar language to many web and front-end developers. With SDK for Node.js - z/OS, the same skills and code can now be shared to create an end-to-end JavaScript stack. JavaScript and SDK for Node.js - z/OS offer a versatile platform that typically improves speed of development, delivering scalable applications in fewer lines of code.

A prominent use of IBM SDK for Node.js - z/OS is to develop network applications that provide a web portal or http endpoints, while orchestrating services and data in the backend. SDK for Node.js - z/OS provides a non-blocking, event-driven, single-threaded approach that exploits z/OS's asynchronous I/O capabilities to achieve scalability. Instead of having dedicated threads to handle server connections, a network request coming into the Node.js application will result in an asynchronous I/O event, which will then trigger the corresponding handler function within the Node.js application. As a result, Node.js applications can typically scale well to large number of connections, while maintaining low memory and CPU footprint.

The IBM SDK for Node.js - z/OS provides two main executables in the z/OS UNIX System Services (z/OS UNIX) environment - `node` and `npm`. A basic Node.js application provides an initial JavaScript file that serves as the entry point of the application, which is typically named `server.js` or `app.js`. To invoke the Node.js application, you need to execute the following command from z/OS UNIX:

```
$> node server.js
```

Node.js applications tend to be developed in smaller modules that are managed by the built-in `npm` tool. This style discourages monolithic applications, and encourages better encapsulation and reuse that is amenable to agile development, micro-services, and APIs. Module dependencies are tracked within a `package.json` file, which also contains other versioning and meta-data about the application. The `npm` tool will parse the `package.json` file and install the necessary dependencies to build the application.

IBM SDK for Node.js - z/OS also supports a native add-on feature, which allows C/C++ code to be bound as part of the JavaScript module. This feature is useful for invoking native drivers or existing assets that are not written in JavaScript language. The SDK for Node.js - z/OS includes a 64-bit C/C++ compiler (`njsc/njsc++`) with C++11 language support to facilitate the compilation of native add-on modules.

For an example of how to build your own sample Node.js application, refer to Chapter 5, “Getting started with IBM SDK for Node.js - z/OS,” on page 9.

Chapter 3. Planning

You must meet the following requirements for installing and running the IBM SDK for Node.js - z/OS.

Hardware and software requirements

IBM SDK for Node.js - z/OS runs on the following IBM Z® servers:

- z14™
- z13® or z13s®
- zEnterprise® EC12 or zEnterprise BC12
- zEnterprise 196 or zEnterprise 114

IBM SDK for Node.js - z/OS requires z/OS V02.02.00 with a service level that satisfies APAR UI46658, or z/OS V02.03.00. Integrated Cryptographic Services Facility (ICSF) must be enabled on systems where SDK for Node.js - z/OS is run.

Installation requirements

Refer to the Program Directory for detailed information.

Chapter 4. Installing and configuring

The IBM SDK for Node.js - z/OS is available as an SMP/E installable from Shopz.

The Program Directory for the product details specific installation requirements and instructions in Chapters 5 and Chapter 6. For information about the latest APAR fixes, see Fix list for IBM SDK for Node.js - z/OS.

The following checklist summarizes the key configuration steps for a successful installation.

Hardware prerequisites

zEnterprise 196, zEnterprise 114, or newer.

Software prerequisites

- z/OS V2R2 with PTF UI46658, z/OS V2R3, or higher.
- Integrated Cryptographic Services Facility (ICSF) must be enabled on systems where SDK for Node.js is run. For details, refer to ICSF System Programmer's Guide (SC14-7507) and ICSF Administrator's Guide (SC14-7506).
- Python 2.7.13 or higher that is provided by Rocket Software. Note that Python 3.x is not compatible.
- GNU Make 4.1 or higher that is provided by Rocket Software.

Configuration

IBM SDK for Node.js - z/OS is an OMVS-based application, which requires certain configuration on the UNIX System Services file system to ensure proper operation.

- For z/OS V2R2, to confirm whether PTF UI46658 is installed, validate the timestamp of the `/usr/include/xutility` file. If the time stamp is earlier than April 26, 2017, the PTF is not installed.
- Validate that `/usr/bin/env` exists. If not configured, refer to the instructions in Chapter 8, "**Verifying the env command path**," on page 15.
- Ensure that `/tmp` has at least 1 GB or more of disk space configured. To use an alternative file system, you can set the `TMP` environment variable to a directory that has sufficient space.

Environment variables

You need to set the following environment variables before using IBM SDK for Node.js - z/OS.

- Configure the `PATH` environment variable to include the `bin` directories for IBM SDK for Node.js - z/OS, Python, and GNU Make tools with the following command:

  ```
  export PATH=$PATH:$PathPrefix/usr/lpp/IBM/cnj/IBM/node-latest-os390-s390x/
  bin:$PathPrefix/usr/lpp/IBM/cnj/njsc/bin:/rsusr/rocket/bin
  ```

  Note: `$PathPrefix` can be undefined for default installation under `/usr/lpp/IBM/cnj/IBM`. If the product is not installed in the default installation location, contact your system administrator for the `$PathPrefix` value and

ensure the installation was performed properly. For detailed information, see section 6.2.1 in Program Directory (GI13-4703-00).

- The C/C++ environment comes pre-configured when using **npm** to build native add-ons. However, if you invoke **node-gyp** directly to build native code, the following C/C++ compiler environment variables need to be set:

```
export CC=eb2as.sh
export CXX=eb2as.sh
export LINK=eb2as.sh
export _C89_CCMODE=1
export _CC_CCMODE=1
export _CXX_CCMODE=1
```

npm configuration

The **npm** utility is included in IBM® SDK for Node.js - z/OS® to install Node.js modules and packages. The **npm** utility performs checks to limit unsafe installation of modules by root / BPXROOT. To proceed to use BPXROOT id, you can take either of the following steps:

- Run with the **--unsafe-perm** npm option. For example:

  ```
  npm install <npm_module> --unsafe-perm
  ```

  You can configure this option as default with:

  ```
  npm config set unsafe-perm true
  ```

- Create a user id: `nobody` and ensure it is a member of a group. **npm** switches to this `nobody` uid/gid as necessary when running as BPXROOT.

Chapter 5. Getting started with IBM SDK for Node.js - z/OS

To get started with IBM SDK for Node.js - z/OS, you can work with the Node edition of a "Hello World" program as an example. You can set up a simple web-server that responds to a GET request with the `Hello World` message through the following steps:

1. Under z/OS UNIX System Services (z/OS UNIX), create a separate HFS directory for the example "Hello World" program.

   ```
   $>mkdir myapp
   $>cd myapp
   ```

2. Set up the project by using the `npm` executable and filling out the requested details to populate a `package.json` file.

   ```
   $>npm init
   ```

3. Node.js boasts an impressive 450k+ open source add-on modules, which can be used to develop applications. You can use the Express® web framework for the web-server example. To install the Express web framework, you can use the `npm` executable.

   ```
   $>npm install express --save
   ```

4. After the Express web framework is installed, a `node_modules` directory is added under the `myapp` directory.

5. Write the following sample code into a `server.js` file.

   ```
   /*Load express and instantiate, get the port number from command line*/
   var express = require("express");
   var server_instance = express();
   var port = process.argv[2];

   /*Respond to get requests to root dir*/
   server_instance.get("/", function (req, res){
       res.send("Hello World\n");
   })

   /*Start webserver*/
   server_instance.listen(port,function() {
       console.log("Listening on: " + port);
   })
   ```

6. Start the web-server application from the z/OS UNIX prompt.

   ```
   $>node server.js 1339
   Listening on: 1339
   ```

7. Open a web browser or use the `wget` command to access the server through HTTP on port 1339. The `Hello World` message is returned.

   ```
   $>wget https://<server address>:1339
   cat index.html
   Hello World
   ```

Through the previous steps, a simple web server is up and running.

Chapter 6. Debugging

You can debug your IBM SDK for Node.js - z/OS applications by using standard debugging techniques. In general, the process for debugging IBM SDK for Node.js - z/OS applications is much like that for debugging applications written in other server-side scripting languages. For example, you can use a debug mode on your application to inspect status and variable content, interrupt execution at specific breakpoint or when defined conditions or exceptions are encountered.

Node.js has a built-in debugger client that allows you to set breakpoints, step through code, and perform other debug operations. For more information about the client, refer to the Node.js documentation.

Chapter 7. Running

After you install and configure an IBM SDK for Node.js - z/OS application, you can run the application.

All IBM SDK for Node.js - z/OS applications are run in the same way:

1. Invoke the IBM SDK for Node.js - z/OS runtime binary file.
2. Supply the name of the required application

For example, you can use the following command to run an IBM® SDK for Node.js - z/OS application:

```
node myapplication.js
```

If you want the IBM SDK for Node.js - z/OS runtime binary file to be invoked when your system starts, you can create a startup (shell) script that is called during system startup.

More arguments can be supplied to your application by appending them to the invocation command. For example,

```
node applicationexample.js args1 args2 args3
```

Each of the arguments is available to your application by using the `process.argv` array.

- The first element in the array (`process.argv[0]`) is the name of the IBM SDK for Node.js - z/OS runtime binary file: `node`.
- The second element in the array (`process.argv[1]`) is the name of the IBM SDK for Node.js - z/OS application script. In this example, `process.argv[1]` has the value `applicationexample.js`.
- Any remaining elements in the array correspond to other arguments that are supplied on the command line.

Chapter 8. Verifying the env command path

The shell scripts for IBM SDK for Node.js - z/OS require `/usr/bin/env`, but your system might have only `/bin/env`. You can take the following steps to verify the path for the **env** command.

1. Ensure that `/usr/bin/env` exists and provides a correct listing of the environment. In an SSH or Telnet shell environment, run the following command to verify the location and contents of **env**. The command returns a list of name and value pairs for the environment in your shell.

   ```
   /usr/bin/env
   ```

   If `/usr/bin/env` does not exist, complete the following steps to set it up:

   a. Locate the **env** program on your system. A potential location is in `/bin/env`.
   b. Create a symbolic link (symlink) so that `/usr/bin/env` resolves to the true location of **env**. For example:

      ```
      ln -s /bin/env /usr/bin/env
      ```

   c. In an SSH or Telnet shell environment, run the following command to verify that the symlink works. The command returns a list of name and value pairs for the environment in your shell.

      ```
      /usr/bin/env
      ```

2. Verify that the symbolic link for the **env** command persists across system IPLs.

   Depending on how `/usr/bin/` is configured on your system, the symbolic link for `/usr/bin/env` might not persist across an IPL without additional setup. Ensure that your IPL setup includes creation of this symbolic link, if necessary.

Chapter 9. Tagging files

The IBM SDK for Node.js - z/OS runtime is a native z/OS application that, by default, treats any files on the file system as an EBCDIC text file, unless otherwise tagged. Following the specification of JavaScript (ECMAScript), any text is converted to an UTF-8/UTF-16 internal representation within the JavaScript engine. If a Node.js application is hosting a web server (that is, http or Express-based application), text data streamed to client connections is in UTF.

For input files that might not be EBCDIC text (that is, ASCII text files or binary files), you can use the chtag utility to properly tag such files. The IBM SDK for Node.js - z/OS runtime queries any file tags and respects the encoding that is specified. Failure to provide proper tagging may result in corrupted input data that undergoes an unnecessary EBCDIC to ASCII conversion.

Binary files support

To tag a file as binary, use the following command:

```
chtag -b <path/to/binary/file>
```

To verify that the file has the binary tag, use the following command:

```
ls -T <path/to/binary/file>
```

You will get the following output:

```
b binary   T=off  path/to/binary/file
```

Enhanced ASCII support

Some applications take advantage of Enhanced ASCII support, which requires ASCII encoded text files to be tagged as ASCII text files. The IBM SDK for Node.js - z/OS runtime also supports reading files that are tagged as ASCII text files.

To tag a file as an ASCII text file, use the following command:

```
chtag -tc ISO8859-1 <path/to/ascii/text/file>
```

To verify that a file is tagged as an ASCII text file, use the following command:

```
ls -T <path/to/ascii/text/file>
```

You will get the following output:

```
t ISO8859-1   T=on  path/to/ascii/text/file
```

Usage

If you have some source files on an ASCII platform and you want to use them on z/OS, you can tag those files with the following steps:

1. Create a zip file of your source files on the ASCII platform.
2. Unzip the zip file on z/OS.
3. Tag all text files using the following command:

   ```
   chtag -tc ISO8859-1
   ```
4. Tag all binary files using the following command:

```
chtag -b
```

To copy files remotely from an ASCII platform to z/OS, you can use the **scp** command, which converts every file from ASCII to EBCDIC as it copies. In this case, tagging is not necessary.

Related information

Using git for z/OS with GitHub

Chapter 10. Troubleshooting

This chapter describes some common issues that you might encounter with the IBM SDK for Node.js - z/OS runtime. For JavaScript debugging and troubleshooting, refer to Chapter 6.

z/OS UNIX System Services Semaphores

Node.js uses z/OS UNIX System Services semaphores. If the Node.js runtime ends unexpectedly, stale semaphores might be left on the system, potentially resulting in a lack of available semaphores for invoking more Node.js instances. Currently, you must manually clean up stale semaphores, by using the **ipcrm** command. Too many stale semaphores results in the following node assert or error:

```
CEE3204S The system detected a protection exception (System Completion Code=0C4).
         From entry point _ZN4node6AssertEPA4_KPKc at compile unit offset
         +0000000025F3AC86 at entry offset +00000000000000BE at address
         0000000025F3AC86.
```

Integrated Cryptographic Service Facility not started - hang

If the node process hangs without any output check, you can run the following command to confirm whether `/dev/random` is functional:

```
cat /dev/random | od | head
```

If `/dev/random` is not available, ensure that Integrated Cryptographic Service Facility(ICSF) is started.

C/C++ Native add-ons assumptions

Ensure there are no implicit addressing mode dependencies unless they are 64-bit assumptions. If there are non IBM-1047 code page assumptions, the code must be updated to work with that code page. Ensure that only supported compiler options are used during compilation. The man or help page for the compiler can be used to list the supported options.

Forceful termination

If the node process needs to be killed forcefully, you can try sending an abort signal by using the following command:

```
kill -3 <process_id_ of_node_process>
```

The process ID can be obtained using the **ps** command.

Diagnostic report for Node.js

The node-report npm module is available for the IBM SDK for Node.js - z/OS. It provides human-readable diagnostic data that is written to a file. A report is created automatically when an unhandled exception or fatal error event (such as an out of memory error) occurs. You can also trigger a report, for example by sending a USR2 signal to a Node.js process, or by an API call from a JavaScript application.

For more information, see the github repository for the node-report module. Further information and examples are available on the Monitoring & Diagnostics page and the Blogs page of the Node.js at IBM Developer Center.

Chapter 11. Known issues and limitations

IBM SDK for Node.js - z/OS has the following known issues or limitations:

- The current release requires an address space of at least 700 MB. Use the **ulimit -A** command to set or display the maximum size of the address space.
- The `Node.js fs.watch` API on subdirectories is not supported on z/OS.
- You cannot use IBM Monitoring and Diagnostic Tools – Interactive Diagnostic Data Explorer.
- There is no Node Application Metrics component in this release, so you cannot use IBM Monitoring and Diagnostic Tools – Health Center to monitor your applications.
- Node Inspector is disabled in this release. There is a built-in debugger client that can be used for debugging.
- Some C++11 and C11 features are not supported or have limited support:
  - Thread local storage - not supported
  - Thread specific locales - not supported
  - Atomics - limited support (non-lock-free atomics are not supported)

Chapter 12. Support

To find help about the IBM SDK for Node.js - z/OS, it is important to collect as much information as possible about your installation configuration.

To establish what version of the IBM SDK for Node.js - z/OS is in use, run the following command:

```
node --version
```

The version of Node.js is displayed.

To get more details about the exact build of the IBM SDK for Node.js - z/OS, run the following command:

```
node --help
```

The IBM SDK for Node.js - z/OS includes the Node.js `npm` utility for working with modules. To establish what version of `npm` is in use, run the following command:

```
npm --version
```

The version of the Node.js `npm` utility is displayed.

Contacting IBM

IBM welcomes your comments. You can add comments or issues in Github: https://github.com/ibmruntimes/node/issues

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Corporation
J74/G4
555 Bailey Avenue
San Jose, CA 95141-1099
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors.

Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this publication to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at https://www.ibm.com/legal/us/en/copytrade.shtml.

Node.js is a trademark of Joyent Inc. IBM SDK for Node.js - z/OS is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Product Number: 5655-SDK

Printed in USA

SC27-9054-00