Course Exercises Guide

Developing REST APIs with Node.js

Course code VY102 ERC 6.0

IBM Training

May 2019 edition

Notices

Trademarks

Contents

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

Bluemix®	DB™	Express®
Notes®	z/OS®	

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

LoopBack® is a trademark or registered trademark of StrongLoop, Inc., an IBM Company.

Other product and service names might be trademarks of IBM or other companies.

Exercises description

This course includes the following exercises:

- Install, verify, and develop a Node application
- Develop a REST API with Node.js
- Static code analysis and unit testing
- Debug and build node applications
- Deploy a REST API on IBM Cloud

In the exercise instructions, you can check off the line before each step as you complete it to track your progress.

Most exercises include required sections, which should always be completed. It might be necessary to complete these sections before you can start later exercises.

If you have sufficient time and want an extra challenge, some exercises might also include optional sections that you can complete.

Information

This course was tested on an Ubuntu 16.04 Linux system.

If you are running the lab exercises on your own workstation on the Windows, Mac OS, or Linux operating systems, you should be able to follow the steps in the exercise guide with minor modifications.

The exercise with Node Inspector assumes that the Google Chrome browser is already installed. No instructions are provided to install the Chrome browser.

On some Linux systems, you might need administrative level access to install the software and complete the exercises.

Important

Online course material updates might exist for this course. To check for updates, see the Instructor wiki at: http://ibm.biz/CloudEduCourses

Exercise 1. Installing, verifying, and developing a Node application

Estimated time

00:45

Overview

In this exercise, you set up the Node.js runtime environment on your own workstation. With the Node.js runtime, you can develop and test Node applications in a local environment. The Node.js runtime environment is a prerequisite to creating APIs with the LoopBack framework.

What is the user story

As an API developer, you want to install, verify, and test the Node runtime environment on your local workstation so that you can develop and test interaction services locally.

Objectives

After completing this exercise, you should be able to:

- Install the Node.js runtime environment on a local workstation
- Verify the setup of the Node.js runtime environment
- Verify the setup of the Node package manager, npm
- Update the Node package manager on your workstation
- Define a package manifest file
- Install a third-party package in a Node application
- Start the Node interactive shell
- Run a Node application

Introduction

The Node.js runtime environment runs server-side applications that are written in the JavaScript programming language. While client-side JavaScript applications respond to user events in a web browser, Node applications respond to network events on a web server.

As an interpreted scripting language, you do not need to compile a Node application before you run its code. You can build a Node web application with a text editor and the Node interpreter for your computing platform.

In this exercise, you install the Node.js runtime environment in your workstation. The installation of the Node runtime also sets up npm, the Node.js package manager. You create a sample web application to test the runtime environment.

Requirements

This exercise requires a workstation with internet access. You can complete this exercise on a computer with a Linux, Mac OS X, or Microsoft Windows operating system.

For a list of supported operating systems and platforms, see: https://nodejs.org/en/download/

Exercise instructions

Preface

- The authors of this exercise tested the instructions on a Linux operating system. Unless otherwise specified, the instructions should work with minor modifications on Mac OS X and Microsoft Windows operating systems.
- The command-line examples work in a UNIX-compatible shell environment, such as Linux or Mac OS X. For example, use a text editor of your choice when the instructions list the gedit editor as an example.

1.1. Install the Node.js runtime environment

In this section, download and install the Node.js runtime environment on your computer. You require a runtime environment to run and test the Node applications that you write on your local workstation.

__ 1. Download the Node.js installer.

__ a. Open https://nodejs.org in a web browser.

__ b. Click the **Downloads** link.

__ c. Select the list of **LTS** (Long Term Support) installers.

__ d. Download the Node.js LTS installer package for your operating system and processor architecture (32-bit or 64-bit) using one of the following steps.

W **Windows**

Download the **Windows Installation (.msi)** package.

__ a. Run the Windows Installer package.

__ b. Install the Node.js runtime to the default program files directory.

__ c. Skip forward to the next section of this exercise.

M **Mac OS**

Download the **Mac OS X Installer (.pkg)** package.

__ a. Run the Mac OS X Installer package.

__ b. Install the Node.js runtime to the default installation directory.

__ c. Skip forward to the next section of this exercise.

L **Linux**

Download the **Linux Binaries (.tar.xz)** package.

__ a. Open a terminal window.

__ b. Expand the contents of the binary archive to the `/usr/local` directory.

```
$ sudo tar -C /usr/local --strip-components=1 -xf
node-vx.y.z-linux-x64.tar.xz
```

__ c. Confirm that the files decompressed successfully into the `/usr/local/-vx.y.z-linux-x64` directory.

__ d. Add the `/usr/local/-v`*`x.y.z`*`-linux-x64/bin` directory to the path
```
$ sudo nano /etc/environment
PATH="/usr/local/node-vx.y.z-linux-x64/bin:/usr/local/sbin:/usr/local/bin
:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games"
```

__ e. Save the change to the environment variable.

__ f. Log out from the user.

__ g. Sign on to the same user.

__ h. Skip forward to the next section of this exercise.

NOTE: If you download the Node 10.16.0 LTS version and run the command to expand the contents to the `/usr/local` directory, then the Linux system automatically adds node and npm to the `/usr/local/bin` directory. You do not need to update the `$PATH` variable.

1.2. Verify the Node.js runtime environment

Before you develop your first Node application, make sure that you can run the node and npm applications from the terminal or command prompt.

__ 1. Verify the node application version.

__ a. Open a terminal (Mac OS, Linux) or command prompt (Microsoft Windows).

__ b. Check the version and location of the node application.

```
$ node -v
v10.15.3
$ which node
/usr/bin/node-v10.15.3-linux-x64/bin/node
```

__ c. Check the version and location of the npm application.

```
$ npm -v
6.4.1
$ which npm
/home/localuser/node-v10.15.3-linux-x64/bin/npm
```

Note

The version of your node and npm applications might be newer than the ones listed earlier. The purpose of this test is to confirm that you can start the node and npm applications from any directory. The exact version numbers do not need to match.

1.3. Test the Node.js interactive shell

When you run the node command without any parameters, the application starts the Read, Evaluate, Print, Loop (REPL) environment. Use REPL to run JavaScript commands interactively and review the results.

__ 1. Start the REPL environment.

__ a. Run node application without parameters.

```
$ node
```

__ 2. Run a line of JavaScript code in the REPL environment.

__ a. Display the current date and time in the `console.log` object.

```
> console.log(new Date());
2019-05-27T21:40:12.250Z
```

__ 3. Examine information about the Node runtime process.

__ a. Check the version of Node.js currently running.

```
> process.release
{ name: 'node',
{ lts: 'Dubnium',
  sourceUrl:
'https://nodejs.org/download/release/v10.15.3/node-v10.15.3.tar.gz',
  headersUrl:
'https://nodejs.org/download/release/v10.15.3/node-v10.15.3-headers.tar.g
z' }
```

__ b. Print the location of the Node.js executable.

```
> process.execPath
'/usr/local/node-v10.15.3-linux-x64/bin/node'
```

__ c. Display the process memory use, total heap size, and heap usage, in bytes.

```
> process.memoryUsage()
{ rss: 33841152, heapTotal: 9682944, heapUsed: 6034336, external: 8786 }
```

__ 4. Exit the REPL environment.

__ a. Press Ctrl+C twice.

1.4. Set up a directory for your exercise applications

Before you write your first application, create a directory that is named `projects` to save the source code for your Node applications by using one of the following steps.

Windows

Create the projects directory in your user account directory.

__ a. Open a command prompt window.

__ b. Navigate to your user account directory.

__ c. Create a directory that is named `projects`.

```
> mkdir projects
```

__ d. Skip to the next section of this exercise.

Mac OS

Create the projects directory in your home directory.

__ a. Open a terminal window.

__ b. Navigate to your home directory.

```
$ cd ~
```

__ c. Create a directory that is named `projects`.

```
$ mkdir projects
```

__ d. Skip to the next section of this exercise.

Linux

Create the projects directory in your home directory.

__ a. Open a terminal window.

__ b. Navigate to your home directory.

```
$ cd ~
```

__ c. Create a directory that is named `projects`.

```
$ mkdir projects
```

__ d. Skip to the next section of this exercise.

1.5. Create a Node application

As an interpreted language, the simplest Node application is a set of JavaScript code that is saved in a text file. You do not need to generate or compile the code before running the script. By convention, Node developers build applications in one or more script files with a `.js` or `.node` file extension.

__ 1. Create a directory for the node application.

__ a. In a terminal (Mac OS X, Linux) or command prompt (Microsoft Windows), go to the projects directory.

```
$ cd projects
```

__ b. Create a directory that is named `status-app`.

```
$ mkdir status-app
$ cd status-app
```

__ 2. Write a JavaScript program that prints the day of the week to the console.

__ a. In the `status-app` directory, open a file that is named `today.js` in a text editor.

```
$ gedit today.js
```

__ b. Write a JavaScript program that displays the name of the week based on the `date.getDay()` function.

```
var date = new Date();
var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday'];
var today = days[ date.getDay() ];
console.log('The day of the week is %s', today);
```

__ c. Save and close `today.js`.

Note

You can choose to use the text editor of your choice to open and edit the Node package manifest and script files.

- The `Atom` editor is an open source application that is available on the Mac OS X, Linux, and Windows operating systems. See: https://atom.io
- The `nano` and `emacs` editors are more advanced editors for Linux and Mac OS users that are familiar with command-line operations.
- The `gedit` editor is a graphical text editor that is available in some Linux distributions.
- The `textedit` editor is available on the Mac OS X platform.
- The `notepad` editor is available on the Microsoft Windows operating system.

__ 3. Run the `today.js` script in Node.

__ a. In the `status-app` directory, start the node command with the script name as the parameter.

__ b. Confirm that the correct day of week appears in the console.

```
$ node today.js
The day of the week is Monday.
```

1.6. Define a node module

In the previous section, you created a Node application that consists of a single file. In most applications that you build, your application source files consist of multiple scripts, node modules, and configuration files.

The Node package manifest file, package.json, describes the main script as the entry point into your application. The manifest also lists the name, version, owner, and source code license for the application.

In this section, add a manifest file to the status application. Test the application as a module, instead of a script file.

__ 1. Define the package.json manifest file.

__ a. In the status-app directory, run the npm init utility.

```
$ npm init
```

__ b. Press Enter to accept the default values for the name and version of the application.

```
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible
defaults.

See 'npm help json' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg> --save' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (status-app)
Press Enter
version: (1.0.0)
Press Enter
Description: Return Node runtime status information
```

__ c. Accept **today.js** as the main entry point for the application.

```
entry point: (today.js)
```

__ d. Set the test command to **node today.js**.

```
test command: node today.js
```

__ e. Leave the Git repository and keywords fields as blank.

```
git repository:
keywords:
```

__ f. Type your own name and email address in the author field.

```
author: John Doe <jdoe@example.com>
```

__ g. Accept the the software license.

```
License: (ISC)
Press Enter

Is this OK? (yes)
Press Enter
```

__ 2. Review the package manifest file.

__ a. Open the `package.json` file in a text editor.

```
$ gedit package.json
```

__ a. Examine the `package.json` manifest file.

```
{
  "name": "status-app",
  "version": "1.0.0,
  "description": "Return Node runtime status information",
  "main": "today.js",
  "scripts": {
    "test": "node today.js"
  },
  "author": "John Doe <jdoe@example.com>",
  "license": "ISC"
}
```

Information

Review the application metadata in the package manifest file, `package.json`.

Name: The name of the application. By convention, the name of the directory that contains the package manifest file has the same name.

Version: The version identifier for the application. When you install a Node package to your application, you can specify a specific version or a range of versions to use.

Description: A short sentence that explains the purpose of the application.

Main: The entry point for your application. When you run the node runtime with the name of a directory with a `package.json` file, node runs the script that you specified in the main field.

Scripts: You can specify a command to execute when you run the `npm` command with a script name. For example, run `npm test` to execute the script in the test field.

Author: Enter your full name and an email address as the owner of the package.

License: The software license terms that are applied to the application. For a list of license codes, see: https://opensource.org/licenses/category/

__ 3. Add a start script to run the `today.js` script.

__ a. In the scripts object, add a `start` field with a value of `"node today.js"`.

```
"scripts": {
  "start": "node today.js",
  "test": "node today.js"
}
```

__ b. Save and close `projects.json`.

__ 4. Run the `status-app` module.

__ a. In the terminal (Mac OS X, Linux) or command prompt (Microsoft Windows), launch the package manifest start script.

__ b. Review the result in the console.

```
$ npm start

> status-app@1.0.0 start /home/localuser/projects/status-app
> node today.js

The day of the week is Monday.
```

1.7. Export a function outside of a Node script

The `exports` keyword takes a function that you define in a script and makes it accessible to other Node scripts. In this section, refactor the code in the `today.js` script as a function. Export the function in the `module.exports` field to make it available to other Node scripts.

__ 1. Export an anonymous function that returns the day of the week.

__ a. In the `status-app` directory, open `today.js` in a text editor.

```
$ gedit today.js
```

__ b. Wrap the code that returns the day of the week in an anonymous function.

__ c. Assign the anonymous function to the `module.exports` object.

```
module.exports = function() {
  var date = new Date();
  var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday', 'Saturday'];
  return days[ date.getDay() ];
}
```

__ d. Save and close `today.js`.

__ 2. Create a second script, `server.js`, which prints the day of the week from the `today.js` script.

__ a. In the `status-app` directory, create a script that is named `server.js` in a text editor.

```
$ gedit server.js
```

__ b. In `server.js`, import and save the `today` module to a variable named `today`.

```
var today = require('./today');
```

__ c. Print the day of the week to the console.

```
console.log('The day of the week is %s.', today());
```

__ d. Save and close `server.js`.

__ 3. Update the `package.json` package manifest to call `server.js` instead of `today.js`.

__ a. Open `package.json` in a text editor.

__ b. Change each instance of `today.js` to: `server.js`

```
{
  "name": "status-app",
  "version": "1.0.0",
  "description": "Return Node runtime status information",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "test": "node server.js"
  },
  "author": "John Doe <jdoe@example.com>",
  "license": "ISC"
}
```

__ c. Save and close `package.json`.

__ 4. Run the `status-app` module.

__ a. In the terminal (Mac OS X, Linux) or command prompt (Microsoft Windows), launch the package manifest start script.

__ b. Review the result in the console.

```
$ npm start

> status-app@1.0.0 start /home/localuser/projects/status-app
> node server.js

The day of the week is Monday.
```

1.8. Create a web application

The main use case for Node.js is to build server-side applications and services. In this section, display the greeting to the client's web browser instead of the console log. Learn how to handle HTTP request and response messages with the http Node package.

__ 1. Create a web application that returns the greeting in the HTTP response message.

__ a. Open `server.js` in a text editor.

__ b. Define a variable, **`http`**, which imports the `http` module.

Information

The `require` function imports the features of a Node module into your application. When you specify `require` with the module name only, the Node runtime environment searches your global and local `node_modules` directories for the specified module.

```
var http = require('http');
```

__ c. Create a server object with the `http.createServer()` function.

The `createServer()` function expects a callback function as the first parameter. The callback function has two parameters: the HTTP request and response messages.

```
var server = http.createServer( function(request, response) {

});
```

__ d. In the callback function, create the greeting with a call to the `today()` function.

__ e. Use the `response.writeHead()` function to set an HTTP response status code of OK (200).

The `writeHead()` function takes two parameters: an HTTP status code, and an object with HTTP response header names and values.

__ f. Set the HTTP `Content-Length` header to the length of the response message body. Set the HTTP `Content-Type` header to `'text/plain'`.

__ g. Compare your script with the following solution code:

```
var today = require('./today');
var http = require('http');

var server = http.createServer(function(request, response) {
  var body = "The day of the week is " + today();
  response.writeHead(200, {
    'Content-Length': body.length,
    'Content-Type': 'text/plain'
  });
});
```

__ h. Save and close `server.js`.

__ 2. Run the `status-app` module.

__ a. In the terminal (Mac OS X, Linux) or command prompt (Microsoft Windows), launch the package manifest start script.

__ b. Review the result in the console.

```
$ npm start

> status-app@1.0.0 start /home/localuser/projects/status-app
> node server.js

$
```

Questions

When you ran the `server.js` script, Node.js stopped the application without waiting for HTTP requests. Why did the framework exit immediately?

To intercept incoming HTTP requests, you must set the server object to listen on a specific port.

__ 3. Set the server object to listen to port 3000 for HTTP requests.

__ a. Open `server.js` in a text editor.

__ b. After the server variable declaration, set the server object to listen to HTTP request on port **3000**.

```
server.listen(3000);
```

__ c. Save and close `server.js`.

__ 4. Test the `status-app` in a web browser.

__ a. Launch the package manifest start script.

```
$ npm start
```

__ b. Confirm that the Node application does not immediately exit.

__ c. Open a web browser to: `http://localhost:3000`

Questions

The web browser attempts to load the web page, but the operation never completes. Why is the web browser stuck rendering the page?

The callback function must call `response.write()` or `response.end()` to send the response message to the web browser. Node.js keeps the HTTP connection open, but it does not send the data to the web browser until the application calls one of the two functions.

__ 5. Close the HTTP response connection at the end of the callback function in the `http.createServer()` call.

__ a. Open `server.js` in a text editor.

__ b. Call **`response.end(body)`** at the end of the callback function to print the message body and close the HTTP connection.

```
var server = http.createServer(function(request, response) {
  var body = "The day of the week is "+ today()+ ".";
  response.writeHead(200, {
    'Content-Length': body.length,
    'Content-Type': 'text/plain'
  });
  response.end(body);
});
server.listen(3000);
```

__ c. Save and close `server.js`.

__ 6. Test the `status-app` in a web browser.

__ a. Launch the package manifest start script.

__ b. Open a web browser to `http://localhost:3000`.

__ c. Confirm that the web application returns the day of the week in the web browser.

__ d. In the terminal (Mac OS X, Linux) or command prompt (Microsoft Windows), press Ctrl+C to exit Node.js.

End of exercise

Solution

status-app/package.json:

```
{
  "name": "status-app",
  "version": "1.0.0",
  "description": "Return Node runtime status information",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "test": "node server.js"
  },
  "author": "John Doe <jdoe@example.com>",
  "license": "ISC"
}
```

status-app/today.js:

```
module.exports = function() {
  var date = new Date();
  var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday', 'Saturday'];
  return days[ date.getDay() ];
}
```

status-app/server.js:

```
var today = require('./today');
var http = require('http');

var server = http.createServer(function(request, response) {
  var body = "The day of the week is "+ today()+ ".";
  response.writeHead(200, {
    'Content-Length': body.length,
    'Content-Type': 'text/plain'
  });
  response.end(body);
});
server.listen(3000);
```

Exercise review and wrap-up

In the first part of the exercise, you installed the Node.js runtime environment in your local workstation. With the runtime environment, you developed and tested a simple web application. You also listed the name, version, description, and third-party module dependencies in a package manifest file, `package.json`.

Exercise 2. Developing a REST API with Node.js

Estimated time

01:00

Overview

In this exercise, you develop a REST API as a Node application. You build a web application with the Express framework that handles HTTP method requests on web resources. In the implementation of your Node web application, you call remote services with the Request package. You also develop a callback function to handle the response and error message from remote services.

What is the user story

As an API developer, you want to build Node Express web applications so that you can develop and test interaction services that can be deployed to the IBM Cloud.

Objectives

After completing this exercise, you should be able to:

- Install the Express node package
- Define an Express web application
- Handle requests to web resources with Express
- Call remote services with the Request package
- Create a callback function to handle responses from remote calls
- Handle errors with a callback return parameter
- Test a callback function in a Node application

Introduction

In an earlier exercise, you built a simple web application with the http package. In real-world applications, Node developers rely on frameworks to handle common tasks, such as mapping web requests to business logic in a JavaScript function.

The Express Node application defines an application object that calls a function to handle web requests. With the http package, you must write your own code to parse the web resource and HTTP method type. In contrast, the Express package handles the mapping from a request to a handler function on your behalf.

When you build interaction services, your API operations call remote services to handle the request. The Request Node package provides an abstraction to making an HTTP request, and handling the HTTP response or error message.

In this exercise, you update the existing status application to an Express application. You create an API operation to handle requests, and you make remote service calls with the Request package.

Requirements

You must complete the steps in Exercise 1 before starting this lab.

This exercise requires a workstation with internet access. You can complete this exercise on a computer with a Linux, Mac OS X, or Microsoft Windows operating system.

For a list of supported operating systems and platforms, see: https://nodejs.org/en/download/

Exercise instructions

Preface

- The authors of this exercise tested the instructions on a Linux operating system. Unless otherwise specified, the instructions should work with minor modifications on Mac OS X and Microsoft Windows operating systems.
- The command-line examples work in a UNIX-compatible shell environment, such as Linux or Mac OS X. For example, use a text editor of your choice when the instructions list the gedit editor as an example.

2.1. Update the package manifest file

Update the minor version number in the `package.json` file to differentiate your work in this exercise with other exercises.

__ 1. Go to the `status-app` directory.

__ a. Open a terminal (Mac OS X, Linux) or a command prompt window.

__ b. Go to the `/projects/status-app` directory.

```
$ cd projects/status-app/
$ pwd
/home/localuser/projects/status-app
```

__ 2. Update the version number in the package manifest to 1.0.1.

__ a. Open `package.json` in a text editor.

```
$ gedit package.json
```

__ b. Update the version field to 1.0.1.

```
  "version": "1.0.1",
```

__ c. Save and close `package.json`.

2.2. Install the Express node package

By default, the Node framework does not provide any classes to build web applications: you must program at a network socket level and intercept HTTP requests manually. The Express node package simplifies your code by mapping HTTP requests to functions that you write.

In this section, download the Express package and its dependent libraries with the `npm install` command. Review the updates to your application package manifest and the `node_modules` directory.

__ 1. Install the Express node package.

__ a. Install the express node package with the `--save` parameter.

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

__ b. Confirm that the node installation process has no errors.

__ 2. Review the `package.json` package manifest file.

__ a. Display the contents of the `package.json` file.

```
$ cat package.json
```

__ b. Examine the dependencies section.

```
"dependencies": {
  "express": "^4.17.1"
}
```

Information

When you enter the `npm install` command with the `--save` parameter, the utility adds the version number in the dependencies section of the package manifest. When you deploy your application to a server, the `npm install` command reads the package manifest and retrieves a copy of the package.

__ 3. Examine the Express package dependencies.

__ a. List the contents of the `node_modules` directory in the Express module.

```
$ ls node_modules
accepts              escape-html        mime              safer-buffer
array-flatten        etag               mime-db           send
body-parser          express            mime-types        serve-static
bytes                finalhandler       ms                setprototypeof
content-disposition  forwarded          negotiator        statuses
content-type         fresh              on-finished       toidentifier
cookie               http-errors        parseurl          type-is
cookie-signature     iconv-lite         path-to-regexp    unpipe
debug                inherits           proxy-addr        utils-merge
depd                 ipaddr.js          qs                vary
destroy              media-typer        range-parser
ee-first             merge-descriptors  raw-body
encodeurl            methods            safe-buffer
```

Information

The `npm install` command downloads and saves the Node scripts that make up the Express package. In addition, the utility resolves any dependencies for the Express package.

2.3. Refactor the application code into an Express node application

The Express web application framework is one of the most popular building blocks for web applications. The third-party module implements an “app” class that you map to a web resource path.

In this section, use the Express web application framework to build a web application that handles the web service requests.

__ 1. Open `server.js` in a text editor.

__ 2. Create an Express app route for the context root `'/'`.

__ b. Assign the express package to a variable named express.

__ c. Create an instance of the Express object in a variable named app.

```
var express = require('express');
var app = express();
```

__ d. Define a route that maps an HTTP GET request to the path `'/api/today'`.

```
app.get('/api/today', function(req, res) {

});
```

__ e. Move the code to print the day of the week into the route’s callback function.

```
app.get('/api/today', function(req, res) {
  var body = "The day of the week is "+ today()+ ".";
  res.type('text/plain');
  res.set('Content-Length', Buffer.byteLength(body));
  res.status(200).send(body);
});
```

Information

The Express module provides convenience methods for the response object. The `res.type()` call sets the `'Content-Type'` property in the HTTP response header. The `res.set()` call sets any arbitrary HTTP response header field. The `res.status()` call sets the HTTP status code.

__ 3. Delete the `http.createServer()` code.

__ 4. Create an instance of the Express server.

__ a. At the beginning of the `server.js` script, define a variable that is named **`port`** with a value of **`3000`**.

```
var port = 3000;
```

__ b. At the end of the `server.js` script, call the `app.listen` function on the application port.

```
app.listen(port, function() {
  console.log('Listening on port %s.', port);
});
```

__ c. Delete the `server.listen()` code.

__ 5. Verify that the code matches the solution.

```
var port = 3000;

var today = require('./today');
var express = require('express');
var app = express();

app.get('/api/today', function(req, res) {
  var body = "The day of the week is "+ today()+ ".";
  res.type('text/plain');
  res.set('Content-Length', Buffer.byteLength(body));
  res.status(200).send(body);
});

app.listen(port, function() {
  console.log('Listening on port %s.', port);
});
```

__ 6. Save and close `server.js`.

__ 7. Test the `status-app` Node application.

__ a. In the terminal (Mac OS X, Linux) or command prompt (Microsoft Windows), run the package manifest start script.

__ b. Review the result in the console.

```
$ npm start

> status-app@1.0.1 start /home/localuser/projects/status-app
> node server.js

Listening on port 3000.
```

__ c. Open a web browser to http://localhost:3000/api/today.

__ d. Confirm that the status message appears in the web browser.

__ e. In the terminal (Mac OS X, Linux) or command prompt (Microsoft Windows), press Ctrl+C to exit Node.js.

2.4. Review a remote web service

Interaction services rely on data sources to build a response to the client. When you develop an API operation in the interaction services layer, you call on external databases and remote services in your business logic.

The National Weather Service provides weather observations from major airports in the United States. You can retrieve the current weather conditions from a web service on the `weather.gov` website.

In this section, review the format of the XML data from the web service.

__ 1. Review the current weather observation at San Francisco International Airport.

__ a. In a web browser, open: http://weather.gov/xml/current_obs/KSFO.xml

__ b. Select the **View Page Source** command to examine the contents of the page.

Note

You can view the page source by right-clicking some blank area in the web page, and then click **View Page Source**.

__ c. Examine the page source.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="latest_ob.xsl" type="text/xsl"?>

<current_observation version="1.0"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation=
     "http://www.weather.gov/view/current_observation.xsd">

  ...
  <suggested_pickup>15 minutes after the hour
  </suggested_pickup>
  <suggested_pickup_period>60</suggested_pickup_period>
  <location>
    San Francisco, San Francisco International Airport, CA
  </location>
  <station_id>KSFO</station_id>
  <latitude>37.61961</latitude>
  <longitude>-122.36558</longitude>
  <observation_time>Last Updated on Apr 24 2017, 1:56 pm PST
  </observation_time>
  <observation_time_rfc822>Mon, 24 Apr 2017 13:56:00 -0700
  </observation_time_rfc822>
  <weather>Mostly Cloudy</weather>
  <temperature_string>61.0 F (16.1 C)</temperature_string>
  <temp_f>61.0</temp_f>
  <temp_c>16.1</temp_c>
  ...
</current_observation>
```

Information

The International Civil Aviation Organization (ICAO) airport code for San Francisco International Airport is KSFO. To view the current weather observation at another US airport, browse to: http://weather.gov/xml/current_obs/<airport_code>.xml.

Replace <*airport_code*> with a valid ICAO airport code. Examples of ICAO airport codes:

- KORD – Chicago O'Hare International Airport, Chicago, Illinois
- KLAX – Los Angeles International Airport, Los Angeles, California
- KSJC – San Jose International Airport, San Jose, California
- KSEA – Seattle-Tacoma International Airport, Seattle, Washington
- KEWR – Newark International Airport, Newark, New Jersey
- KJFK – Kennedy International Airport, New York, New York
- KLGA – La Guardia Airport, New York, New York

- KIAD – Washington-Dulles International Airport, Dulles, Virginia
- KATL – Hartsfield-Jackson Atlanta International Airport, Atlanta, Georgia

---

__ 2. Review the weather observation at another US airport.

__ 3. Close the web browser.

2.5. Call a remote web service with the request package

The Request node package provides a set of convenience methods for making HTTP requests from your Node application.

In this section, download and install the Request node package with the npm utility. Create an Express app route that calls the National Weather Service and returns the current weather observation for the specified location.

__ 1. Install the Request node package.

__ a. Install the request node package with the `--save` parameter.

```
$ npm install request --save
```

__ b. Confirm that the node installation process has no errors.

__ 2. Review the `package.json` package manifest file.

__ a. Display the contents of the `package.json` file.

```
$ cat package.json
```

__ b. Confirm that the request package appears in the dependencies section.

```
"dependencies": {
  "express": "^4.17.1",
  "request": "^2.88.0"
}
```

__ 3. Open `server.js` in a text editor.

__ 4. Import the Request package.

```
var request = require('request');
```

__ 5. Call the weather service web service and retrieve the current conditions at San Francisco International Airport in a route named `'/api/weather'`.

__ a. Create an Express app route for `'/api/weather'`.

```
app.get('/api/weather', function(req, res) {

});
```

__ b. Store the host name, relative path, and request header parameters for the San Francisco International Airport weather service.

```
var options = {
  method: 'GET',
  uri: 'http://weather.gov/xml/current_obs/KSFO.xml',
  headers: {
        'User-agent': 'weatherRequest/1.0'
  }
};
```

__ c. Define a callback handler that displays the result from the weather service call to the `/api/weather` response message.

```
var callback = function(error, response, body) {
  res.type('text/plain');
  res.status(response.statusCode).send(body);
};
```

__ d. Call to the weather service with the options and callback handler that you defined.

```
request(options, callback);
```

__ 6. Verify that the code matches the solution.

```
var port = 3000;

var today = require('./today');
var request = require('request');
var express = require('express');
var app = express();

app.get('/api/today', function(req, res) {
  var body = "The day of the week is "+ today()+ ".";
  res.type('text/plain');
  res.set('Content-Length', Buffer.byteLength(body));
  res.status(200).send(body);
});

app.get('/api/weather', function(req, res) {
  var options = {
    method: 'GET',
    uri: 'http://weather.gov/xml/current_obs/KSFO.xml',
    headers: {
      'User-agent': 'weatherRequest/1.0'
    }
  };
  var callback = function(error, response, body) {
    res.type('text/plain');
    res.status(response.statusCode).send(body);
  };
  request(options, callback);
});

app.listen(port, function() {
  console.log('Listening on port %s.', port);
});
```

__ 7. Save and close `server.js`.

__ 8. Test the `status-app` Node application.

__ a. In the terminal (Mac OS X, Linux) or command prompt (Microsoft Windows), run the package manifest start script.

__ b. Review the result in the console.

```
$ npm start

> status-app@1.0.3 start /home/localuser/projects/status-app
> node server.js

Listening on port 3000.
```

__ c. Open a web browser to http://localhost:3000/api/weather.

__ d. Confirm that the weather observation appears in the web browser.

2.6. Propagate errors to callback functions

As an asynchronous framework, Node.js makes extensive use of callback functions to return values back to the calling function.

The node.js modules in the SDK use a convention of passing the error object as the first parameter in a callback function:

```
function ( error, parameter, ... ) { ... }
```

With this convention, the calling function always knows to check the first parameter for any error messages. If the error parameter is null, then the function processes the remaining parameters from the call.

__ 1. Open `server.js` in a text editor.

__ 2. Develop an error handler routine in the remote weather service call.

__ a. Review the Express route for the `'/api/weather'` path. The callback function returns an error object as the first parameter.

```
var callback = function(error, response, body) {
  res.type('text/plain');
  res.status(response.statusCode).send(body);
};
```

__ b. Write an error handling routine that prints the error result to the `'/api/weather'` response message.

```
var callback = function(error, response, body) {
  if (error) {
    res.status(500).send(error.message);
  }
  res.type('text/plain');
  res.status(res.statusCode).send(body);
};
```

__ 3. Create an error condition to the call.

__ a. In the options object, delete the HTTP protocol from the `uri` field.

```
var options = {
    method: 'GET',
    uri: 'weather.gov/xml/current_obs/KSFO.xml',
    headers: {
      'User-agent': 'weatherRequest/1.0'
    }
  };
```

__ 4. Save and close `server.js`.

__ 5. Test the `status-app` Node application.

__ a. In the terminal (Mac OS X, Linux) or command prompt (Microsoft Windows), run the package manifest start script.

__ b. Review the result in the console.

```
$ npm start

> status-app@1.0.1 start /home/localuser/projects/status-app
> node server.js

Listening on port 3000.
```

__ c. Open a web browser to http://localhost:3000/api/weather.

__ d. Confirm that the Request function call displays the error in the web browser.

```
Invalid URI "weather.gov/xml/current_obs/KSFO.xml"
```

__ 6. Revert the changes to the `uri` field in the options object.

__ a. Open `server.js` in a text editor.

__ b. In the options object, change the `uri` field to:
`'http://weather.gov/xml/current_obs/KSFO.xml'`

__ c. Save and close `server.js`.

2.7. Parse XML response data with the xml2js package

The term “web services” covers a range of invocation styles and data formats. For example, enterprise systems use a mix of REST and SOAP web services. In this latter style, SOAP services format the response messages with XML. Beyond REST and SOAP, one older style of web service returns an XML document as a response to an HTTP operation.

For the weather web service, the HTTP GET operation on an airport location returns an XML document.

In this section, install the xml2js node package to parse the XML response from the National Weather Service into a JSON data type.

__ 1. Install the `xml2js` Node package.

__ a. Open a terminal (Mac OS X, Linux) or command prompt (Microsoft Windows).

__ b. Go to the `status-app` directory.

__ c. Run `npm install xml2js`.

```
$ npm install xml2js --save
npm WARN status-app@1.0.1 No repository field.

+ xml2js@0.4.19
added 3 packages from 48 contributors and audited 192 packages in 1.441s
found 0 vulnerabilities
```

__ 2. Review the package manifest.

__ a. Examine the contents of `package.json`.

```
$ cat package.json
{
  "name": "status-app",
  "version": "1.0.1",
  "description": "Return Node runtime status information",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "test": "node server.js"
  },
  "author": "John Doe <jdoe@example.com>",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "request": "^2.88.0",
    "xml2js": "^0.4.19"
  }
}
```

__ b. Confirm that the package manifest lists the `xml2js` package as a dependency.

__ 3. Examine the existing implementation for the weather API.

__ a. Open `server.js` in a text editor.

__ b. Examine the implementation for the `/api/weather` route.

```
app.get('/api/weather', function(req, res) {
  var options = {
    method: 'GET',
    uri: 'http://weather.gov/xml/current_obs/KSFO.xml',
    headers: {
      'User-agent': 'weatherRequest/1.0'
    }
  };
  var callback = function(error, response, body) {
    if (error) {
      res.status(500).send(error.message);
    }
    res.type('text/plain');
    res.status(response.statusCode).send(body);
  };
  request(options, callback);
});
```

The current implementation makes an HTTP request to the remote weather website, and displays the entire web service response. In the next step, use the `parseString` function to extract the current weather observation.

__ 4. Import the `parseString` function from the xml2js package.

__ a. In the beginning of the `server.js` script, assign the variable `parse` with the `parseString` function from the `xml2js` package.

```
var parse = require('xml2js').parseString;
```

__ 5. Parse the response from the call to the remote weather web service.

__ a. In the callback handler for the weather web service call, parse the current temperature in degrees Fahrenheit from the XML response.

__ b. Add a `'content-length'` header with the size of the response message body.

__ c. Print a message with the weather reading in the HTTP response message for `'/api/weather'`.

```
var callback = function(error, response, body) {
  if (error) {
    res.status(500).send(error.message);
  }
  parse(body, function(err, result) {
    var message =
      'The current temperature is ' +
      result.current_observation.temp_f[0] +
      ' degrees Fahrenheit.';
    res.type('text/plain');
    res.set('Content-Length', Buffer.byteLength(message));
    res.status(response.statusCode).send(message);
  });
};
```

__ 6. Save and close `server.js`.

__ 7. Test the `status-app` in a web browser.

__ a. Run the package manifest start script.

__ b. Open a web browser to http://localhost:3000/api/weather.

__ c. Confirm that the web application returns the weather observation in the web browser.

__ d. In the terminal (Mac OS X, Linux) or command prompt (Microsoft Windows), press Ctrl+C to exit Node.js.

2.8. Pass parameters to a web application route

The application returns the current weather reading from San Francisco International Airport. Make the weather module more general in scope by passing input parameters into the current function.

__ 1. Open `server.js` in a text editor.

__ 2. Define a parameter that is named `location` in the `'/api/weather'` route.

__ a. Locate the Express GET handler for the `'/api/weather'` route.

__ b. Change the route to `'/api/weather/:location'`.

```
app.get('/api/weather/:location', function(req, res) {
  ...
});
```

Information

The `:location` syntax represents a path parameter in the `'/api/weather'` web route. When the Express framework intercepts a call to the `'/api/weather'` web route, it saves the path after `/weather` into a field, `request.param.location`.

__ 3. Change the request options to call the remote web service with the specified airport location.

__ a. In the `options` variable, change the **`uri`** field to use the `location` parameter in place of the airport code for San Francisco International Airport.

```
app.get('/api/weather/:location', function(req, res) {
  var options = {
    method: 'GET',
    uri: 'http://weather.gov/xml/current_obs/'
      + req.params.location + '.xml',
    headers: {
      'User-agent': 'weatherRequest/1.0'
    }
  };
...
});
```

__ 4. Save and close `server.js`.

__ 5. Test the `status-app` in a web browser.

__ a. Run the package manifest start script.

__ b. Open a web browser to http://localhost:3000/api/weather/KJFK.

__ c. Confirm that the web application returns the weather observation in the web browser.

Note

Test the http://localhost:3000/api/weather/<airport_code> API endpoint with locations in other airports. Examples of ICAO airport codes:

- KORD – Chicago O'Hare International Airport, Chicago, Illinois
- KLAX – Los Angeles International Airport, Los Angeles, California
- KSJC – San Jose International Airport, San Jose, California
- KSEA – Seattle-Tacoma International Airport, Seattle, Washington
- KEWR – Newark International Airport, Newark, New Jersey
- KJFK – Kennedy International Airport, New York, New York
- KLGA – La Guardia Airport, New York, New York
- KIAD – Washington-Dulles International Airport, Dulles, Virginia
- KATL – Hartsfield-Jackson Atlanta International Airport, Atlanta, Georgia

__ 6. In the terminal (Mac OS X, Linux) or command prompt (Microsoft Windows), press Ctrl+C to exit Node.js.

End of exercise

Solution

status-app/package.json:

```
{
  "name": "status-app",
  "version": "1.0.1",
  "description": "Return Node runtime status information",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "test": "node server.js"
  },
  "author": "John Doe <jdoe@example.com>",
  "license": "ISC",
   "dependencies": {
    "express": "^4.17.1",
    "request": "^2.88.0",
    "xml2js": "^0.4.19"
  }
}
```

status-app/today.js:

```
module.exports = function() {
  var date = new Date();
  var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday', 'Saturday'];
  return days[ date.getDay() ];
}
```

status-app/server.js:

```
var port = 3000;
var parse = require('xml2js').parseString;
var today = require('./today');
var request = require('request');
var express = require('express');
var app = express();

app.get('/api/today', function(req, res) {
  var body = "The day of the week is "+ today()+ ".";
  res.type('text/plain');
  res.set('Content-Length', Buffer.byteLength(body));
  res.status(200).send(body);
});

api.get('/api/weather/:location', function(req, res) {
  var options = {
    method: 'GET',
    uri: 'http://weather.gov/xml/current_obs/'
      + req.params.location + '.xml',
    headers: {
      'User-agent': 'weatherRequest/1.0'
    }
  };
  var callback = function(error, response, body) {
    if (error) {
      res.status(500).send(error.message);
    }
    parse(body, function(err, result) {
      var message =
        'The current temperature is ' +
        result.current_observation.temp_f[0] +
        ' degrees Fahrenheit.';
      res.type('text/plain');
      res.set('Content-Length', Buffer.byteLength(message));
      res.status(response.statusCode).send(message);
    });
  };
  request(options, callback);
});

app.listen(port, function() {
  console.log('Listening on port %s.', port);
});
```

Exercise review and wrap-up

In the first part of the exercise, you updated the status application with the Express web application framework. You defined two web routes: one for the today REST service, and one for the weather REST service. In the latter service, you called a remote web service and parsed the result in the response message.

Exercise 3. Static code analysis and unit testing

Estimated time

01:30

Overview

In this exercise, you validate and test your Node application REST API implementation. You validate the application source code with the ESLint package, and develop and run a suite of unit test cases on the Node application functions with Mocha and Supertest.

What is the user story

As an API developer, you want to test your Node application so that you can validate your implementation before you deploy your REST API.

Objectives

After completing this exercise, you should be able to:

- Explain the purpose of static code analysis
- Explain the purpose of unit testing
- Perform static code analysis with ESLint
- Create and run a function test suite in Mocha
- Create and run a web application test suite in Supertest

Introduction

In the previous exercise, you developed a REST API with a node application. Before you deploy your API, validate and test the source code on your workstation.

A *linting* utility runs a static analysis of the application source code: it reviews the application logic and source code syntax without running the application. Since JavaScript is an interpreted programming language, you cannot rely on a compiler to do a static analysis on your code.

Beyond static analysis, you must test the functions against the business logic in the design for your application. For example, you must verify that the `/api/today` API operation works on a range of dates.

In this exercise, you install *ESLint*, an open source lint utility for JavaScript applications. Install and run the *ESLint* node package on your application code to check your source code for potential errors.

You build a set of unit tests at the functional and API operation level with *Mocha* and *Supertest*.

Requirements

You must complete the steps in Exercise 2 before starting this lab.

This exercise requires a workstation with internet access. You can complete this exercise on a computer with a Linux, Mac OS X, or Microsoft Windows operating system.

For a list of supported operating systems and platforms, see: https://nodejs.org/en/download/

Exercise instructions

Preface

- The authors of this exercise tested the instructions on a Linux operating system. Unless otherwise specified, the instructions should work with minor modifications on Mac OS X and Microsoft Windows operating systems.
- The command-line examples work in a UNIX-compatible shell environment, such as Linux or Mac OS X. For example, use a text editor of your choice when the instructions list the gedit editor as an example.

3.1. Update the package manifest file

Update the minor version number in the `package.json` file to differentiate your work in this exercise from other exercises.

__ 1. Go to the `status-app` directory.

__ a. Open a terminal (Mac OS X, Linux) or command prompt (Microsoft Windows).

__ b. Go to the `/projects/status-app` directory.

```
$ cd projects/status-app/
```

__ 2. Update the version number in the package manifest to 1.0.2.

__ a. Open `package.json` in a text editor.

```
$ gedit package.json
```

__ b. Update the version field to 1.0.2.

```
"version": "1.0.2",
```

__ c. Save and close `package.json`.

3.2. Update the today API operation

The `/api/today` API operation returns the day of the current week. In this section, update the `today.js` implementation to take any arbitrary date. If the application calls the today module without any parameters, use the current date.

__ 1. Update the implementation for `today.js`.

__ a. In the `/projects/status-app` directory, open `today.js` in a text editor.

```
$ cd projects/status-app
$ gedit today.js
```

__ b. Review the `today.js` implementation.

```
module.exports = function() {
  var date = new Date();
  var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday', 'Saturday'];
  return days[ date.getDay() ];
}
```

The current implementation returns the day of the week for the system date.

__ c. Add a parameter, `date`, to the `module.exports` function.

```
module.exports = function(date) {
```

__ d. Assign the date parameter to the `date` variable declaration.

__ e. Verify your changes with the solution.

```
module.exports = function(date) {
  var date = date;
  var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday', 'Saturday'];
  return days[ date.getDay() ];
}
```

__ f. Save and close `today.js`.

__ 2. Update the route for the `/api/today` API operation in `server.js`.

__ a. Open `server.js` in a text editor.

```
$ gedit server.js
```

__ b. Examine the web application route for `/api/today`.

```
app.get('/api/today', function(req, res) {
  var body = "The day of the week is "+ today()+ ".";
  res.type('text/plain');
  res.set('Content-Length', Buffer.byteLength(body));
  res.status(200).send(body);
});
```

__ c. Set the `date` variable to today's date, or the date in the query parameter.

```
var date = req.query.date = null ? new Date() : new Date(req.query.date);
```

__ d. In the `today()` function call, add the path parameter, `date`.

```
var body = "The day of the week is "+ today(date)+ "."
```

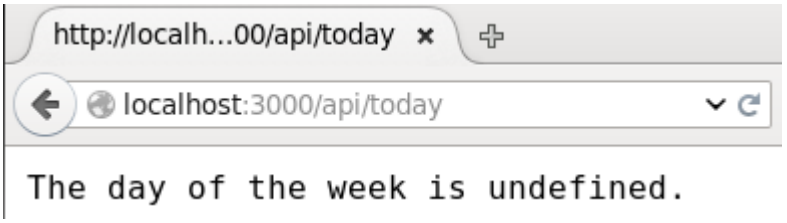
__ e. Save and close `server.js`.

__ 3. Test the implementation.

__ a. Run the status-app node application.

```
$ npm start

> status-app@1.0.2 start /home/localuser/projects/status-app
> node server.js

Listening on port 3000.
```

__ b. In a web browser, open: `http://localhost:3000/api/today`



Questions

Why is the day of the week undefined?

You expect the `/api/today` implementation to return today's date, but it returns an undefined date. You troubleshoot this issue later in the exercise.

__ c. In the terminal (Mac OS, Linux) or command prompt (Microsoft Windows), press Ctrl+C to stop the application.

3.3. Identify coding errors with ESLint static code analysis

Static code analysis identifies potential logic errors by examining the flow of your application source code. The *ESLint* package checks your application against a list of over 200 rules that are designed for browser and server (node) JavaScript code.

Although the `/api/today` API operation does not have any syntax or runtime errors, the application does not return the correct day of the week. In this section, install, configure, and run *ESLint* against your application code. Review and correct the `/api/today` operation.

__ 1. Install the ESLint Node package as a global module.

__ a. Open terminal (Mac OS, Linux) or command prompt (Microsoft Windows).

__ b. Install the ESLint module with the `-g` global option.

```
$ npm install eslint -g
```

__ c. Confirm that the package installation contains no error messages.

__ d. Verify that you can run the ESLint utility.

```
$ eslint -v
v5.16.0
```

Note

You require administrator level access to install the `eslint` module as a global package. For example, if you installed the Node.js runtime in the `/usr/local/` directory, your user account must have write permission to the `/usr/local/` directory as well.

If you cannot install `eslint` as a global package, you can install it as an application-specific package:

```
npm install eslint --save-dev
```

The `--save-dev` parameter marks `eslint` as a local development dependency.

To run the local copy of the eslint module, run the following command:

```
node node_modules/eslint/bin/eslint
```

__ 2. Create an ESLint configuration for the `status-app` Node application.

__ a. In the `status-app` directory, start the ESLint initialization routine.

```
$ eslint --init
```

__ b. Choose **To check syntax and find problems** as a starting point for the configuration.

Information

Use the up or down keys to move through the selections. Use the space bar to select or clear the runtime environment. If you choose the wrong option, you can rerun the initialization routine.

```
? How would you like to use ESLint? (Use arrow keys)
  To check syntax only
> To check syntax and find problems
  To check syntax, find problems, and enforce code style
```

__ c. Select JavaScript modules for ESLint to examine.

```
? What type of modules does your project use? (Use arrow keys)
> JavaScript modules (import/export)
  CommonJS (require/exports)
  None of these
```

__ d. Select None of these for the framework that the project uses.

```
? Which framework does your project use?
  React
  Vue.js
> None of these
```

__ e. Select Node for the option where the code runs.

```
? Where does your code run? (Press <space> to select, <a> to toggle all,
<i> to invert selection)
   Browser
   > Node
```

__ f. Select JavaScript as the configuration file format.

```
? What format do you want your config file to be in?
> JavaScript
  YAML
  JSON

Successfully created .eslintrc.js file in
/home/localuser/projects/status-app
```

__ g. Confirm that the initialization utility completes successfully.

Information

For more information about the configuration settings, see:
http://eslint.org/docs/user-guide/configuring

__ 3. Run the ESLint utility on the JavaScript files.

__ a. Run the ESLint utility and examine the result.

```
$ eslint *.js

/home/localuser/projects/status-app/server.js
   2:13  error  'require' is not defined       no-undef
   3:13  error  'require' is not defined       no-undef
   4:15  error  'require' is not defined       no-undef
   5:15  error  'require' is not defined       no-undef
   9:31  error  Unexpected constant condition  no-constant-condition
  12:29  error  'Buffer' is not defined        no-undef
  35:31  error  'Buffer' is not defined        no-undef
  43:3   error  Unexpected console statement   no-console

/home/localuser/projects/status-app/today.js
  1:1  error  'module' is not defined    no-undef
  2:7  error  'date' is already defined  no-redeclare

? 10 problems (10 errors, 0 warnings)
```

Information

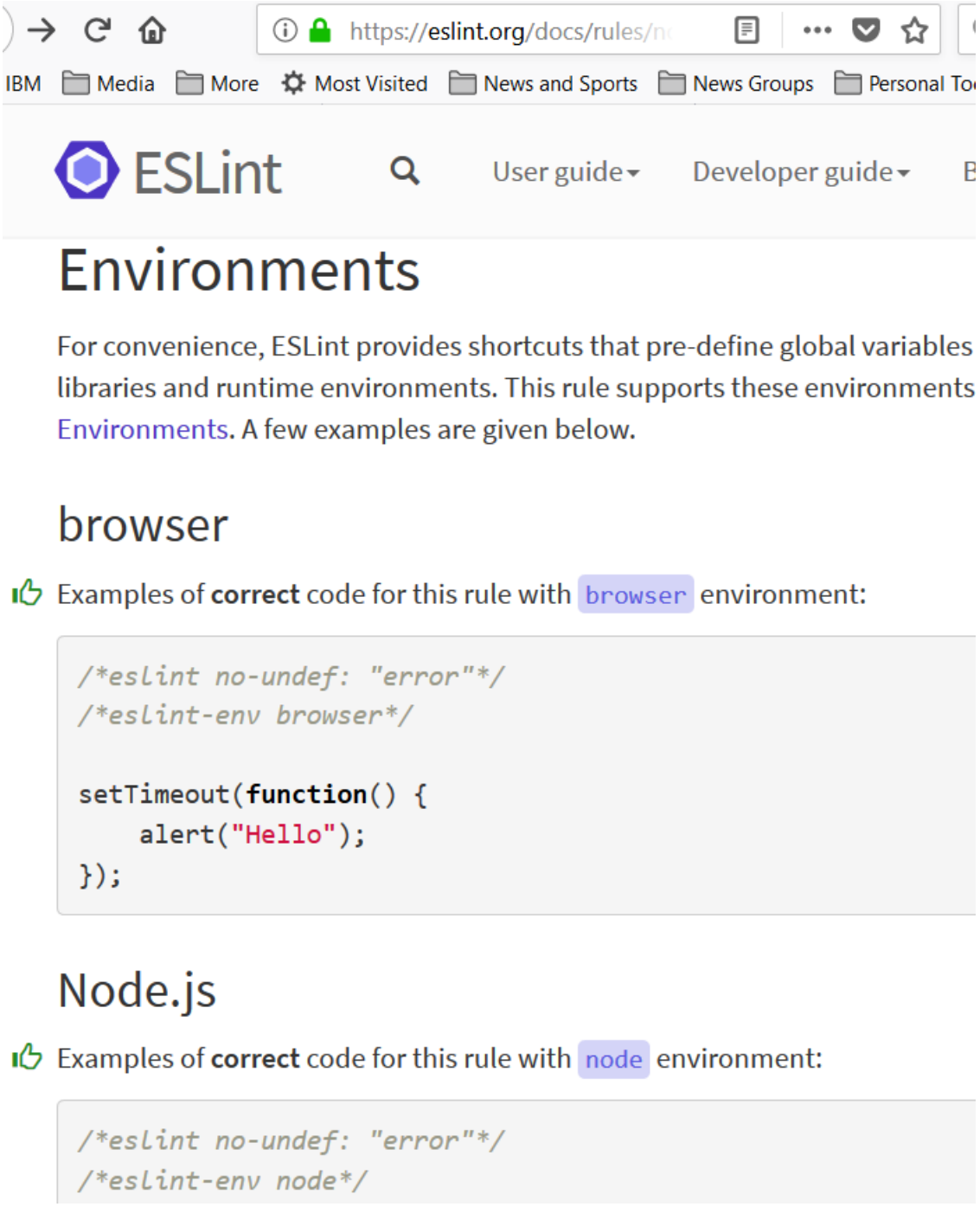
ESLint identifies the line number and row where each error or warning appears in your application. For example, in line 9, row 31 of the `server.js` file, your code has an unexpected constant condition.

For more information about the meaning of each rule, see: https://eslint.org/docs/rules/

__ 4. Review the *Disallow Undeclared Variables (no-undef)* error.

__ a. Open a web browser to: http://eslint.org/docs/rules/no-undef

__ 5. Review the Environment section under the **no-undef** rule.



__ 6. Add the eslint statements for the node environment to the files.

__ a. Open the `server.js` file with an editor. Then, add the statements at the top of the file.

```
/*eslint no-undef: "error"*/
/*eslint-env node*/
```

__ b. Save the change.

__ a. Open the `today.js` file with an editor. Then, add the statements at the top of the file.

```
/*eslint no-undef: "error"*/
/*eslint-env node*/
```

__ b. Save the change

__ 7. Rerun the ESLint utility on the JavaScript files.

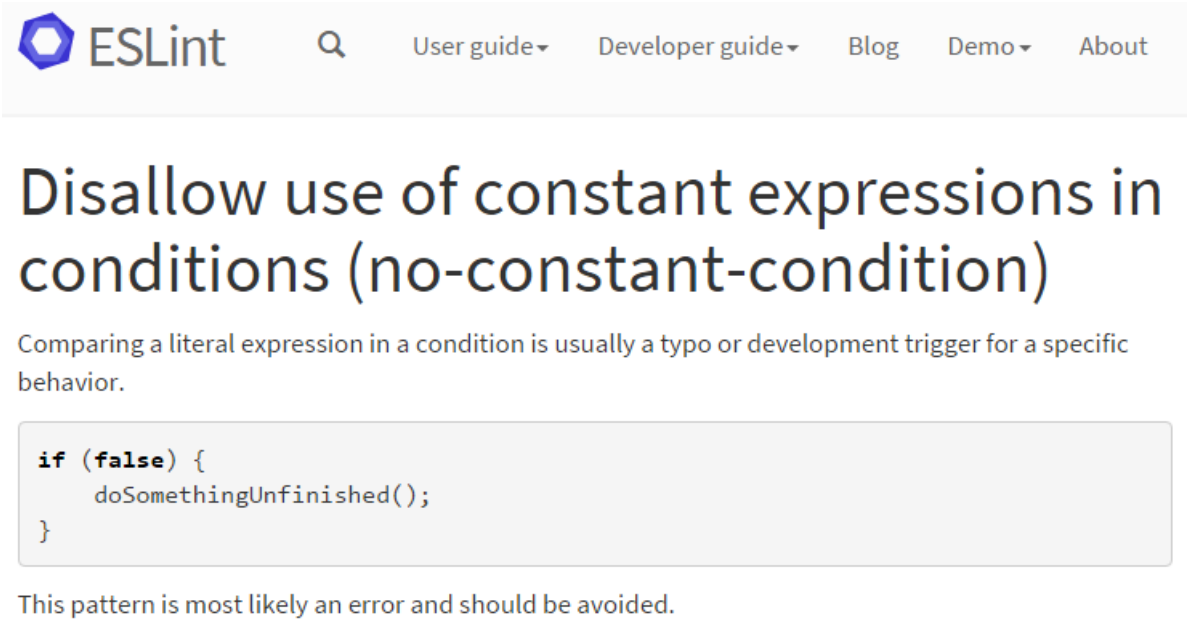
__ a. Run the ESLint utility and examine the result.

```
$ eslint *.js

/home/localuser/projects/status-app/server.js
  11:31 error  Unexpected constant condition  no-constant-condition
  45:3   error  Unexpected console statement   no-console

/home/localuser/projects/status-app/today.js
  4:7  error  'date' is already defined  no-redeclare

  x 3 problems (3 errors, 0 warnings)
```

__ 8. Review the *Unexpected constant condition* error.

__ a. Open a web browser to: http://eslint.org/docs/rules/

__ b. Review the **no-constant-condition** rule.



Note

In a conditional statement, such as an `if` statement, the expression evaluates to either `true` or `false`. In a **constant expression**, the expression always evaluates to one result. That is, you wrote an "*if*" block that always runs, or one that never runs.

Although the code is syntactically correct, it does not make sense to add the statement into your application.

__ 9. Examine and correct the error in `server.js`.

__ a. Open `server.js` in a text editor.

__ b. Examine the line with the constant condition error.

```
app.get('/api/today', function(req, res) {
  var date = req.query.date = null ? new Date() : new
Date(req.query.date);
  var body = "The day of the week is "+ today()+ ".";
  res.type('text/plain');
  res.set('Content-Length', Buffer.byteLength(body));
  res.status(200).send(body);
});
```

Note

The `date` variable uses an expression to determine whether the caller set the date query parameter. For example, the API call `/api/today?date='30-May-2019'` assigns the `req.query.date` property with the value of `30-May-2019`.

The purpose of the `req.query.date = null` code is to check whether the user passed a query parameter that is named `date` in the API operation. If the user did not pass a `date` parameter, `req.query.date` is set to `null`.

However, a single equal sign (=) is an *assignment* operator. To check for *equivalence*, use a double equals sign (==).

__ c. Change the expression to check whether `req.query.date` is null.

```
var date = req.query.date == null ? new Date() : new
Date(req.query.date);
```

__ d. Save and close `server.js`.

__ 10. Check the `server.js` script with the ESLint utility.

__ a. Run ESLint on `server.js`.

```
$ eslint server.js

/home/localuser/projects/status-app/server.js
  45:3  error  Unexpected console statement  no-console

? 1 problem (1 error, 0 warnings)
```

Note

The JavaScript defines the triple equal sign as a **type-safe equality operator**: given `x === y`, the runtime engine checks whether `x` and `y` are the same variable type before it compares the values of `x` and `y`.

When you want to compare two variables for equivalence, it is suggested that you use the === operator instead of the == operator.

The situation is more complicated when you check a variable against `null`. With the equality operator, the expression `req.query.date == null` is true if `req.query.date` is either `null` or undefined. Since you want to use the current date if the `req.query.date` property does not contain a valid date, override the ESLint error and leave the code as is.

For more information, see the `no-eq-null` and `eqeqeq` rules on: http://eslint.org/docs/rules/

__ 11. Disable the `no-console` rule for the entire `status-app` application.

Note

The `no-console` rule flags the `console` object as an error in the JavaScript code. The rationale for this rule is that console logs should not appear in the user's web browser.

However, users cannot see the output from the console log object: it is saved in the server's system log. Therefore, it makes sense to disable the `no-console` rule when you check a Node application.

For more information about the no-console rule, see: http://eslint.org/docs/rules/no-console

__ a. Open the `.eslintrc.js` configuration file in a text editor.

__ b. Add a rule that is named `no-console` and set the rule to `off`.

```
"no-console": "off"
```

__ c. Save and close `.eslintrc.js`.

__ 12. Run the ESLint utility.

__ a. Run ESLint on all JavaScript files in the `status-app` directory.

```
$ eslint *.js

/home/localuser/projects/status-app/today.js
  2:7  error  'date' is already defined  no-redeclare

x 1 problem (1 error, 0 warnings)
```

__ b. Confirm that the `server.js` script contains no errors.

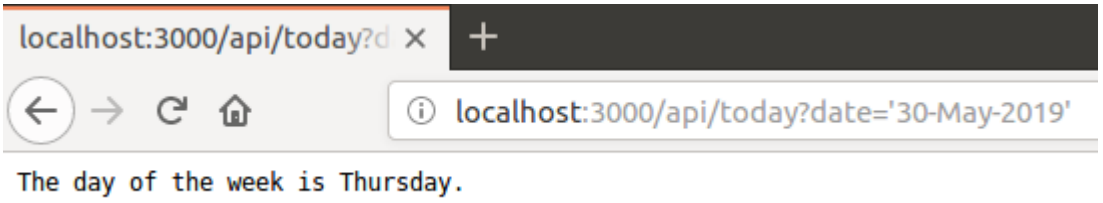
__ c. Review and correct the `no-redeclare` error in the `today.js` script.

__ 13. Test the implementation.

__ a. Run the `status-app` node application.

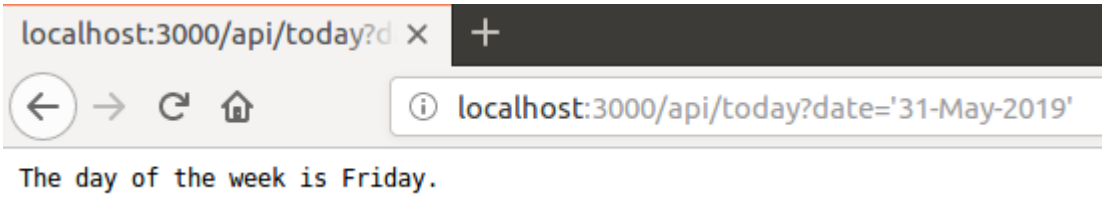
```
$ npm start

> status-app@1.0.2 start /home/localuser/projects/status-app
> node server.js

Listening on port 3000.
```

__ b. In a web browser, open http://localhost:3000/api/today?date='30-May-2019'.



Confirm that the day of the week on May 30, 2019 is Thursday.

__ c. Open http://localhost:3000/api/today?date='31-May-2019'.



Confirm that the day of the week on May 31, 2019 is Friday.

__ d. Open http://localhost:3000/api/today.

__ e. Confirm that the `/api/today` operation returns the current day of the week.

__ 14. Stop the `status-app` application.

__ a. In the terminal (Mac OS, Linux) or command prompt (Microsoft Windows), press Ctrl+C to exit the application.

Information

In static code analysis, the utility examines the structure and logic of your application against a set of suggested practices and common mistakes. The warnings and errors that are flagged by ESLint identify possible logic errors – it does not necessarily mean that they are actual errors in your program.

ESLint is a useful tool in detecting potential errors. However, you must verify the correctness of your application with unit testing.

3.4. Define unit test cases with the Mocha framework

The purpose of *unit testing* is to verify whether a function in your application runs correctly. The *Mocha* package is a popular unit testing framework for Node applications. In this section, you develop a set of unit test cases that verify the logic for the `/api/today` API operation.

__ 1. Install the `mocha` node package.

__ a. Open a terminal (Mac OS, Linux) or command prompt (Microsoft Windows).

__ b. Change the directory to the `status-app` directory.

```
$ cd projects/status-app
```

__ c. Install the `mocha` package as a development dependency.

```
$ npm install mocha --save-dev
```

__ d. Confirm that the `mocha` module is successfully installed.

Information

One of the main components of a unit testing framework is an *assertion* library: when your assertion is correct, the unit test should complete successfully. For example, you can state an assertion that May 30, 2019 is Thursday.

The *Mocha* test framework does not include its own *assertion* library. If a unit test fails, it expects that the test throws a JavaScript error.

Before you write your test cases, import a third-party *assertion* library. The Chai package provides an assertion library that is named *expects*.

__ 2. Install the `chai` node package.

__ a. Install the `chai` node package as a development dependency.

```
$ npm install chai --save-dev
```

__ b. Confirm that the `chai` module is successfully installed.

__ 3. Review the package manifest file.

__ a. Open `package.json` in a text editor.

__ b. Examine the `devDependencies` section.

```
"devDependencies": {
    "chai": "^4.2.0",
    "eslint": "^5.16.0",
    "mocha": "^6.1.4"
  }
```

The *development dependencies* are node packages that the status-app uses during development and testing. When you set the node environment to *production*, the `npm install` command does not install these packages.

__ c. Close `package.json`.

__ 4. Create a script, `today-test.js`, to test the today module.

__ a. Create a test directory to store the unit test cases.

```
$ mkdir test
$ cd test
```

__ b. Create a script that is named `test-today.js` in a text editor.

```
$ gedit today-test.js
```

__ c. Import the `expect` function from the `chai` package.

__ d. Import the `today` function from the `today.js` script.

```
/*eslint no-undef: "error"*/
/*eslint-env node*/
var mocha = require('mocha');
var describe = mocha.describe;
var it = mocha.it;
var expect = require('chai').expect;
var today = require('../today');
```

__ e. Define a test case with a name of `'Today'`.

```
describe('Today', function() {

});
```

__ 5. Write a test case to check the current day of the week.

__ a. In the `describe('Today')` function, define a test case that is named: `'now'`

__ b. In the description, enter: `'returns the current day of the week'`

```
describe('Today', function () {
  describe('now', function() {
    it('should return the current day of the week', function() {

    });
  });
});
```

__ c. In the body of the **now** test case, write a method that returns the current day of the week.

```
// test setup
var date = new Date();
var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday'];
var dayOfWeek = days[ date.getDay() ];
```

Note

The `test setup` part of the unit test calculates the current day of the week. In the next part, compare this value with the result of the `today(date)` function call. Both values should return the same day of the week.

__ d. Add an *expect* statement to compare the output from `today()` against the current day of the week.

```
expect(today(date)).to.equal(dayOfWeek);
```

__ e. Compare your test case against the solution:

```
describe('now', function() {
    it('should return the current day of the week', function() {
      // test setup
      var date = new Date();
      var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday'];

      var dayOfWeek = days[ date.getDay() ];
      expect(today(date)).to.equal(dayOfWeek);
    });
  });
```

__ f. Save the file.

Information

In the *mocha* framework, each test has the following pattern:

```
describe('a name for the unit test', function() {
  it('should return a value x given the input y', function() {
    expected(x).to.equal(y);
  });
});
```

In this example, the `describe` function stores the name of the function or the application that you want to test. The `it` function stores an assertion: a statement that is true if the application works properly.

The code within the anonymous function checks whether the assertion holds true. The `expected(...).to.equal(...)` function takes two parameters. If the two parameters are equal, the function completes without issue. If the two parameters do not match, the `expected(...)` function throws a runtime error.

The *mocha* framework detects the runtime error, and marks the test case as failed.

For more information about the *mocha* framework, see: http://mochajs.org/

For more information about the `expects` function from the *chai* assertion library, see: http://chaijs.com

---

__ 6. Write a test case to check whether the date `'01-January-2018'` matches `Monday`.

__ a. After the first test case, define another test case that is named `'01-January-2018'`.

__ b. Add the description: `'should return Monday for 01-January-2018'`

__ c. Add an `expect` statement for the date `01-January-2018` and the value of `Monday`.

__ d. Compare your test case against the solution.

```
describe('01-January-2018', function() {
  it('should return Monday for 01-January-2018', function() {
    expect(today(new Date('01-January-2018'))).to.equal('Monday');
  });
});
```

__ e. Save the file.

__ 7. Write a test case to check whether the date `'02-January-2018'` matches `Tuesday`.

__ a. After the second test case, define another test case that is named: `'02-January-2018'`

__ b. Add the description: `'should return Saturday for 02-January-2018'`

__ c. Add an `expect` statement for the date `02-January-2018` and the value of `Tuesday`.

__ d. Compare your test case against the solution.

```
describe('02-January-2018', function() {
  it('should return Tuesday for 02-January-2018', function() {
    expect(today(new Date('02-January-2018'))).to.equal('Tuesday');
  });
});
```

__ e. Save the file.

__ 8. Save and close `today-test.js`.

Hint

You can run the ESLint utility against the `today-test.js` test script to check for potential programming errors.

```
eslint test/today-test.js
```

__ 9. Update the test script in the `package.json` file.

__ a. Change the directory to the main `status-app` directory.

__ b. Open `package.json` in a text editor.

```
$ cd ..
$ gedit package.json
```

__ c. Change the test script to run `mocha` on the `test` directory.

```
"scripts": {
  "test": "mocha test",
  "start": "node server.js"
},
```

__ d. Save and close `package.json`.

Information

The `mocha test` command starts the mocha test harness with the JavaScript test scripts in the `test` directory.

__ 10. Run the test cases for the `status-app` application.

__ a. Run the npm test command.

```
$ npm test

> status-app@1.0.2 test /home/student/projects/status-app
> mocha test

  Today
    now
      v should return the current day of the week
    01 January 2018
      v should return Monday for 01-January-2018
    02 January 2018
      v should return Tuesday for 02-January-2018

  3 passing (17ms)
```

Information

The *mocha* command starts a test harness for a series of test cases that are defined by the `describe` function. The `it` function declares an assertion about the application. For example, the `now` test case checks whether the `today` module returns the current day of the week.

Each test case passes when the application does not throw an error during its execution. The `expects` function compares the today function result against the asserted value. If the two values do not match, `expects` throws an error. The *mocha* framework detects the error and marks the test case as failed.

3.5. Define REST API test cases with Supertest

In the previous section, you compared the result from a JavaScript function call against an expected value. To test a REST API, your test script must call an HTTP endpoint and examine the response message.

The *Supertest* node package simplifies the task of making an HTTP request and comparing the result against an asserted value. It uses the following structure to make an HTTP request, and intercept the response with a callback handler:

```
request
.get('/api/path')
.expect(200)
.end(function(err, res) {
  if (err) {
    return done(err);
  }
  done();
}
```

This example makes an HTTP GET request to the `/api/path` route on the server. It makes an assertion that the HTTP status code is `200`. It defines a callback handler that throws an error when the call fails. If the call succeeds, the handler calls `done()` to signify that the asynchronous operation completed successfully.

In this section, define a *mocha* test script that makes several HTTP requests to the `/api/today` operation. Confirm that the operation returns the correct day of the week for the specified dates.

__ 1. Install the `supertest` node package.

__ a. Open a terminal (Mac OS, Linux) or command prompt.

__ b. Install the `supertest` node package as a development dependency.

```
$ npm install supertest --save-dev
```

__ c. Confirm that the `chai` module is successfully installed.

__ 2. Create a test script that is named `api-today-test.js` in the `test` directory.

__ d. Change directory to the `project/status-app/test` directory.

```
$ cd project/status-app/test
```

__ e. Create the `api-today-test.js` script in a text editor.

```
$ gedit api-today-test.js
```

__ 3. Import the test frameworks and the `server.js` script in `api-today-test.js`.

__ a. In the `api-today-test.js` script, import the `server.js` script.

```
var server = require('../server');
```

Important

When you import the `server.js` script, the node runtime environment runs the code within the script. You must add this step to start the REST API server for `/api/today`.

__ b. Import the `chai` and `supertest` frameworks.

```
var mocha = require('mocha');
var describe = mocha.describe;
var it = mocha.it;
var expect = require('chai').expect;
var supertest = require('supertest');
```

__ c. Declare the host and port name for the `/api/today` server.

```
var request = supertest.agent('http://localhost:3000');
```

__ 4. Define a unit test that checks the day of the week against a set of dates.

__ a. Define a test case that is named `'/api/today/'`.

__ b. Within the `'/api/today/'` test case, create a test case that is named `'GET with date query parameter'`.

```
describe('/api/today/', function() {
  describe('GET with #date query parameter', function() {

  });
});
```

__ c. In the test case, define a variable named `tests` with an array of known dates and days of week. For example, January 1, 2016 is on Friday.

```
var tests = [
  {arg: '01-January-2018', expected: 'Monday'},
  {arg: '02-January-2018', expected: 'Tuesday'},
  {arg: '03-January-2018', expected: 'Wednesday'},
  {arg: '04-January-2018', expected: 'Thursday'},
  {arg: '05-January-2018', expected: 'Friday'},
  {arg: '06-January-2018', expected: 'Saturday'},
  {arg: '07-January-2018', expected: 'Sunday'}
];
```

__ d. Iterate through each of the arguments and expected result.

```
tests.forEach(function(test) {

});
```

__ e. In the `forEach` loop, make an HTTP GET request with the `test.arg` arguments.

```
tests.forEach(function(test) {
  it('returns ' + test.expected + ' for ' + test.arg, function(done) {
    request
    .get('/api/today')
    .query({date: test.arg})
    .expect(200)
    .end(function(err,res) {
      done();
    });
  });
});
```

Information

The *Supertest* module consists of two parts: the *super-agent* module, and an *expect* assertion library.

Super-agent simplifies the request object in Node. You use a chained set of function calls to construct an HTTP request. In this example, you create an HTTP GET request on the `'/api/today'` route. You add a query parameter of `date` with the value from the `test.arg` property.

The `expect(200)` operation call checks the HTTP status code. If the status is not `200 OK`, the *Supertest* object throws a runtime error. The *mocha* framework marks this test case as failed when it catches the runtime error.

The `end(function(err,res) {...})` function closes the HTTP request. You define a callback handler that processes either an error object or the response message.

For more information, see: https://github.com/visionmedia/supertest

__ 5. Test the `api-today-test.js` test script.

__ a. Run `npm test` to run all the scripts in the `test` directory.

```
$ npm test

> status-app@1.0.2 test /home/localuser/projects/status-app
> mocha test

Listening on port 3000.
  /api/today
    GET with #date query parameter
      v returns Monday for 01-January-2018 (52ms)
      v returns Tuesday for 02-January-2018
      v returns Wednesday for 03-January-2018
      v returns Thursday for 04-January-2018
      v returns Friday for 05-January-2018
      v returns Saturday for 06-January-2018
      v returns Sunday for 07-January-2018

  Today
    now
      v should return the current day of the week
    01-January-2018
      v should return Friday for 01-January-2018
    02-January-2018
      v should return Saturday for 02-January-2018


  10 passing (115ms)
```

Information

The `api-today-test.js` script iterates through an array of seven test cases:

- A GET request to `/api/today?date='01-January-2018'` should return `'Monday'`
- A GET request to `/api/today?date='02-January-2018'` should return `'Tuesday'`
- A GET request to `/api/today?date='03-January-2018'` should return `'Wednesday'`
- A GET request to `/api/today?date='04-January-2018'` should return `'Thursday'`
- A GET request to `/api/today?date='05-January-2018'` should return `'Friday'`
- A GET request to `/api/today?date='06-January-2018'` should return `'Saturday'`
- A GET request to `/api/today?date='07-January-2018'` should return `'Sunday'`

However, the test case checks only whether the status code from the response message is 200.

In the next step, complete the callback handler implementation and check whether the day of the week matches the expected value.

__ 6. In the GET request callback handler, check whether the response message matches `'The day of the week is ' + test.expected`, where `test.expected` is the expected value from the test data.

__ a. Open `test/api-today-test.js` in a text editor.

__ b. In the `request.get` callback handler, return an `error` to the `done()` function if the call does not complete successfully.

__ c. Otherwise, compare the response with the expected day of the week.

```
.end(function(err,res) {
  if (err) {
    return done(err);
  }
  expect(res.text).to.equal(
    'The day of the week is ' + test.expected + '.');
  done();
});
```

__ 7. Compare your code against the solution:

```
var server = require('../server');
var mocha = require('mocha');
var describe = mocha.describe;
var it = mocha.it;
var expect = require('chai').expect;
var supertest = require('supertest');
var request = supertest.agent('http://localhost:3000');

describe('/api/today', function() {
  describe('GET with #date query parameter', function() {
   var tests = [
         {arg: '01-January-2018', expected: 'Monday'},
         {arg: '02-January-2018', expected: 'Tuesday'},
         {arg: '03-January-2018', expected: 'Wedday'},
         {arg: '04-January-2018', expected: 'Thursday'},
         {arg: '05-January-2018', expected: 'Friday'},
         {arg: '06-January-2018', expected: 'Saturday'},
         {arg: '07-January-2018', expected: 'Sunday'}
    ];

    tests.forEach(function(test) {
      it('returns ' + test.expected + ' for ' + test.arg, function(done) {
        request
        .get('/api/today')
        .query({date: test.arg})
        .expect(200)
        .end(function(err,res) {
          if (err) {
            return done(err);
          }
          expect(res.text).to.equal(
            'The day of the week is ' + test.expected + '.');
          done();

        });
      });
    });

  });
});
```

__ 8. Save and close `api-today-test.js`.

__ 9. Test the `api-today-test.js` script.

__ a. Run `npm test` from the terminal or command prompt.

```
$ npm test

> status-app@1.0.2 test /home/localuser/projects/status-app
> mocha test

Listening on port 3000.
  /api/today
    GET with #date query parameter
      v returns Monday for 01-January-2018 (43ms)
      v returns Tuesday for 02-January-2018
      v returns Wednesday for 03-January-2018
      v returns Thursday for 04-January-2018
      v returns Friday for 05-January-2018
      v returns Saturday for 06-January-2018
      v returns Sunday for 07-January-2018

  Today
    now
      v should return the current day of the week
    01-January-2018
      v should return Monday for 01-January-2018
    02-January-2018
      v should return Tuesday for 02-January-2018

  10 passing (90ms)
```

__ b. Confirm that the tests for the `/api/today` API operation and the `today` node module complete successfully.

End of exercise

Solution

status-app/package.json:

```
{
  "name": "status-app",
  "version": "1.0.2",
  "description": "Return Node runtime status information",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "test": "mocha test"
  },
  "author": "John Doe <jdoe@example.com>",
  "license": "ISC",
  "dependencies": {
   "express": "^4.17.1",
    "request": "^2.88.0",
    "xml2js": "^0.4.19"
  },
  "devDependencies": {
    "chai": "^4.2.0",
    "eslint": "^5.16.0",
    "mocha": "^6.1.4",
    "supertest": "^4.0.2"
  }
}
```

status-app/today.js:

```
/*eslint no-undef: "error"*/
/*eslint-env node*/
module.exports = function(date) {
  var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday', 'Saturday'];
  return days[ date.getDay() ];
}
```

status-app/server.js:

```
/*eslint no-undef: "error"*/
/*eslint-env node*/
var port = 3000;

var today = require('./today');
var parse = require('xml2js').parseString;
var request = require('request');
var express = require('express');
var app = express();

app.get('/api/today', function(req, res) {
  var date = req.query.date == null ? new Date() : new Date (req.query.date);
  var body = "The day of the week is " + today(date) + ".";
  res.type('text/plain');
  res.set('Content-Length', Buffer.byteLength(body));
  res.status(200).send(body);
});

app.get('/api/weather/:location', function(req, res) {
  var options = {
    method: 'GET',
    uri: 'http://weather.gov/xml/current_obs/'
      + req.params.location + '.xml',
    headers: {
      'User-agent': 'weatherRequest/1.0'
    }
  };
  var callback = function(error, response, body) {
    if (error) {
      res.status(500).send(error.message);
      return;
    }
    parse(body, function(err, result) {
      var message =
        'The current temperature is ' +
        result.current_observation.temp_f[0] +
        ' degrees Fahrenheit.';
      res.type('text/plain');
      res.set('Content-Length', Buffer.byteLength(message));
      res.status(response.statusCode).send(message);
    });
  };
  request(options, callback);
});

app.listen(port, function() {
  console.log('Listening on port %s.', port);
```

```
});
```

/status-app/test/today-test.js

```
/*eslint no-undef: "error"*/
/*eslint-env node*/
var mocha = require('mocha');
var describe = mocha.describe;
var it = mocha.it;
var expect = require('chai').expect;
var today = require('../today');

describe('Today', function () {
  describe('now', function() {
    it('should return the current day of the week', function() {

      // test setup
      var date = new Date();
      var days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday"];

      var dayOfWeek = days[ date.getDay() ];
      expect(today(date)).to.equal(dayOfWeek);
    });
  });

  describe('01-January-2018', function() {
    it('should return Friday for 01-January-2018', function() {
      expect(today(new Date('01-January-2018'))).to.equal('Monday');
    });
  });

  describe('02-January-2018', function() {
    it('should return Friday for 02-January-2018', function() {
      expect(today(new Date('01-January-2018'))).to.equal('Tuesday');
    });
  });
});
```

/status-app/test/api-today-test.js

```
var server = require('../server');
var mocha = require('mocha');
var describe = mocha.describe;
var it = mocha.it;
var expect = require('chai').expect;
var supertest = require('supertest');

var request = supertest.agent('http://localhost:3000');

describe('/api/today', function() {
  describe('GET with #date query parameter', function() {
      var tests = [
            {arg: '01-January-2018', expected: 'Monday'},
            {arg: '02-January-2018', expected: 'Tuesday'},
            {arg: '03-January-2018', expected: 'Wednesday'},
            {arg: '04-January-2018', expected: 'Thursday'},
            {arg: '05-January-2018', expected: 'Friday'},
            {arg: '06-January-2018', expected: 'Saturday'},
            {arg: '07-January-2018', expected: 'Sunday'}
    ];

    tests.forEach(function(test) {
      it('returns ' + test.expected + ' for ' + test.arg, function(done) {
        request
        .get('/api/today')
        .query({date: test.arg})
        .expect(200)
        .end(function(err,res) {
          if (err) {
            return done(err);
          }
          expect(res.text).to.equal(
            'The day of the week is ' + test.expected + '.');
          done();
        });
      });
    });
  });
});
```

Exercise review and wrap-up

The first part of the exercise you ran static code analysis on the status application. You identified and corrected coding logic issues and syntax errors in your code. In the second part of the application, you developed unit tests to verify the today function and the today API operation.

Exercise 4. Debugging and building Node applications

Estimated time

01:00

Overview

In this exercise, you work with various Node.js debug utilities on your own workstation. You work with the command-line and graphical debug utilities. You also create scripts in the package.json file that npm uses to build and run node applications.

What is the user story

As an API developer, you want to debug node applications so that you can troubleshoot application and deployment issues.

Objectives

After completing this exercise, you should be able to:

- Use the standard node debug utility of Node
- Enable Node Inspector
- Work with the Node Inspector graphical debug tool
- Use package lock to set node module versions
- Use script objects and npm to build node applications

paIntroduction

In this exercise, you install the Node Inspector graphical debugger, and you explore how to use the Node Inspector with the Chrome browser to debug node applications.

You learn how the `npm shrinkwrap` command can be used to lock the version numbers of the package dependencies for cloning and packaging applications.

Finally, you learn how npm commands can be used to create a build script. Then, you use the npm run command to run the scripts.

Requirements

This exercise requires a workstation with internet access. You can complete this exercise on a computer with a Linux, Mac OS, or Microsoft Windows operating system.

To complete this exercise, you must install the node-inspector command-line interface on your workstation.

Exercise instructions

Preface

- The authors of this exercise tested the instructions on a Linux operating system. Unless otherwise specified, the instructions should work with minor modifications on Mac OS X and Microsoft Windows operating systems.
- The command-line examples work in a UNIX-compatible shell environment, such as Linux or Mac OS X. For example, use a text editor of your choice when the instructions list the gedit editor as an example.

4.1. Update the package manifest file

Update the minor version number in the package.json file to differentiate your work in this exercise from other exercises.

__ 1. Go to the status-app directory.

__ a. Open a terminal (Mac OS X, Linux) or command prompt (Microsoft Windows).

__ b. Go to the /projects/status-app directory.

```
$ cd projects/status-app/
```

__ 2. Update the version number in the package manifest to 1.0.3.

__ a. Open package.json in a text editor.

```
$ gedit package.json
```

__ b. Update the version field to 1.0.3.

```
  "version": "1.0.3",
```

__ c. Save and close package.json.

4.2. Use the standard node debug utility

Node.js includes a debugging utility that is accessible from the command line.

To use the standard debugger, start node.js with the `debug` parameter followed by the path to the script to debug.

__ 1. Go to the `status-app` directory.

__ a. Open a terminal (Mac OS X, Linux) or command prompt (Microsoft Windows).

__ b. Go to the `/projects/status-app` directory.
```
$ cd projects/status-app/
```

__ 2. Run the debug utility.

__ c. From the terminal in the `status-app` directory, type:

```
$ node inspect server.js
< Debugger listening on
ws://127.0.0.1:9229/be7df03c-1f64-495c-9e2d-4e6f7bf2367c
< For help, see: https://nodejs.org/en/docs/inspector
< Debugger attached.
Break on start in file:///home/localuser/projects/status-app/server.js:1
> 1 /*eslint no-undef: "error"*/
  2 /*eslint-env node*/
  3 var port = 3000;
debug>
```

__ d. Type `next` (or `n`) to step to the next line.

```
break in file:///home/localuser/projects/status-app/server.js:3
  1 /*eslint no-undef: "error"*/
  2 /*eslint-env node*/
> 3 var port = 3000;
  4 var parse = require('xml2js').parseString;
  5 var today = require('./today');
debug>
```

__ e. Type `cont` (or `c`) to continue execution.

```
< Listening on port 3000.
```

__ f. The application runs to completion.

__ g. Type `.exit`.

__ 3. For more information about stepping through code and setting breakpoints with the node debugger, see: https://nodejs.org/api/debugger.html

Many developers come from a programming background that includes the use of powerful IDEs and graphical debuggers.

These developers might prefer the more visual debugging experience that is provided by node-inspector.

You use the full-feature Node Inspector graphical debugger in the next part to do more comprehensive debugging on the application.

4.3. Work with Node Inspector

In this part, you work with the Node Inspector that can be used with a web-based client. Documentation of the Inspector is found at:
https://nodejs.org/en/docs/guides/debugging-getting-started/

Information

Node Inspector works in the Chrome or Opera browsers only.

The example screen captures that are used in this part show Node Inspector running in the Google Chrome browser.

No instructions are provided for installing the Chrome browser in this exercise.

V8 Inspector integration for Node.js allows attaching Chrome DevTools to Node.js instances for debugging and profiling. It uses the Chrome DevTools Protocol.

__ 1. Go to the `status-app` directory.

__ a. Open a terminal (Mac OS X, Linux) or command prompt (Microsoft Windows).

__ b. Go to the `/projects/status-app` directory.

```
$ cd ~/projects/status-app
$ pwd
/home/localuser/projects/status-app
```

__ 2. Start the inspector.

The Inspector can be enabled by passing the `--inspect` flag when starting a Node.js application.

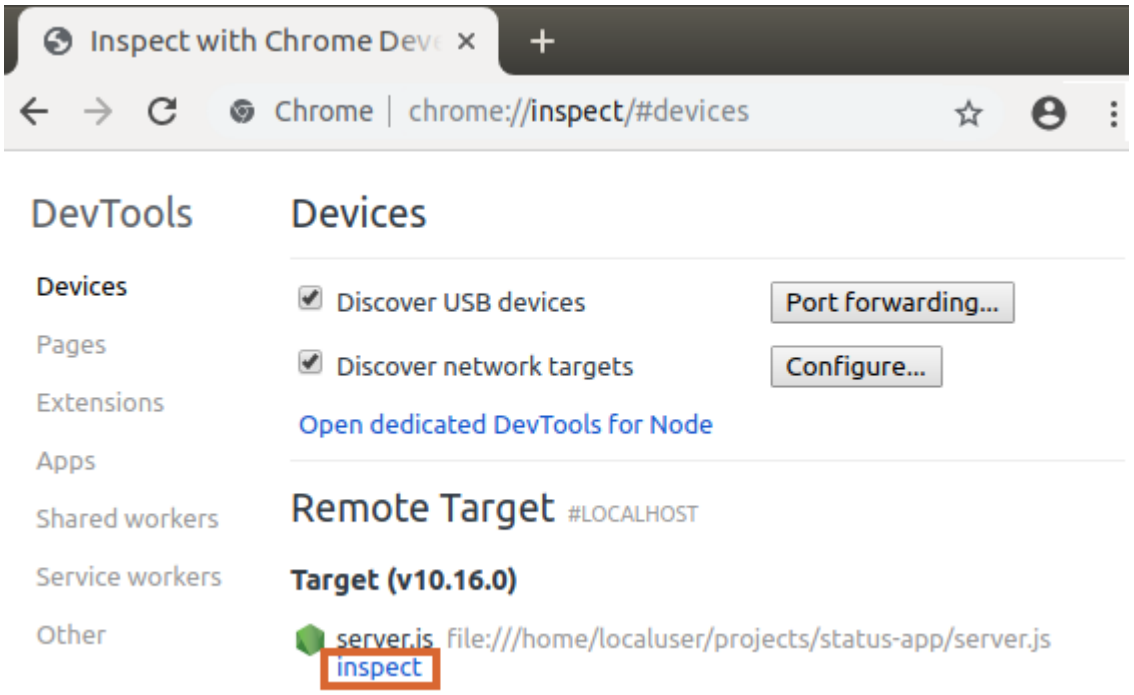
You can get the Inspector to break on the first statement of the script with the `--inspect-brk` flag.

__ a. Enable `Inspector`.

```
$ node --inspect-brk server.js
Debugger listening on
ws://127.0.0.1:9229/bbd335fb-f9e2-4578-b6e0-d13f7c07e8ab
For help, see: https://nodejs.org/en/docs/inspector
```

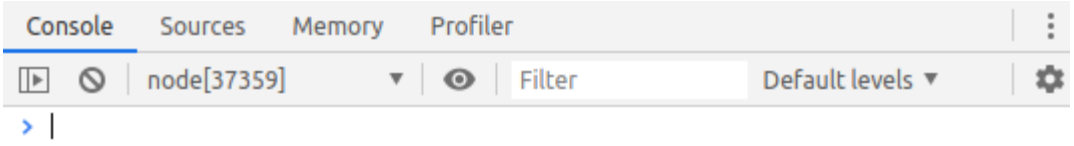
__ b. The Node Inspector starts and displays which port the debugger is listening on.

__ c. Open the Chrome browser. Type `chrome://inspect` in the browser address area.

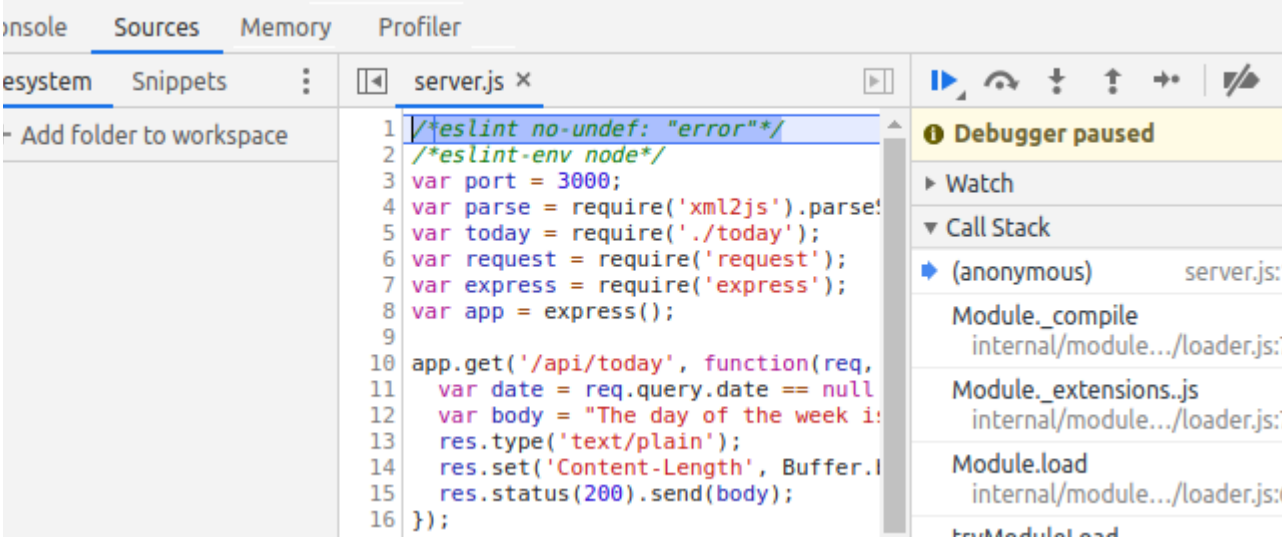


__ d. Click the `inspect` link below the name of the file that is inspected.

__ e. Node Inspector is displayed in the browser.



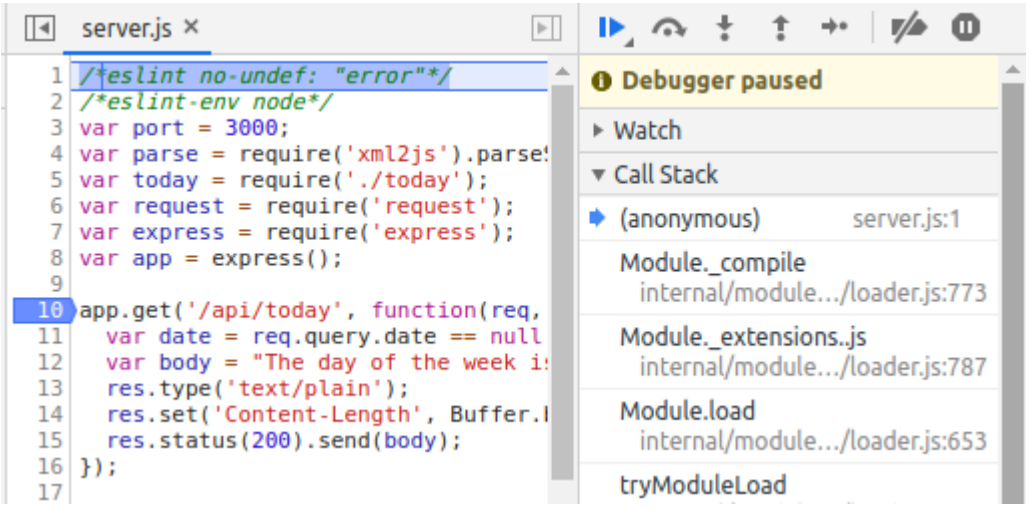
__ f. Click the **Sources** tab. Notice that Node Inspector is stopped on the first line of the server.js file.



__ 3. Review the application by using the step features of the Node Inspector.

__ a. Set a breakpoint in the `server.js` source by clicking the line number in the row that starts with:

```
app.get('/api/today', function(req, res) {
```



__ b. You now see that the code is highlighted at the beginning of the Source view. The application encounters a breakpoint at the start of the code. The Call Stack displays a function named anonymous function.

__ c. In the Node Inspector, click the option to resume script execution (F8).

__ d. The code runs to the breakpoint at the `app.get('/api/today', function(req, res)` line.

__ e. Click the option to step over the next function call (F10).

The application stops at the next function, which is the `app.get('api/weather', function(req, res)` line.

__ f. Click the option to step into the next function call (F11).

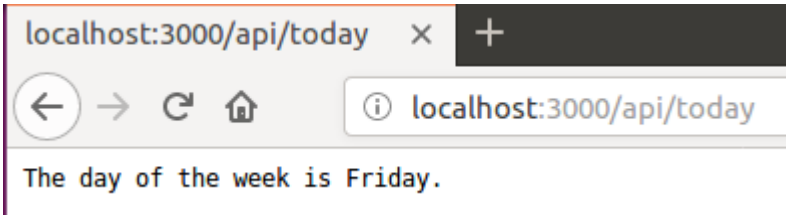
You see that Node Inspector now displays the code in `application.js`.
The `application.js` code is part of the Express module, not something you coded.

__ g. Click the option to step out of the current function.

The code is now at the last function in the `server.js` code.

__ h. Click the option to resume script execution. The code runs to completion. The message “Listening on port 3000” is displayed in the terminal window.

__ i. In another browser, type `localhost:3000/api/today`.



The day of the week is displayed in the browser. You ran through to the end of the code by using the step functions.

__ 4. Restart the application in debug mode with an automatic break at the beginning of the source code.

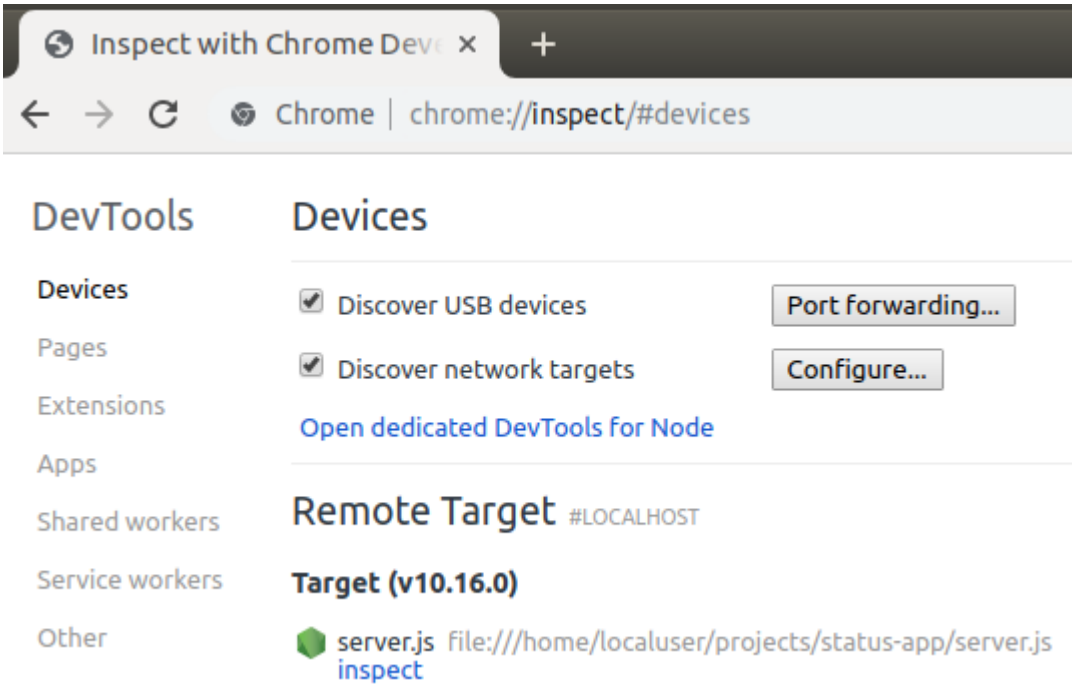
__ a. Stop the application that is running in the terminal window (Ctrl+C). From the same terminal window, type:

```
node --inspect-brk server.js
```

__ 5. Review the application by going further into the code in Node Inspector.

__ a. Open the Chrome browser. Type `chrome://inspect` in the browser address area

__ b. Click the `Open dedicated DevTools` for Node option.



__ c. Remove the current breakpoint by clearing the breakpoint in the breakpoints area.

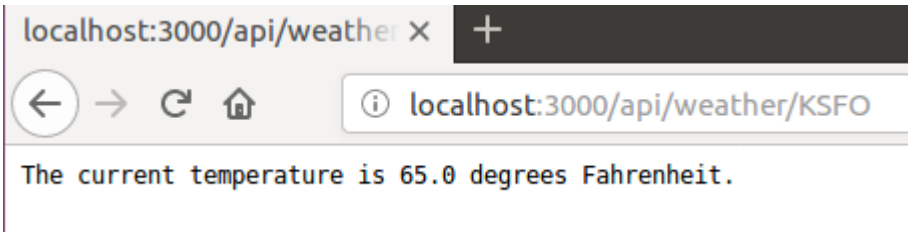
__ d. In the Node Inspector, click the option to resume script execution (F8).

__ e. The code runs to the end of the execution.

__ f. In another browser tab, type:

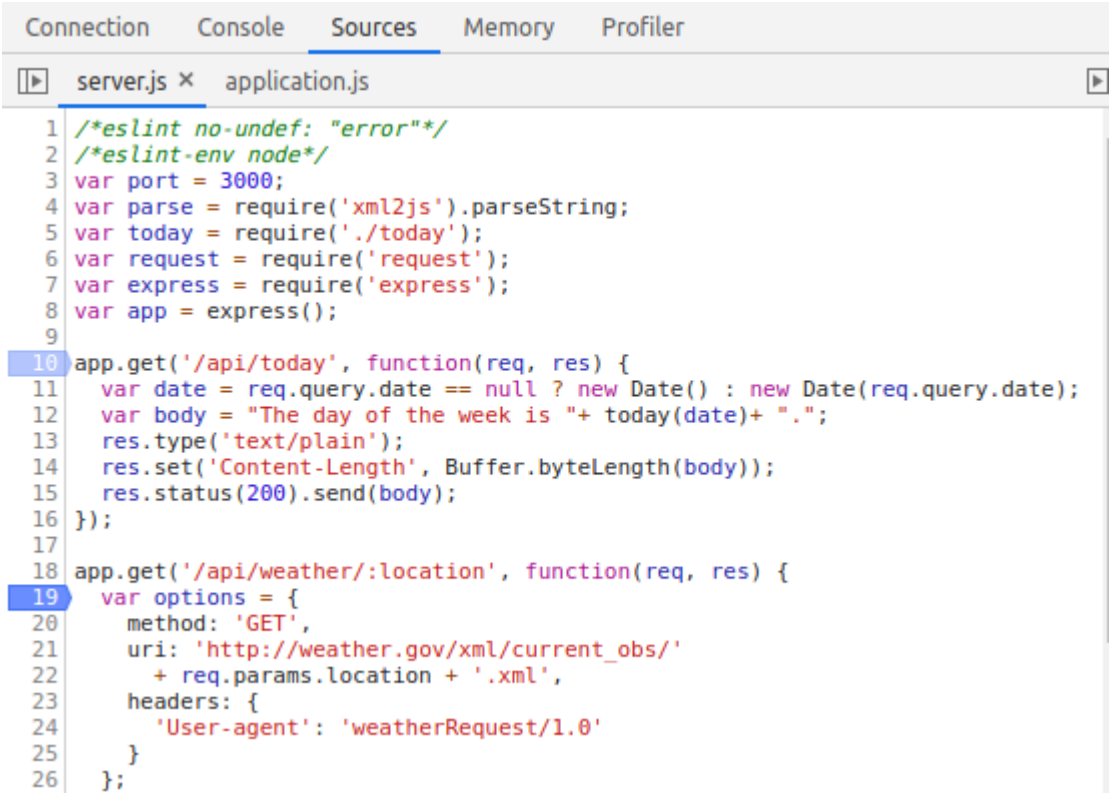
`localhost:3000/api/weather/KSFO`

__ g. The result is displayed in the browser.



__ h. On the **Sources** tab of the Node Inspector, set a new breakpoint on the `var options = {` line in the `server.js` file.

__ i. Click the icon to hide the navigator in the Inspector.



__ j. Since you set a breakpoint inside the `app.get('/api/weather/:location', function(req, res)` line, this causes a break each time you run this function.

__ k. In another browser tab, refresh or type:

`localhost:3000/api/weather/KSFO`

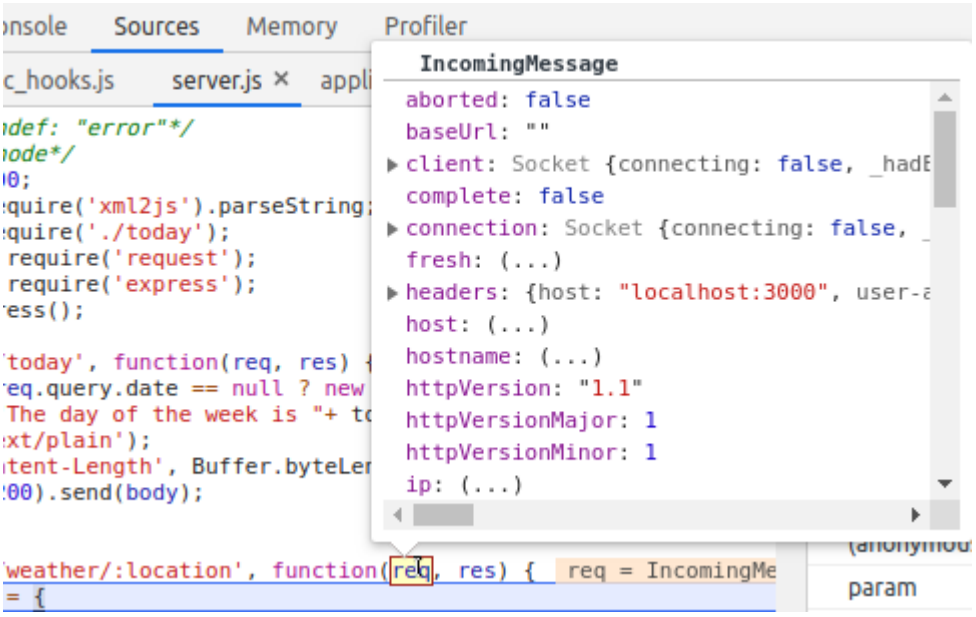
__ l. The debugger is stopped on the breakpoint.

Questions

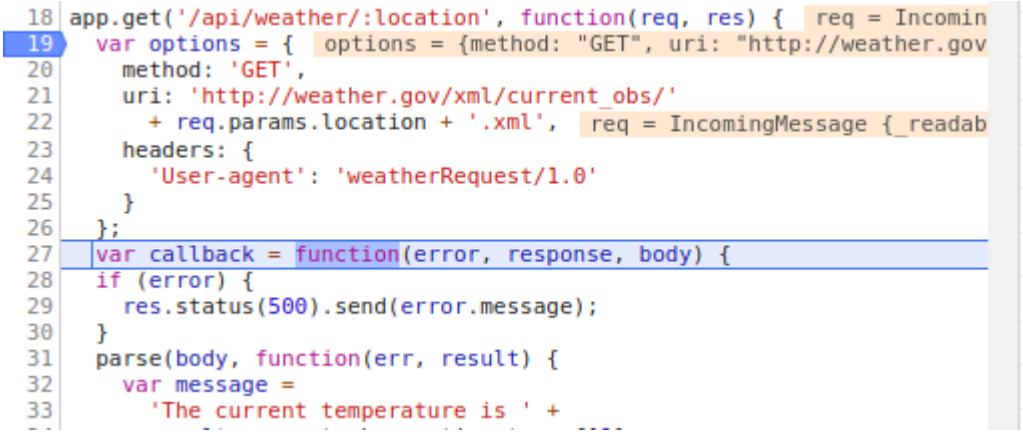
Why did the Inspector trigger the breakpoint in the `app.get("/api/weather/:location")` function?

Answer: The `app.get("/api/weather:location")` registers the `/api/weather` web route with an anonymous callback function. The callback is only called the first time that `server.js` is run. To trigger a breakpoint when the user calls `/api/weather` in the web browser, you must place a breakpoint within the callback handler.

__ m. Back in the Node Inspector, if you hover over the `req` parameter, Node Inspector displays the variables of the IncomingMessage.



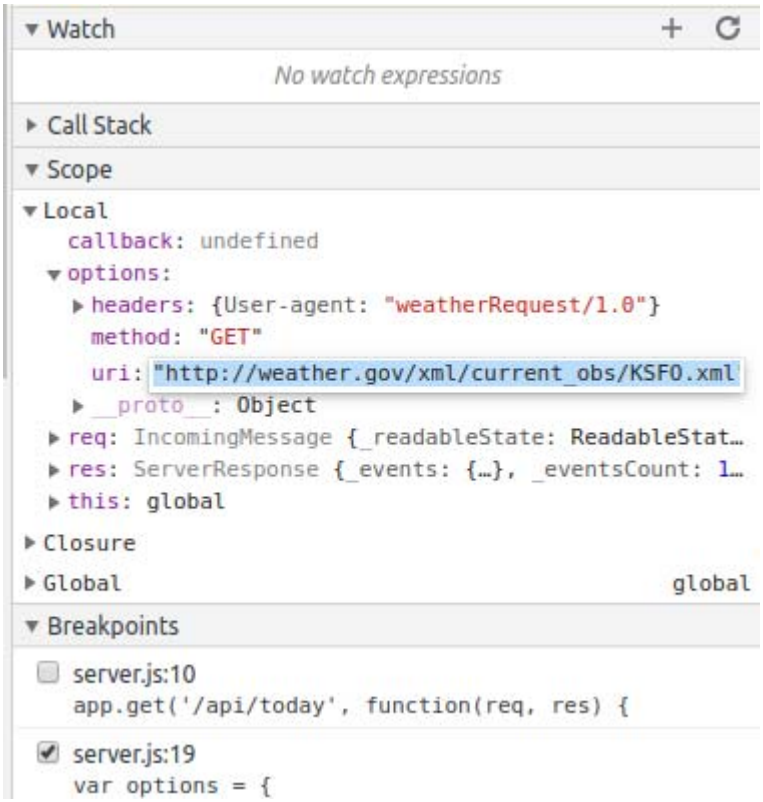
__ n. Select the option to Step into the next function call.
The code stops on the callback function.



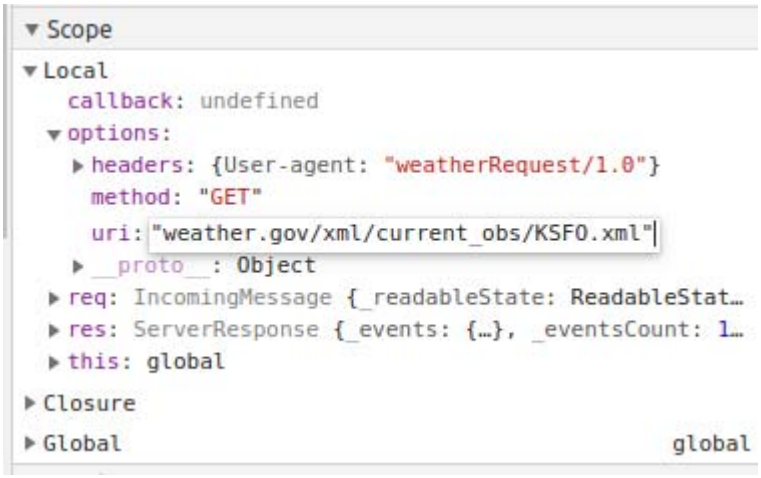
__ 6. Change the application by dynamically changing a variable value in the code inside Node Inspector.

__ a. In the Scope Variables area, expand the **options** variable.

__ b. Double-click the value of the **uri** variable to highlight it.



__ c. Change the value of the **uri** variable by removing the `http://` prefix.

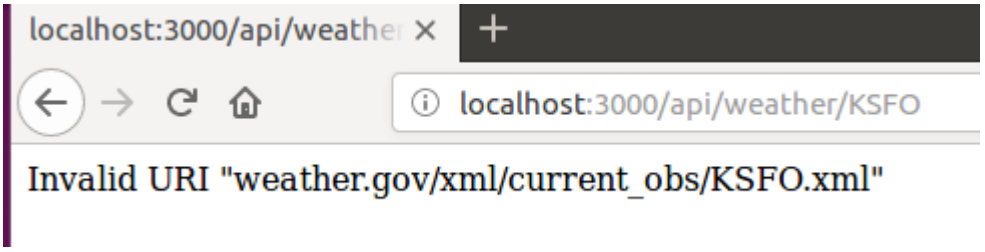


Then, press Enter.

__ d. Click the Step into the next function.

__ e. Click the Resume script execution so that the GET weather API operation ends.

__ f. Click the browser with localhost:3000/api/weather/KSFO. Instead of the temperature result that you saw earlier, an Invalid URI message is now displayed.



__ 7. Rerun the GET weather function.

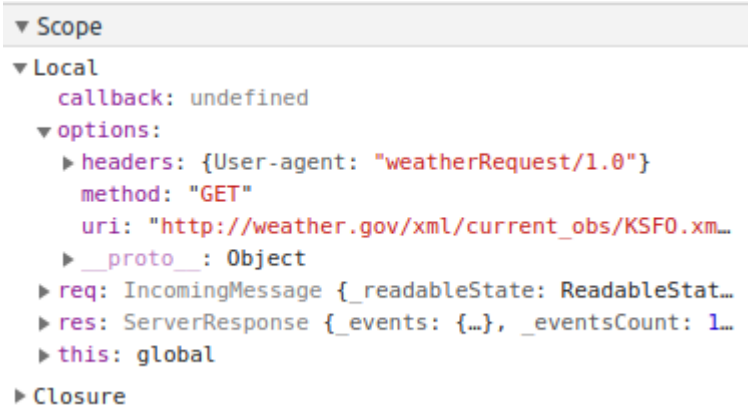
__ a. Click the browser with the localhost:3000/api/weather/KSFO to rerun the script.

__ b. Node Inspector shows that the application is paused at the breakpoint inside the `app.get('api/weather', function(req,res)`.

__ c. The code in the Node Inspector is stopped at the breakpoint.

__ d. Click the Step into the next function (F11) in Node Inspector.

__ e. In the Scope Variables area, notice that the **uri** variable under options is reset to its original value that includes the `http://` prefix.



__ f. Click the Resume script execution (F8) so that the GET weather API operation ends.

__ g. Click the browser tab with localhost:3000/api/weather/KSFO.
The result is displayed.

__ 8. Close the browser and stop the application.

__ a. Close the Chrome browser.

__ b. Stop the server application in the terminal window.

__ c. Stop the Node Inspector in the terminal, and close the terminal window.

4.4. Use package lock to set node module versions

The `package-lock.json` is automatically generated for any operations where npm modifies either the `node_modules` tree, or `package.json`. The file is created by default if your Node Package Manager is at version 5 or later.

The `package-lock.json` file describes the exact tree that gets generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.

The package lock replaces the `npm shrinkwrap` command that is used in earlier versions of npm to lock down the version numbers of all the packages and their dependencies in your `node_modules` directory. If both `package-lock.json` and `npm-shrinkwrap.json` are present in the root of a package, package-lock.json is completely ignored.

The `package-lock.json` file is intended to be committed into a source code repository to ensure repeatability of the installs.

Installing fixed package versions is important in a production environment where you need to ensure that each deployment always installs the same versions of the packages.

In this part, you review the `package-lock.json` file and explore how it is used to lock down the version numbers of all the packages in your application.

__ 1. Ensure that the package dependencies are installed in the application.

__ a. Open a terminal window.

__ b. Change to the `status-app` directory.

__ c. From the terminal window, type:

```
$ npm install
npm WARN status-app@1.0.3 No repository field.

audited 635 packages in 2.08s
found 0 vulnerabilities
```

__ d. The `npm install` ensures that all the dependent modules are installed and up-to-date.

__ e. Open the `package.json` file for your application with an editor.
You see that the version numbers of some dependencies are marked by a range.

__ f. Close the editor without saving when you are finished reviewing the file.

__ g. From the terminal, type:

```
npm install
```

The `npm install` ensures that all the dependent modules are installed and up-to-date.

__ 2. Review `package-lock.json`. The file is generated by default when the package.json file is modified.

__ h. Open the `package-lock.json` file in an editor.

The file lists all of the installed packages in the entire hierarchy.

```
ock.json (~/projects/status-app) - gedit
Open
"name": "status-app",
"version": "1.0.3",
"lockfileVersion": 1,
"requires": true,
"dependencies": {
  "@babel/code-frame": {
    "version": "7.0.0",
    "resolved": "https://registry.npmjs.org/@babel/code-frame/-/code-
rame-7.0.0.tgz",
    "integrity": "sha512-
FC2uemaknXr87bdLUkWog7nYuliM9Ij5HUcajsVcMCpQrcLmtxRbVFTIqmcSkSeYRBFBRxs2
    "dev": true,
    "requires": {
      "@babel/highlight": "^7.0.0"
    }
  },
  "@babel/highlight": {
    "version": "7.0.0",
    "resolved": "https://registry.npmjs.org/@babel/highlight/-/
ighlight-7.0.0.tgz",
    "integrity": "sha512-UFMC4ZeFC48Tpvj7C8UgLvtkaUuovQX+5xNWrsIoMG8o2z
(FKjKaN9iVmS84dPwVN00W4wPmqvYoZF3EGAsfw==",
```

The `npm install` command creates the module tree that is displayed in the `package-lock.json` file. When reproducing the structure that is described in the file, npm uses the specific files that are referenced in the “resolved” field, if available. Otherwise, npm falls back to normal package resolution by using the “version” number. The value in the “resolved” field includes the name of the registry and the file name that are used when installing the modules.

__ i. In the editor search for the string express.

```
pen
      "is-stream": "^1.1.0",
      "npm-run-path": "^2.0.0",
      "p-finally": "^1.0.0",
      "signal-exit": "^3.0.0",
      "strip-eof": "^1.0.0"
    }
  },
  "express": {
    "version": "4.17.1",
    "resolved": "https://registry.npmjs.org/express/-/express-4.17.1.tgz
    "integrity": "sha512-mHJ9O79RqluphRrcw2X/
13k9tVv8YcoyY4Kkh4WDMUYKRZUq0h1o0w2rrrxBqM7VoeUVqgb27xlEMXTnYt4g==",
    "requires": {
      "accepts": "~1.3.7",
      "array-flatten": "1.1.1",
      "body-parser": "1.19.0",
```

When the node_modules dependency tree is installed, express version 4.17.1 is installed. Compare this with the contents of the package.json file you saw earlier.

__ j. Close the editor without saving any changes when you are finished reviewing the file.

Note

When you have development dependencies that are specified with the devDependencies statement in your `package.json` file, `package-lock.json` includes these dependencies. If you want the install utility to exclude the development dependencies, run the command with the `--production` or the `--prod` flag:

```
npm install --prod
```

4.5. Clone and test the packed locked application (optional)

__ 1. Create a clone directory in the `/projects` directory.

```
cd ..
mkdir status-app-clone
```

__ 2. Change to the `/projects/status-app` directory.

```
cd status-app
```

__ 3. Copy the files in the directory to the `/projects/status-app-clone` directory.

```
cp *.* ../status-app-clone/
```

__ 4. Change directory to the newly created `status-app-clone`.

```
cd ../status-app-clone
ls
```

You see that the files are copied from the `status-app` to the `status-app-clone` directory without the subdirectories.

__ 5. Run the `npm install` command from the new location.

__ a. From the terminal, type:

```
npm install
```

__ b. The `npm install` command ensures that all the dependent modules are installed by matching the version number and installation URIs that are specified in the `package-lock.json` file.
The cloned application and the original application both install the identical package and package dependency version numbers.

__ 6. Test the application from the new location.

__ a. From the terminal, type: `node server.js`

__ b. In a browser, type: `localhost:3000/api/today`
The output is displayed in the browser.

__ 7. In the terminal (Mac OS, Linux) or command prompt (Microsoft Windows), press Ctrl+C to stop the application.

For more information about the npm package lock file, see:
https://docs.npmjs.com/files/package-lock.json

4.6. Use script objects and npm to build node applications

In previous exercises, you used the `npm run` command to run scripts. In this part, you see how the npm command can be used to automate code validation, testing, and running of applications.

__ 1. Use the `npm run` command to display a list of script properties for the application.

__ a. In a terminal window, go to the `projects/status-app` folder.

__ b. In the terminal, type:

```
$ npm run
```

The `npm run` command displays the names of the scripts for the application in the terminal window.

```
Lifecycle scripts included in status-app:
  start
    node server.js
  test
    mocha test
```

__ c. If you want to run a specific script from the `package.json` file, you use the command:

```
npm run <script>
```

Note

You do not need to include the `run` parameter for the test or start script.

You can use the shorthand `npm <script>`.
If you want only the output of the test, use the `-s` flag, which silences the output from the `npm run` command.

```
npm run test -s
```

__ 2. Open the `package.json` file in an editor.

__ 3. Modify the properties in the scripts object in the `package.json` file.

__ a. Change the `scripts` object in the `package.json` file to include the line for `lint`:

```
"scripts": {
    "lint": "echo 'Running ESLint now' && eslint *.js",
    "start": "node server.js",
    "test": "mocha test"
  },
```

__ b. **Save** the changes.

Information

You can chain tasks within the scripts object by using the `&&` syntax. An example of chaining the `echo` command with running ESLint on the JavaScript files is shown in the preceding scripts snippet.

__ 4. Run the lint script.

__ a. In a terminal window, type:

```
$ npm run lint -s
Running ESLint now
```

The lint validator runs and displays the result in the terminal window. No errors are reported.

__ 5. Modify `server.js` to cause a validation error.

__ a. Open `server.js` with an editor.

__ b. In the `app.get('/api/weather/:location'` function, change the "`if (error)`" in the callback handler to "`if (eror)`"

```
var callback = function(error, response, body) {
    if (eror) {
```

__ c. Save the changes.

__ 6. Run the lint script.

__ a. In the terminal window, type: `npm run lint -s`

__ b. The ESLint JavaScript validator runs and displays the output that indicates an expected literal on the right side of the "==" in the source code.

```
$ npm run lint -s
Running ESLint now

/home/localuser/projects/status-app/server.js
  28:7  error  'eror' is not defined  no-undef

? 1 problem (1 error, 0 warnings)
```

The error displays the name of the source file and line number and position where the validation failed.

__ c. Fix the error in the `server.js` file by setting the statement back to its original value, and save it.

__ d. Rerun the npm lint script. You see no errors.

__ 7. Create a build script in the `package.json` file.

You can chain any number of scripts together to make a build script that can include validation, testing, and running the code.

__ a. Open `package.json` in an editor. Change the scripts object to match the code:

```
{
  "name": "status-app",
  "version": "1.0.3",
  "description": "Return Node runtime status information",
  "main": "today.js",
  "scripts": {
    "lint": "echo 'Running ESLint now' && eslint *.js",
    "start": "node server.js",
    "test": "mocha test",
    "build": "echo 'Running build' && npm run lint && npm run test"
  },
  "author": "John Doe <jdoe@example.com>",
  "license": "ISC",
"dependencies": {
    "express": "^4.17.1",
    "request": "^2.88.0",
    "xml2js": "^0.4.19"
  },
  "devDependencies": {
    "chai": "^4.2.0",
    "eslint": "^5.16.0",
    "mocha": "^6.1.4",
    "supertest": "^4.0.2"
  }
}
```

__ b. Save the change.

__ 8. Run the build script.

__ a. In a terminal window, type:

```
npm run build
```

You see the output that is displayed in the terminal.

```
Running build

> status-app@1.0.3 lint /home/localuser/projects/status-app
> echo 'Running ESLint now' && eslint *.js

Running ESLint now

> status-app@1.0.3 test /home/localuser/projects/status-app
> mocha test



Listening on port 3000.
  /api/today
    GET with #date query parameter
      v returns Monday for 01-January-2018 (58ms)
      v returns Tuesday for 02-January-2018
      v returns Wednesday for 03-January-2018
      v returns Thursday for 04-January-2018
      v returns Friday for 05-January-2018
      v returns Saturday for 06-January-2018
      v returns Sunday for 07-January-2018

  Today
    now
      v should return the current day of the week
    01 January 2018
      v should return Monday for 01-January-2018
    02 January 2018
      v should return Tuesday for 02-January-2018


  10 passing (118ms)
```

End of exercise

Solution

status-app/package.json:

```
{
  "name": "status-app",
  "version": "1.0.3",
  "description": "Return Node runtime status information",
  "main": "server.js",
  "scripts": {
    "lint": "echo 'Running ESLint now' && eslint *.js",
    "start": "node server.js",
    "test": "mocha test",
    "build": "echo 'Running build' && npm run lint && npm run test"
  },
  "author": "John Doe <jdoe@example.com>",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "request": "^2.88.0",
    "xml2js": "^0.4.19"
  },
  "devDependencies": {
    "chai": "^4.2.0",
    "eslint": "^5.16.0",
    "mocha": "^6.1.4",
    "supertest": "^4.0.2"
  }
}
```

Exercise review and wrap-up

In the exercise, you worked with various utilities that are used to debug Node applications.

First, you reviewed an example that used the standard command-line debug utility of Node.js.

You installed the Node Inspector graphical debugger, and you explored how to use the Node Inspector with the Chrome browser to debug node applications.

Next, you reviewed the npm shrinkwrap command for locking the version numbers of package dependencies.

Finally, you reviewed how to stack npm commands to create a build script and use the npm run command to run the scripts.

Exercise 5. Deploying a REST API on IBM Cloud (optional)

Estimated time

00:45

Overview

In this exercise, you deploy a packaged REST API into your IBM Cloud account. You learn how to install the IBM Cloud Developer Tools command-line interface (CLI) utility, and connect and manage your IBM Cloud account with the Cloud CLI. You also learn how to deploy your REST API as an IBM Cloud Node.js application.

What is the user story

As an API developer, you want to deploy a Node application on IBM Cloud so that you can publish and manage interaction services that run on a Cloud application platform.

Objectives

After completing this exercise, you should be able to:

- Register for an IBM Cloud account
- Install the IBM Cloud command-line interface (CLI)
- Enable an existing application for IBM Cloud
- Build and run an application on a local container that is built with the IBM Cloud Developer Tools
- Deploy a node application into IBM Cloud
- Retrieve the application logs from an IBM Cloud application

Introduction

In an earlier exercise, you developed, tested, and packaged a REST API written as a Node application. Before you can manage and enforce the API with API Connect, you must host the API on a server.

IBM Cloud is a platform-as-a-service solution that simplifies the task of building, deploying, and managing web applications. IBM Cloud supports Cloud Foundry buildpacks: prebuilt and tested containers with an application runtime engine.

In this exercise, you deploy your Node.js application on the IBM Cloud environment. With a buildpack solution, you do not need to install, configure, and manage the server operating system or the Node runtime environment software.

You use the IBM Cloud command-line interface (CLI) to manage, deploy, and monitor your Node-based API running as an IBM Cloud application.

Requirements

You must complete the steps in Exercise 4 before starting this lab.

This exercise requires a workstation with internet access. You can complete this exercise on a computer with a Linux, Mac OS X, or Microsoft Windows operating system.

For a list of supported operating systems and platforms, see: https://nodejs.org/en/download/

Exercise instructions

Preface

- The authors of this exercise tested the instructions on a Linux operating system. Unless otherwise specified, the instructions should work with minor modifications on Mac OS X and Microsoft Windows operating systems.
- The command-line examples work in a UNIX-compatible shell environment, such as Linux or Mac OS X. For example, use a text editor of your choice when the instructions list the gedit editor as an example.

5.1. Update the package manifest file

Update the minor version number in the `package.json` file to differentiate your work in this exercise from other exercises.

__ 1. Go to the `status-app` directory.

__ a. Open a terminal (Mac OS X, Linux) or command prompt (Microsoft Windows).

__ b. Go to the `/projects/status-app` directory.

```
$ cd projects/status-app/
```

__ 2. Update the version number in the package manifest to 1.0.4.

__ a. Open `package.json` in a text editor.

```
$ gedit package.json
```

__ b. Update the version field to 1.0.4.

```
"version": "1.0.4",
```

__ c. Save and close `package.json`.

__ 3. Update the version number in the package-lock,json to match the changed package manifest.

__ a. `$ rm -f package-lock.json && npm install`

__ b. The package-lock.json file is regenerated.

5.2. Register for an IBM Cloud account

You must sign up and create your IBM Cloud account before you start this exercise. The IBM Cloud account provides a runtime environment for you to run node.js applications in the Cloud.

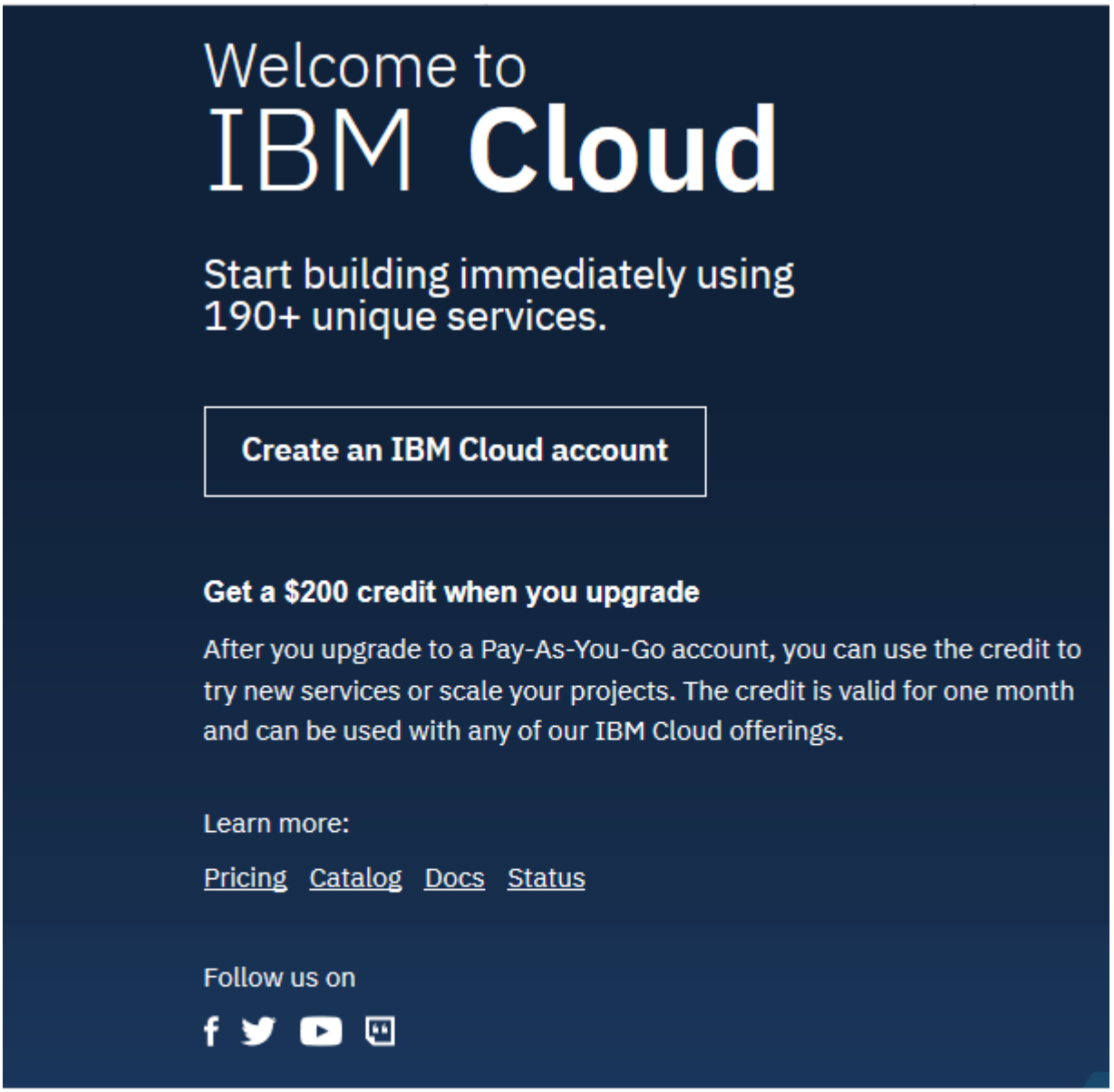
Note

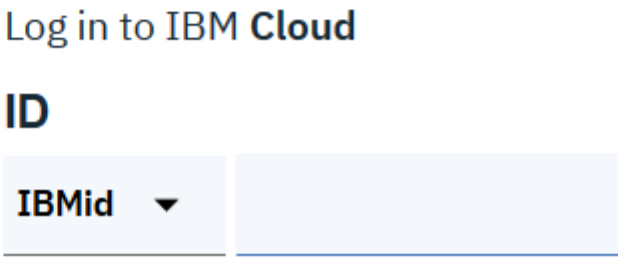
If you have an IBM Cloud account, skip ahead to the next section of this exercise.

__ 1. Sign up for an IBM Cloud account.

__ a. Open https://cloud.ibm.com in a web browser.

__ b. Click **Create an IBM Cloud account**.





__ c. If you previously registered for an IBM ID, type your IBMid to sign in to the IBM Cloud. Then, click **Continue**.

__ d. Otherwise, complete the form and register for a Cloud account.

__ e. Review and accept the terms and conditions.

__ 2. Check your email inbox for a message with an email validation link.

Important

As part of the sign-up process, you must *validate your email address*. Remember to register to Cloud with an email address with which you can access.

__ 3. Wait until your IBM Cloud account is activated.

5.3. Review your IBM Cloud account

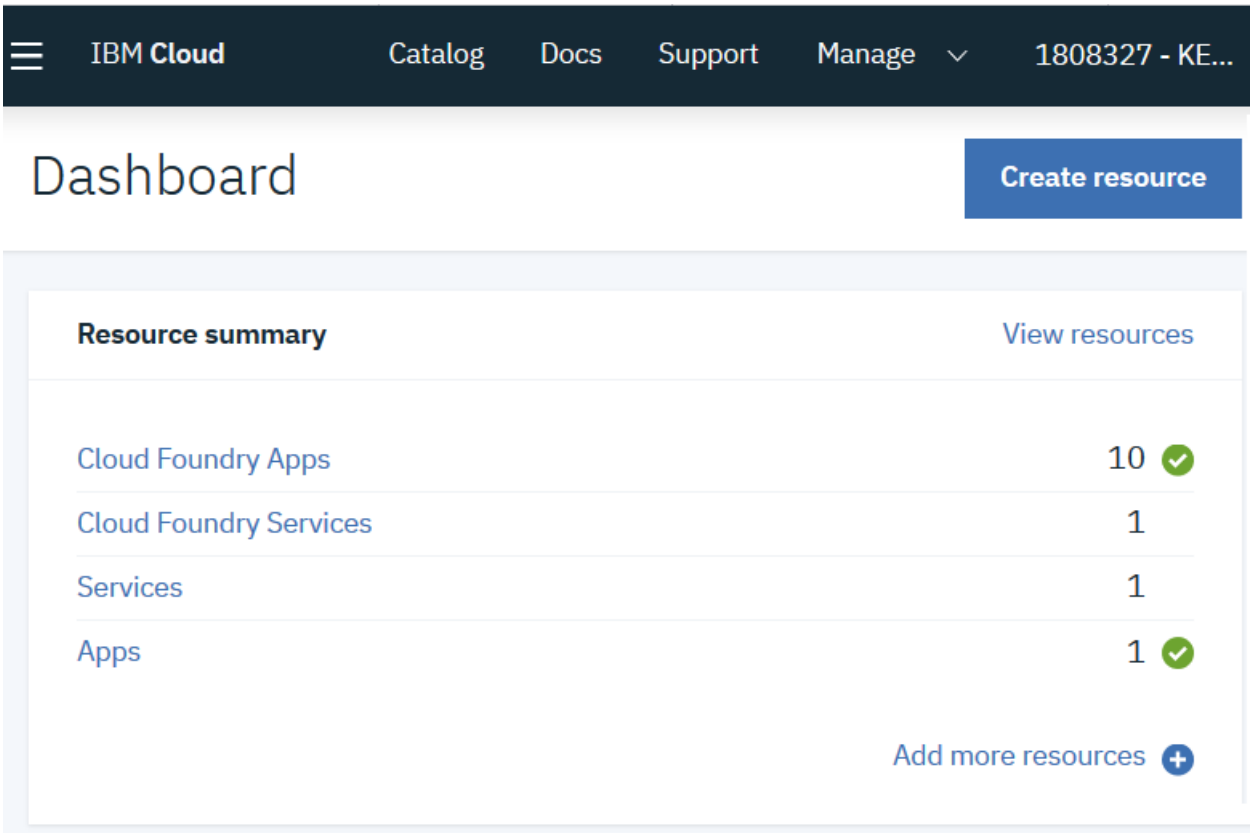
Before you can create applications with the IBM Cloud Developer Tools command-line interface (CLI) utility, you must have an IBM Cloud account. In this section, log in to the web-based Cloud dashboard and review your account settings.

__ 1. Open the IBM Cloud dashboard.

__ a. Open https://cloud.ibm.com in a web browser.

__ b. Log in to your IBM Cloud account.

__ c. The IBM Cloud dashboard is displayed.



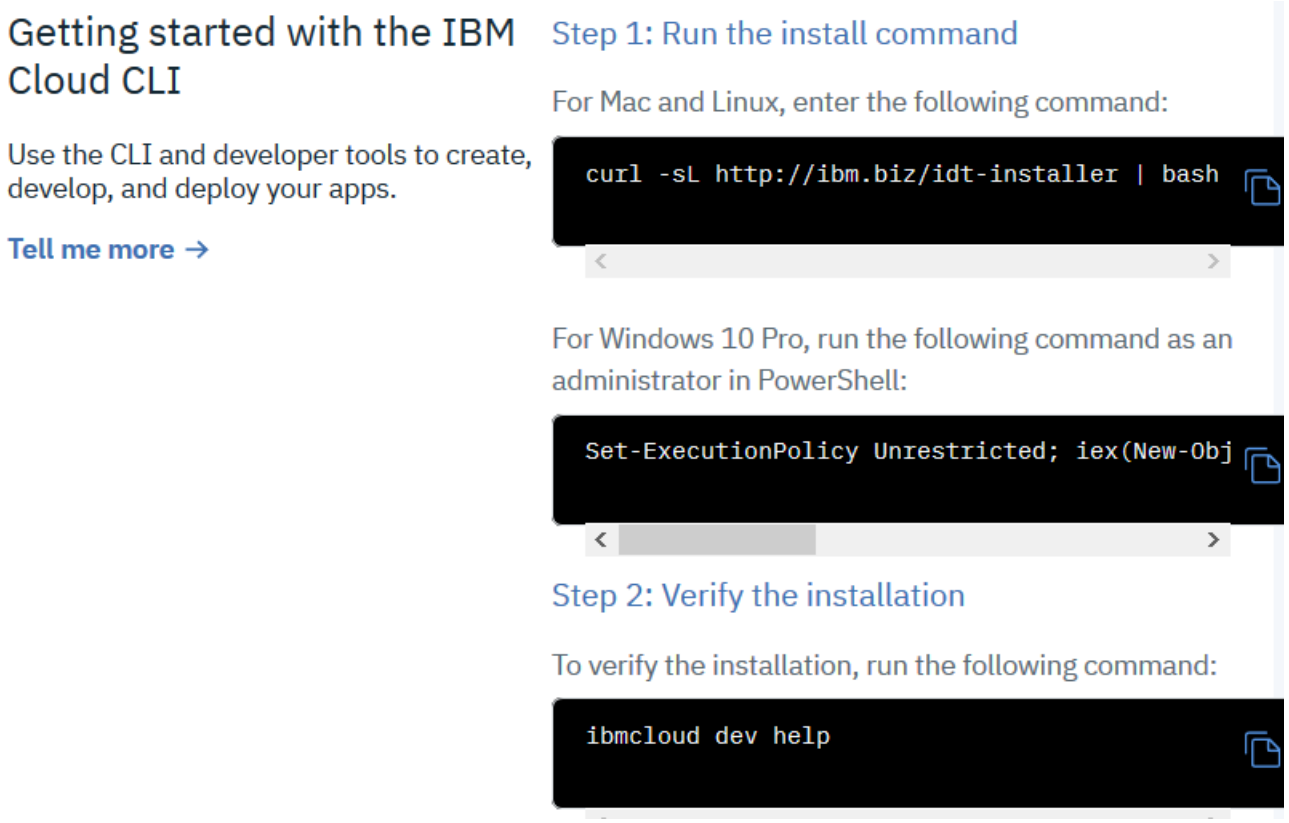
5.4. Install the IBM Cloud CLI

In this step, you review the getting started documentation and install the IBM Cloud CLI.

__ 1. Open the Cloud documentation.

__ a. Click **Docs** from the Cloud console navigation bar.

__ b. Click the **Tools** tab in the documentation.
The steps to install the Cloud CLI are displayed.



__ 2. Install the curl utility if it is not already installed on your workstation.

__ a. Open a terminal (Mac OS X, Linux) or command prompt (Microsoft Windows).

__ b. Go to the home directory.

```
$ cd ~
```

__ c. Install the curl utility:

```
$ sudo apt install curl
```

__ d. Verify that the curl utility is installed:

```
$ curl --version
curl 7.47.0 (x86_64-pc-linux-gnu) libcurl/7.47.0 GnuTLS/3.4.10 zlib/1.2.8
libidn/1.32 librtmp/2.3
```

__ 3. Install the IBM Cloud CLI.

__ a. Install the Cloud CLI:

```
$ curl -sL http://ibm.biz/idt-installer | bash
```

__ b. Add the docker group to the user account

```
$ sudo usermod -aG docker $USER
```

__ c. Log out of the user account. Then, log in to the same user.

__ d. Test the docker command.

```
$ docker version
Client:
 Version:           18.09.6
 API version:       1.39
 Go version:        go1.10.8
 Git commit:        481bc77
 Built:             Sat May  4 02:35:27 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          18.09.6
  API version:      1.39 (minimum version 1.12)
  Go version:       go1.10.8
  Git commit:       481bc77
  Built:            Sat May  4 01:59:36 2019
  OS/Arch:          linux/amd64
  Experimental:     false
```

__ 4. Verify the installation of the Cloud CLI.

```
$ ibmcloud dev help
NAME:
   ibmcloud dev - A CLI plugin to create, manage, and run applications on IBM
Cloud

USAGE:
   ibmcloud dev command [arguments...] [command options]

VERSION:
   2.2.0
COMMANDS:
   build             Build the application in a local container
   code              Download the code from an application
   console           Opens the IBM Cloud console for an application
   create           Creates a new application and gives you the option to add
services
   diag            This command displays version information about installed
dependencies
   debug             Debug your application in a local container
   delete            Deletes an application from your space
   deploy            Deploy an application to IBM Cloud
   edit              Add or remove services for your application
   enable            Add IBM Cloud files to an existing application.
   get-credentials   Gets credentials required by the application to enable
use of connected services.
   list              List all IBM Cloud applications in a space
   run               Run your application in a local container
   shell             Open a shell into a local container
   status            Check the status of the containers used by the CLI
   stop              Stop a container
   test              Test your application in a local container
   view              View the URL of your application
   help              Show help
Enter 'ibmcloud dev help [command]' for more information about a command.

GLOBAL OPTIONS:
   --version, -v                 Print the version
   --help, -h                    Show help
```

5.5. Enable the existing application for IBM Cloud

The IBM Cloud Developer Tools includes the enable command to enable an existing application for IBM Cloud. You run the command in this part.

The enable command generates and adds files that can be used for local Docker containers, Cloud Foundry deployment, Cloud Foundry Enterprise Environment deployment, or Kubernetes Container deployment. All deployment environments can be leveraged through a manual deployment or by using a DevOps toolchain.

__ 1. Ensure that you are in the `status-app` directory.

__ a. Open a terminal (Mac OS X, Linux) or command prompt (Microsoft Windows).

__ b. Go to the `/projects/status-app` directory.

```
$ cd projects/status-app/
```

__ 2. Run the IBM Cloud Developer Tools command to enable the application for IBM Cloud:

```
$ ibmcloud dev enable --force --language node --no-create --trace

IBM Cloud CLI version:    0.16.1+99fad54-2019-05-28T06:19:24+00:00
dev plugin version:       2.2.0
cr plugin version:        0.1.382
cs plugin version:        0.3.34
cf plugin version:        6.41.0+dd4c76cdd.2018-11-28
docker version:           18.09.6, build 481bc77
docker-compose version:   MISSING
kubectl version:          MISSING
helm client version:      v2.14.0
git version:              2.21.0

The enable feature is currently in Beta.
Please provide your experience and feedback at:
https://ibm-cloud-tech.slack.com/messages/developer-tools/

Only server-side apps are supported by the enable feature


REQUEST:  GET
https://us-south.devx.cloud.ibm.com/appmanager/v1/starters?tag=notDeveloperC
onsole hasRequiredCapabilities=false

REQUEST HEADER:  {
    "Authorization": "",
    "Content-Type": "application/json",
    "User-Agent": "bx_dev 2.2.0 user"
}

...

Unzipping app into directory: /home/localuser/projects/status-app

The following files were added to your app:


.cfignore
.dockerignore
Dockerfile
Dockerfile-tools
Jenkinsfile
README.md
manifest.yml
cli-config.yml
run-debug
run-dev
```

```
.bluemix/deploy.json
.bluemix/pipeline.yml
.bluemix/toolchain.yml
.bluemix/scripts/container_build.sh
.bluemix/scripts/kube_deploy.sh
chart/statusapp/Chart.yaml
chart/statusapp/values.yaml
chart/statusapp/templates/basedeployment.yaml
chart/statusapp/templates/deployment.yaml
chart/statusapp/templates/hpa.yaml
chart/statusapp/templates/istio.yaml
chart/statusapp/templates/service.yaml
.gitignore
LICENSE

The app, status-app, has been successfully saved into the current
directory.
```

__ 3. The application is cloud enabled.

__ a. List the contents of the status-app directory.

```
$ ls -al

total 192
drwxrwxr-x   6 localuser localuser  4096 Jun  5 09:04 .
drwxrwxr-x   4 localuser localuser  4096 Jun  3 16:07 ..
drwxr-xr-x   3 localuser localuser  4096 Jun  5 09:04 .bluemix
-rw-rw-r--   1 localuser localuser    39 Jun  5 09:04 .cfignore
drwxr-xr-x   3 localuser localuser  4096 Jun  5 09:04 chart
-rw-rw-r--   1 localuser localuser  2413 Jun  5 09:04 cli-config.yml
-rw-rw-r--   1 localuser localuser   425 Jun  5 09:04 Dockerfile
-rw-rw-r--   1 localuser localuser   490 Jun  5 09:04 Dockerfile-tools
-rw-rw-r--   1 localuser localuser    34 Jun  5 09:04 .dockerignore
-rw-rw-r--   1 localuser localuser   351 May 30 10:40 .eslintrc.js
-rw-rw-r--   1 localuser localuser    80 Jun  5 09:04 .gitignore
-rw-r--r--   1 localuser localuser   198 Jun  5 09:04 .ibm-project
-rw-rw-r--   1 localuser localuser    98 Jun  5 09:04 Jenkinsfile
-rw-rw-r--   1 localuser localuser  1059 Jun  5 09:04 LICENSE
-rw-rw-r--   1 localuser localuser   170 Jun  5 09:04 manifest.yml
drwxrwxr-x 274 localuser localuser 12288 May 30 13:59 node_modules
-rw-rw-r--   1 localuser localuser   620 Jun  4 16:09 package.json
-rw-rw-r--   1 localuser localuser 90523 Jun  4 16:10 package-lock.json
-rw-rw-r--   1 localuser localuser  3135 Jun  5 09:04 README.md
-rw-rw-r--   1 localuser localuser   471 Jun  5 09:04 run-debug
-rw-rw-r--   1 localuser localuser    77 Jun  5 09:04 run-dev
-rw-rw-r--   1 localuser localuser  1315 Jun  3 17:02 server.js
drwxrwxr-x   2 localuser localuser  4096 May 30 14:28 test
-rw-rw-r--   1 localuser localuser   210 May 30 10:47 today.js
```

5.6. Build and run the application in a local container (optional)

The IBM Cloud Developer Tools includes the Docker engine for building and running the application in a local container.

In this part, you build and test that the application can run on a local Docker container. Normally, this is only necessary if you deploy the application as a container on the IBM Cloud. You deploy your application as a Cloud Foundry application on the IBM Cloud later in this exercise.

__ 1. Review the Docker build file that was generated by the IBM Cloud Developer Tools enable command in a previous step.

__ a. Open the `Dockerfile` in the editor.

```
$ gedit Dockerfile

FROM node:8-stretch

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
 && apt-get dist-upgrade -y \
 && apt-get clean \
 && echo 'Finished installing dependencies'

# Install npm production packages
COPY package.json /app/
RUN cd /app; npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

EXPOSE 3000

CMD ["npm", "start"]
```

__ b. The Dockerfile contains the commands to build a Docker image that contains the application and its prerequisite libraries and modules.

__ c. Close the editor when you are finished reviewing the file.

__ 2. Build a Docker image for the Node application.

__ a. Ensure that you are in the projects/status-app directory.

__ b. Type the IBM Cloud Development Tools command to build the Docker image:

```
$ ibmcloud dev build --use-root-user-tools --trace

IBM Cloud CLI version:    0.16.1+99fad54-2019-05-28T06:19:24+00:00
dev plugin version:       2.2.0
cr plugin version:        0.1.382
cs plugin version:        0.3.34
cf plugin version:        6.41.0+dd4c76cdd.2018-11-28
docker version:           18.09.6, build 481bc77
docker-compose version:   MISSING
kubectl version:          MISSING
helm client version:      v2.14.0
git version:              2.21.0

Validating Docker image name
OK
Using these variable values:
01: ContainerName string =
02: ContainerNameRun string = statusapp-express-run
03: ContainerNameTools string = statusapp-express-tools
04: HostPathRun string = .
05: HostPathTools string = .
06: ContainerPathRun string = /app
07: ContainerPathTools string = /app
08: IsUseRootUserTools bool = true
09: BuildCmdRun string = npm install
10: BuildCmdDebug string = npm install
11: TestCmd string = npm run test
12: DebugCmd string = npm run debug
13: RunCmd string =
14: ContainerPortMap string = 3000:3000
15: ContainerPortMapDebug string = 9229:9229
16: ImageNameTools string = statusapp-express-tools
17: ImageNameRun string = statusapp-express-run
18: DockerfileRun string = Dockerfile
19: DockerfileTools string = Dockerfile-tools
20: ContainerMountsRun []map[string]string =
[map[./node_modules_linux:/app/node_modules]]
21: ContainerMountsTools []map[string]string =
[map[./node_modules_linux:/app/node_modules]]
22: IsDebug bool = false
23: IsTrace bool = true
24: Version string = 0.0.3
25: DeployTarget string =
26: IsForce bool = false
27: Language string =
28: IsNoCreate bool = false
29: IbmCluster string =
```

```
30: ChartPath string = chart/statusapp
31: DockerRegistry string =
32: GeneratedID string = c50ca169-66cb-4001-b804-82984558ea7e
33: CredsFilepath string = server/localdev-config.json, vcap-local.js,
credentials.json, localdev-config.json
34: NoOpen bool = false
35: WebAppRoot string =
36: ContainerShell string = /bin/sh
37: ContainerShellTarget string = tools
38: AppID string =
39: Hostname string =
40: Domain string =
Checking if Docker container statusapp-express-tools is running
OK
Checking Docker image history to see if image already exists
OK
Creating image statusapp-express-tools based on Dockerfile-tools ...

Executing docker image build --file Dockerfile-tools --tag
statusapp-express-tools --rm --pull --build-arg bx_dev_userid=0
--build-arg
bx_dev_user=root .


Waiting for Docker image to build
Sending build context to Docker daemon  153.6kB
Step 1/18 : FROM node:8-stretch
8-stretch: Pulling from library/node
c5e155d5a1d1: Pulling fs layer
221d80d00ae9: Pulling fs layer
4250b3117dca: Pulling fs layer
3b7ca19181b2: Pulling fs layer
425d7b2a5bcc: Pulling fs layer
69df12c70287: Pulling fs layer
2a68245de447: Pulling fs layer
4f61e9705839: Pulling fs layer
e17df9513db6: Pulling fs layer
3b7ca19181b2: Waiting
425d7b2a5bcc: Waiting
69df12c70287: Waiting
2a68245de447: Waiting
4f61e9705839: Waiting
e17df9513db6: Waiting
4250b3117dca: Verifying Checksum
4250b3117dca: Download complete
221d80d00ae9: Verifying Checksum
221d80d00ae9: Download complete
c5e155d5a1d1: Verifying Checksum
```

```
c5e155d5a1d1: Download complete
69df12c70287: Verifying Checksum
69df12c70287: Download complete
3b7ca19181b2: Verifying Checksum
3b7ca19181b2: Download complete
c5e155d5a1d1: Pull complete
2a68245de447: Verifying Checksum
2a68245de447: Download complete
4f61e9705839: Verifying Checksum
4f61e9705839: Download complete
221d80d00ae9: Pull complete
e17df9513db6: Verifying Checksum
e17df9513db6: Download complete
4250b3117dca: Pull complete
3b7ca19181b2: Pull complete
425d7b2a5bcc: Verifying Checksum
425d7b2a5bcc: Download complete
425d7b2a5bcc: Pull complete
69df12c70287: Pull complete
2a68245de447: Pull complete
4f61e9705839: Pull complete
e17df9513db6: Pull complete
Digest:
sha256:957cab2653bde49d195e0a98c6ae0c1700ed51eb94fce30faadaceacf331a0a1
Status: Downloaded newer image for node:8-stretch
 ---> c5d36fec051d
Step 2/18 : ENV PORT 3000
 ---> Running in 5ae6c531b20b
Removing intermediate container 5ae6c531b20b
 ---> d1c0d970d03f
Step 3/18 : ENV NODE_HEAPDUMP_OPTIONS nosignal
 ---> Running in 62573b6412ab
Removing intermediate container 62573b6412ab
 ---> 83eac4b4f606
Step 4/18 : EXPOSE 3000
 ---> Running in 73e0afc7cb18
Removing intermediate container 73e0afc7cb18
 ---> 31fd0e8235a3
Step 5/18 : EXPOSE 9229
 ---> Running in eaaa7e09a880
Removing intermediate container eaaa7e09a880
 ---> 6983c0547b7c
Step 6/18 : WORKDIR "/app"
 ---> Running in 5aa3e020ea3e
Removing intermediate container 5aa3e020ea3e
 ---> 0f0010ae9f0b
Step 7/18 : COPY . /app
 ---> 80da4a6e0073
```

```
Step 8/18 : COPY run-dev /bin
 ---> e21274e4b5bf
Step 9/18 : COPY run-debug /bin
 ---> 698fbf9f87ec
Step 10/18 : RUN chmod 777 /bin/run-dev /bin/run-debug
 ---> Running in 89699eebfca1
Removing intermediate container 89699eebfca1
 ---> d42a3989ed86
Step 11/18 : RUN apt-get update
 ---> Running in 139ee6102b4a
Get:1 http://security.debian.org/debian-security stretch/updates
InRelease [94.3 kB]
Ign:2 http://deb.debian.org/debian stretch InRelease
Get:3 http://deb.debian.org/debian stretch-updates InRelease [91.0 kB]
Get:4 http://deb.debian.org/debian stretch Release [118 kB]
Get:5 http://security.debian.org/debian-security stretch/updates/main
amd64 Packages [492 kB]
Get:6 http://deb.debian.org/debian stretch Release.gpg [2434 B]
Get:7 http://deb.debian.org/debian stretch-updates/main amd64 Packages
[27.2 kB]
Get:8 http://deb.debian.org/debian stretch/main amd64 Packages [7082 kB]
Fetched 7907 kB in 2s (3833 kB/s)
Reading package lists...
Removing intermediate container 139ee6102b4a
 ---> 10108d4a9194
Step 12/18 : RUN apt-get install bc
 ---> Running in ad64618d17c7
Reading package lists...
Building dependency tree...
Reading state information...
The following NEW packages will be installed:
  bc
0 upgraded, 1 newly installed, 0 to remove and 3 not upgraded.
Need to get 105 kB of archives.
After this operation, 238 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian stretch/main amd64 bc amd64
1.06.95-9+b3 [105 kB]
debconf: delaying package configuration, since apt-utils is not installed
Fetched 105 kB in 0s (574 kB/s)
Selecting previously unselected package bc.
(Reading database ... 29980 files and directories currently installed.)
Preparing to unpack .../bc_1.06.95-9+b3_amd64.deb ...
Unpacking bc (1.06.95-9+b3) ...
Setting up bc (1.06.95-9+b3) ...
Removing intermediate container ad64618d17c7
 ---> c80e17207821
Step 13/18 : CMD ["/bin/bash"]
 ---> Running in 5a117eb81023
```

```
Removing intermediate container 5a117eb81023
 ---> 30b80c657c00
Step 14/18 : ARG bx_dev_user=root
 ---> Running in 971caba28b5d
Removing intermediate container 971caba28b5d
 ---> 74e6de6ecbc3
Step 15/18 : ARG bx_dev_userid=1000
 ---> Running in 0d136f3c49a2
Removing intermediate container 0d136f3c49a2
 ---> a3ff0adc83bb
Step 16/18 : RUN BX_DEV_USER=$bx_dev_user
 ---> Running in fc02ff6a2cb5
Removing intermediate container fc02ff6a2cb5
 ---> e149c8dba8d3
Step 17/18 : RUN BX_DEV_USERID=$bx_dev_userid
 ---> Running in ca7ec81199b4
Removing intermediate container ca7ec81199b4
 ---> 3d7827a18e9f
Step 18/18 : RUN if [ "$bx_dev_user" != root ]; then useradd -ms /bin/bash
-u $bx_dev_userid $bx_dev_user; fi
 ---> Running in 1c401a77f4cc
Removing intermediate container 1c401a77f4cc
 ---> cbba7ff52bca
Successfully built cbba7ff52bca
Successfully tagged statusapp-express-tools:latest
OK
Creating a container named 'statusapp-express-tools' from that
image...
OK
Starting the 'statusapp-express-tools' container...
OK
Executing npm install command started at Wed Jun  5 15:13:49 2019
npm
 WARN status-app@1.0.4 No repository field.

audited 635 packages in 1.926s
found 0 vulnerabilities

OK
Process time: 2.455471185s
Stopping the 'statusapp-express-tools' container...
OK
```

__ 3. Display the list the the Docker images:

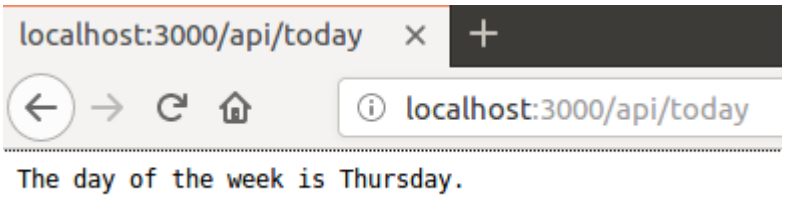
```
$ docker images
REPOSITORY              TAG               IMAGE ID          CREATED
SIZE
statusapp-express-tools   latest              cbba7ff52bca        13 seconds
ago      914MB
node                  8-stretch         c5d36fec051d       24 hours ago
895MB
```

__ 4. Run the application in a Docker container:

```
$ ibmcloud dev run statusapp-express-tools --container-port-map 3000:3000

The run-cmd option was not specified
Stopping the 'statusapp-express-run' container...
The 'statusapp-express-run' container was not found
Validating Docker image name
Binding IP and ports for Docker image.
OK
Checking if Docker container statusapp-express-run is running
OK
Checking Docker image history to see if image already exists
OK
Creating image statusapp-express-run based on Dockerfile ...

Executing docker image build --file Dockerfile --tag
statusapp-express-run --rm --pull .

OK
Creating a container named 'statusapp-express-run' from that
image...
OK
Starting the 'statusapp-express-run' container...
OK
Logs for the statusapp-express-run container:

> status-app@1.0.4 start /app
> node server.js

Listening on port 3000.
```

__ 5. Test the application in a local Docker container:

__ a. In a web browser, open: `http://localhost:3000/api/today`



The application is called on the Docker container and the result is displayed in the browser.

__ 6. Stop the running Docker container:

__ a. In another terminal window, type: `docker ps`

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                    NAMES
4cca0cb8ddef        statusapp-express-run   "docker-entrypoint.s…"   10
minutes ago       Up 10 minutes       0.0.0.0:3000->3000/tcp
statusapp-express-run
```

__ b. Stop the running container with the commands:

```
docker stop <CONTAINER ID>
docker rm <CONTAINER ID>
```

where `<CONTAINER ID>` is the value displayed in the docker ps command.

5.7. Review the process for deploying to the IBM Cloud CLI

Before you package and deploy your application, review the steps and settings for deploying a Node application as an IBM Cloud application.

When you deploy an application to IBM Cloud, the Cloud Foundry command-line interface reads the `manifest.yml` file to configure the Cloud application environment. In this section, review and edit the `manifest.yml` file in the `status-app` directory.

__ 1. Open the Cloud documentation to the command line interface deploy command.

__ a. From a browser session, click https://cloud.ibm.com/docs/cli/idt?topic=cloud-cli-idt-cli#deploy

__ b. The deploy command requires a `manifest.yml` file in the application root directory. This file is generated by the IBM Cloud Developer Tools enable command.

Information

You define the manifest file in the YAML (yet another markup language) format. Specifically, `manifest.yml` follows these naming conventions:

- The manifest file begins with three dashes (`---`).
- The *applications* block begins with a heading followed by a colon (`:`).
- The application *name* starts with a dash (`-`) and a space.
- Subsequent lines begin with two spaces to align with the name field.

__ 2. Review the generated manifest file for the Node.js application.in the `status-app` directory.

__ a. Open a terminal (Mac OS X, Linux) or command prompt (Microsoft Windows).

__ b. Go to the `/projects/status-app` directory.

```
$ cd projects/status-app/
```

__ c. Open the manifest.yml file in an editor.

```
$ gedit manifest.yml
---
applications:
- instances: 1
  timeout: 180
  name: statusapp
  buildpack: sdk-for-nodejs
  command: npm start
  memory: 256M
  domain: not-used.net
  host: not-used
  random-route: true
```

__ d. Add the `random-route: true` statement at the end of the file.

__ e. The name of the application in the manifest file is set to `statusapp`.

__ f. Save and close `manifest.yml`.

Information

The `manifest.yml` file defines deployment properties for the Cloud Foundry push command:

- **Name** defines the Cloud application name.
- **Disk** sets aside the maximum disk space for the application. The default value is 1 GB, or 1024 MB.
- **Memory** allocates server RAM memory to run the application. To conserve quota space, override the default value of 1 GB.
- **Path** defines the location of the Node application. If the value is set to dot (.), the package`.`json file is in the same directory as `manifest.yml`.
- **Random-route**: By default, the web route for an IBM Cloud application is the application **name** and the **`appdomain.cloud`** domain.

__ 3. Define the runtime engine version for node and npm.

__ a. Retrieve the node version in your local workstation.

```
$ node -v
v10.16.0
```

__ b. Retrieve the npm version in your local workstation.

```
$ npm -v
6.9.0
```

__ c. Open package`.`json in a text editor.

__ d. Define two fields with the version numbers: engines`.`node and engines`.`npm.

```
"engines": {
  "node": "10.16.0",
  "npm": "6.9.0"
}
```

__ e. Save and close package`.`json.

Note

The versions of npm and node do not have to match the examples that are given. You might run a newer version of the runtime engines in your workstation.

5.8. Deploy the Node application to your Cloud account

With the `manifest.yml` file complete, you can log in to your Cloud account and deploy the application source code to your Cloud environment. In this part, you deploy the application to the Cloud with Cloud Foundry.

Information

If you sign on to IBM Cloud with a federated ID, review the topic https://cloud.ibm.com/docs/iam/federated_id?topic=iam-federated_id in the IBM Cloud documentation. Create and download an IBM Cloud API key at https://cloud.ibm.com/iam/apikeys

Copy the downloaded API key to the application folder at `projects/status-app`.

The example that follows uses an API key when signing on to the IBM Cloud from the CLI. In the example, the downloaded API key is named `apiKey.json`.

IBM Cloud Foundry organizes resources into organizations and spaces. For more information, see https://cloud.ibm.com/docs/account?topic=account-orgsspacesusers

__ 1. Log in to your Cloud account.

__ a. Open a terminal (Mac OS, Linux) or command prompt.

__ b. With the IBM Cloud command-line interface (CLI), log in to the Cloud US South API endpoint.

```
$ ibmcloud login --apikey @apiKey.json -r us-south
API endpoint: https://cloud.ibm.com
Authenticating...
OK

Targeted account KEVIN O'MAHONY's Account
(4e044ff6259729357f401515e48643f2) <-> 1808327

Targeted resource group default
Targeted region us-south

API endpoint:      https://cloud.ibm.com
Region:            us-south
User:              kevinom@ca.ibm.com
Account:           KEVIN O'MAHONY's Account
(4e044ff6259729357f401515e48643f2) <-> 1808327
Resource group:    default
CF API endpoint:
Org:
Space:
```

__ c. You are signed in to your IBM Cloud account.

__ d. Use the `ibmcloud target --cf` command with prompts to sign on to any Cloud Foundry organization and spaces you define in your Cloud environment.

```
$ ibmcloud target --cf

Targeted Cloud Foundry (https://api.ng.bluemix.net)
Targeted org kevinom_org

Select a space (or press enter to skip):
1. dev
2. kw-smart
3. kv-cloud
Enter a number> 1
Targeted space dev
```

__ e. Review the settings for the Cloud Foundry CLI utility.

```
API endpoint:      https://cloud.ibm.com
Region:            us-south
User:              kevinom@ca.ibm.com
Account:           KEVIN O'MAHONY's Account
(4e044ff6259729357f401515e48643f2) <-> 1808327
Resource group:    default
CF API endpoint:   https://api.ng.bluemix.net (API version: 2.128.0)
Org:               kevinom_org
Space:             dev
```

__ 2. Query any existing applications on your IBM Cloud account.

```
ibmcloud cf apps
Invoking 'cf apps'...

Getting apps in org kevinom_org / space dev as kevinom@ca.ibm.com...
OK

name              requested state   instances   memory   disk   urls
rest-nodesample   started           1/1         256M     1G
rest-nodesample.mybluemix.net
```

__ 3. Deploy the Node application to your IBM Cloud account.

__ a. Go to the `status-app` directory.

```
$ cd projects/status-app
```

__ b. Deploy the application to the logged in account.

```
$ ibmcloud dev deploy
The hostname for this application will be: status-app
? Press [Return] to accept this, or enter a new value now>
Failed to retrieve the app.
The application ID supplied is blank
Unable to load user defined services, proceeding with deployment
Deploying to Cloud Foundry...

Executing ibmcloud cf push


Invoking 'cf push'...

Pushing from manifest to org kevinom_org / space dev as
kevinom@ca.ibm.com...
Using manifest file /home/localuser/projects/status-app/manifest.yml
Getting app info...
Creating app with these attributes...
+ name:                   statusapp
  path:                   /home/localuser/projects/status-app
  buildpacks:
+   sdk-for-nodejs
+ command:                npm start
+ health check timeout:   180
+ instances:              1
+ memory:                 256M
  routes:
+   status-app.us-south.cf.appdomain.cloud

Creating app statusapp...
Mapping routes...
Comparing local files to remote cache...
Packaging files to upload...
Uploading files...
 6.28 MiB / 6.28 MiB [=============================================]
100.00% 3s

Waiting for API to complete processing files...

Staging app and tracing logs...
   Downloading sdk-for-nodejs...
   Downloaded sdk-for-nodejs
   Cell ed57f560-073e-48e1-a5b8-e42888f07850 creating container for
instance 2312c7da-d42a-4b10-b052-9966e9cbda43
   Cell ed57f560-073e-48e1-a5b8-e42888f07850 successfully created
container for instance 2312c7da-d42a-4b10-b052-9966e9cbda43
   Downloading app package...
```

```
   Downloaded app package (8.1M)
   -----> IBM SDK for Node.js Buildpack v3.26-20190313-1440
          Based on Cloud Foundry Node.js Buildpack v1.5.24
   -----> Creating runtime environment

          NPM_CONFIG_LOGLEVEL=error
          NPM_CONFIG_PRODUCTION=true
          NODE_ENV=production
          NODE_MODULES_CACHE=true
   -----> Installing binaries
          engines.node (package.json):  10.16.0
          engines.npm (package.json):   6.9.0

          Downloading and installing node 10.16.0...
          npm 6.9.0 already installed with node
   -----> Restoring cache
          Skipping cache restore (new runtime signature)
   -----> Building dependencies
          Installing node modules (package.json)
         added 97 packages from 134 contributors and audited 635 packages
in 7.902s
          found 0 vulnerabilities

   -----> Installing App Management
   Checking for Dynatrace credentials
   No Dynatrace Service Found (service with substring dynatrace not found
in VCAP_SERVICES)
   -----> Caching build
          Clearing previous node cache
          Saving 2 cacheDirectories (default):
          - node_modules
          - bower_components (nothing to cache)
   -----> Build succeeded!
          +-- express@4.17.1
          +-- request@2.88.0
          +-- xml2js@0.4.19

   Exit status 0
   Uploading droplet, build artifacts cache...
   Uploading droplet...
   Uploading build artifacts cache...
   Uploaded build artifacts cache (1.5M)
   Uploaded droplet (29.5M)
   Uploading complete
   Cell ed57f560-073e-48e1-a5b8-e42888f07850 stopping instance
2312c7da-d42a-4b10-b052-9966e9cbda43
   Cell ed57f560-073e-48e1-a5b8-e42888f07850 destroying container for
instance 2312c7da-d42a-4b10-b052-9966e9cbda43
```

```
Waiting for app to start...
```

Troubleshooting

The Cloud platform attempts to start your status-app application, but it never starts successfully. Press Ctrl+C to cancel the application deployment and examine your files.

__ 4. Review the status-app application runtime log files.

__ a. In the terminal (Linux, Mac OS) or command prompt (Microsoft Windows), display the most recent log entries from the status-app application.

```
$ ibmcloud cf logs statusapp --recent
```

__ b. Examine the log output.

```
[STG/0]   OUT Uploading complete
[APP/0]   OUT
[APP/PROC/WEB/0] OUT > status-app@1.0.4 start /home/vcap/app
[APP/PROC/WEB/0] OUT > node server.js
[APP/PROC/WEB/0] OUT Listening on port 3000.
[HEALTH/0] ERR Failed to make TCP connection to port 8080: connection
refused
[CELL/0] ERR Timed out after 3m0s: health check never passed.
```

Questions

Why did the Node application fail to start?

According to the application runtime log, the Node application waits for incoming connections at port 3000. However, the Cloud App Management feature cannot connect to the Node application.

__ 5. Review the Cloud documentation for the node runtime environment.

__ a. Open http://docs.cloudfoundry.org/buildpacks/node/node-tips.html#port.

__ b. Review the **Application Port** section.

Application Port

You must use the PORT environment variable to determine which port your app should listen on. To also run your app locally, set the default port as `3000`.

```
app.listen(process.env.PORT || 3000);
```

Information

The `status-app` Node application listens to HTTP requests on port `3000`. However, the Cloud application runtime forwards HTTP requests on a non-standard port number. You must modify your application code to listen to the `process.env.PORT` port for incoming requests.

__ 6. Update the **`port`** variable in the status-app **`server.js`** code.

__ a. Open `server.js` in a text editor.

__ b. Assign the **`port`** variable with the value in the **`process.env.PORT`** variable. If it does not exist, set the port value to **`3000`**.

```
var port = process.env.PORT || 3000;
```

__ c. Save and close `server.js`.

__ 7. Deploy and build the application to your Cloud account.

__ a. Open a terminal (Linux, Mac OS) or command prompt (Microsoft Windows).

__ b. Verify that no coding issues occur in `server.js` with ESLint.

```
$ eslint server.js
```

Information

You get an ESLint error “Unexpected use of process.env”. To fix this error, edit the file `.eslintrc.js` and set the option:

```
"no-process-env": "off"
```

__ 8. Redeploy the application to your Cloud account.

```
$ ibmcloud dev deploy
```

__ c. Examine the terminal console after the build process on the server.

```
Waiting for app to start...

name:              statusapp
requested state:   started
routes:            status-app.us-south.cf.appdomain.cloud
last uploaded:     Thu 06 Jun 15:36:53 PDT 2019
stack:             cflinuxfs3
buildpacks:        sdk-for-nodejs

type:            web
instances:       1/1
memory usage:    256M
start command:   npm start
     state     since                  cpu    memory        disk
details
#0   running   2019-06-06T22:37:36Z   0.5%   36K of 256M   135.3M of 1G


OK
Your app is hosted at http://status-app.us-south.cf.appdomain.cloud/
```

Information

The health and status of the application lists the state of your Cloud application.

- **Instances** describe how many copies of the status-app are running in your Cloud account.
- **routes** list the web route that is bound to your application.
- **stack** describes the name of the thin Linux operating system that is running the Node.js buildpack.
- The **CPU**, **memory**, and **disk** list the resource usage of your application. The `manifest.yml` file and your service quota define the limits of these resources.

__ 9. Review the route for the status application.

__ a. Issue `ibmcloud cf routes` in the terminal or command prompt.

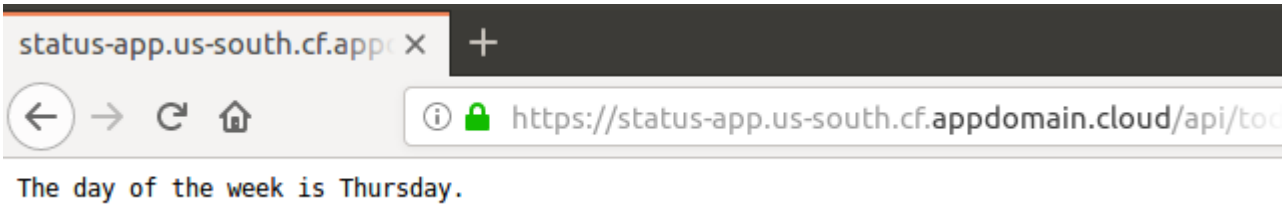
```
$ ibmcloud cf routes
Invoking 'cf routes'...

Getting routes for org kevinom_org / space dev as kevinom@ca.ibm.com ...

space   host            domain                      port   path   type
apps              service
dev     status-app      us-south.cf.appdomain.cloud
statusapp
```

Information

Use the `cf routes` command to find the host name for your Node application that is running on your Cloud account. The web route to your application is `http://<host>.us-south.cf.appdomain.cloud`, where *<host>* is the value that is listed in the `cf routes` command for the `status-app` application.

__ 10. Run the status-app application in the IBM Cloud environment.

__ a. In a web browser, test the GET `/api/today` API operation. Type `https://status-app.us-south.cf.appdomain.cloud/api/today` in the browser address area.



The result is displayed in the browser.

__ 11. View the server log for the statusapp application.

__ a. Issue the `ibmcloud cf logs` command for the `statusapp` application.

__ b. `$ ibmcloud` **`cf logs statusapp --recent`**

__ c. Examine the log output.

```
2019-06-06T15:38:08.26-0700 [RTR/22] OUT
status-app.us-south.cf.appdomain.cloud - [2019-06-06T22:38:08.200+0000]
"GET /api/today HTTP/1.1" 200 0 32 "-" "Mozilla/5.0 (X11; Ubuntu; Linux
x86_64; rv:58.0) Gecko/20100101 Firefox/58.0" "10.186.136.34:59320"
"169.61.179.199:61160" x_forwarded_for:"216.232.198.238, 10.186.136.34"
x_forwarded_proto:"https"
vcap_request_id:"9d02d2a9-4f16-43dc-40d7-3f72b76b7be9"
response_time:0.067922021 app_id:"08a4097e-c754-4f77-ba5f-a2386c0698c7"
app_index:"0" x_global_transaction_id:"9f474cea5cf995d0de5fd8b9"
true_client_ip:"-" x_b3_traceid:"f0499677cc1dfcf9"
x_b3_spanid:"f0499677cc1dfcf9" x_b3_parentspanid:"-"
b3:"f0499677cc1dfcf9-f0499677cc1dfcf9"
```

Information

The application log displays the HTTP traffic from the `statusapp` application. In this example, the Cloud router logs an HTTP GET request to the `/api/today` web route. The rest of the log entry captures the HTTP request headers.

The application log also displays console logs from the Node application. The `ibmcloud cf logs` command is invaluable in troubleshooting runtime errors in a deployed Cloud application. Use the log to verify that your API operations work properly.

The deploy process updates the local `manifest.yml` file with the actual route that is assigned to the application on the IBM Cloud.

__ 12. Review the updated manifest.yml file.

__ a. Open `manifest.yml` in a text editor.

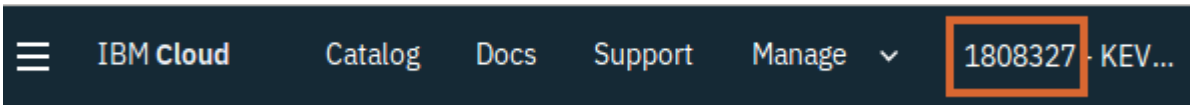
```
applications:
- buildpack: sdk-for-nodejs
  command: npm start
  domain: null
  host: null
  instances: 1
  memory: 256M
  name: statusapp
  random-route: true
  routes:
  - route: status-app.us-south.cf.appdomain.cloud
  timeout: 180
```

__ b. Notice that the route now contains the route value that is assigned for the application on the IBM Cloud.

Information

If you want to pick a unique route name for the application in the IBM Cloud yourself, you can edit the file manifest.yml and change the route value.

For example, change the `route` prefix to `status-app` and append the digits that are displayed for your IBM Cloud account when you sign on to the IBM Cloud with a web browser.



You can change the route to avoid a name collision with another application that uses the same route URL on IBM Cloud. For example:

```
routes:
  - route: status-app-1808327
us-south.cf.appdomain.cloud
```

__ 13. Log out from the IBM Cloud from the CLI.

```
$ ibmcloud logout
```

5.9. Review the Node application with the Cloud dashboard

The Cloud dashboard provides the same information as the Cloud Foundry command-line interface (CLI) utility through a web application. In this section, review the settings and status of the status-app through the dashboard.

__ 1. Open the Compute category in the Cloud dashboard.

__ a. In a web browser, open https://cloud.ibm.com.

__ b. Log in with your Cloud ID and password.

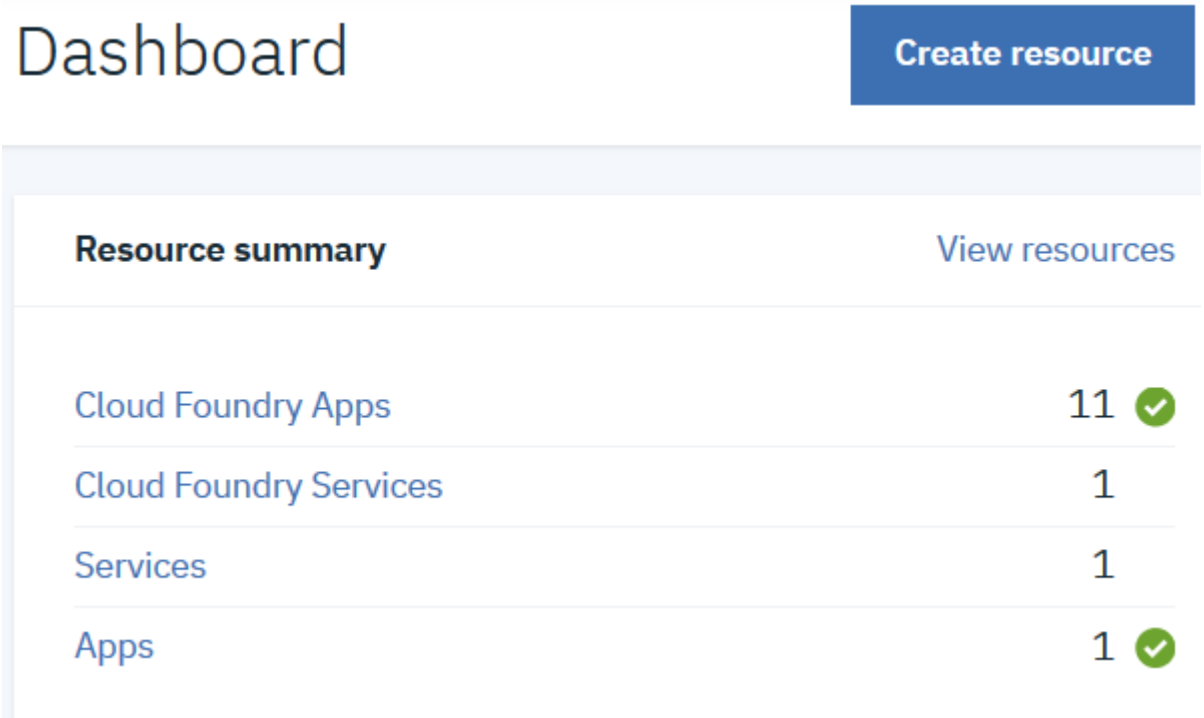
__ c. The Cloud Dashboard is displayed.

Note

If the Dashboard page is not displayed, select the menu icon in the upper-left corner of the page.

Then select Dashboard.

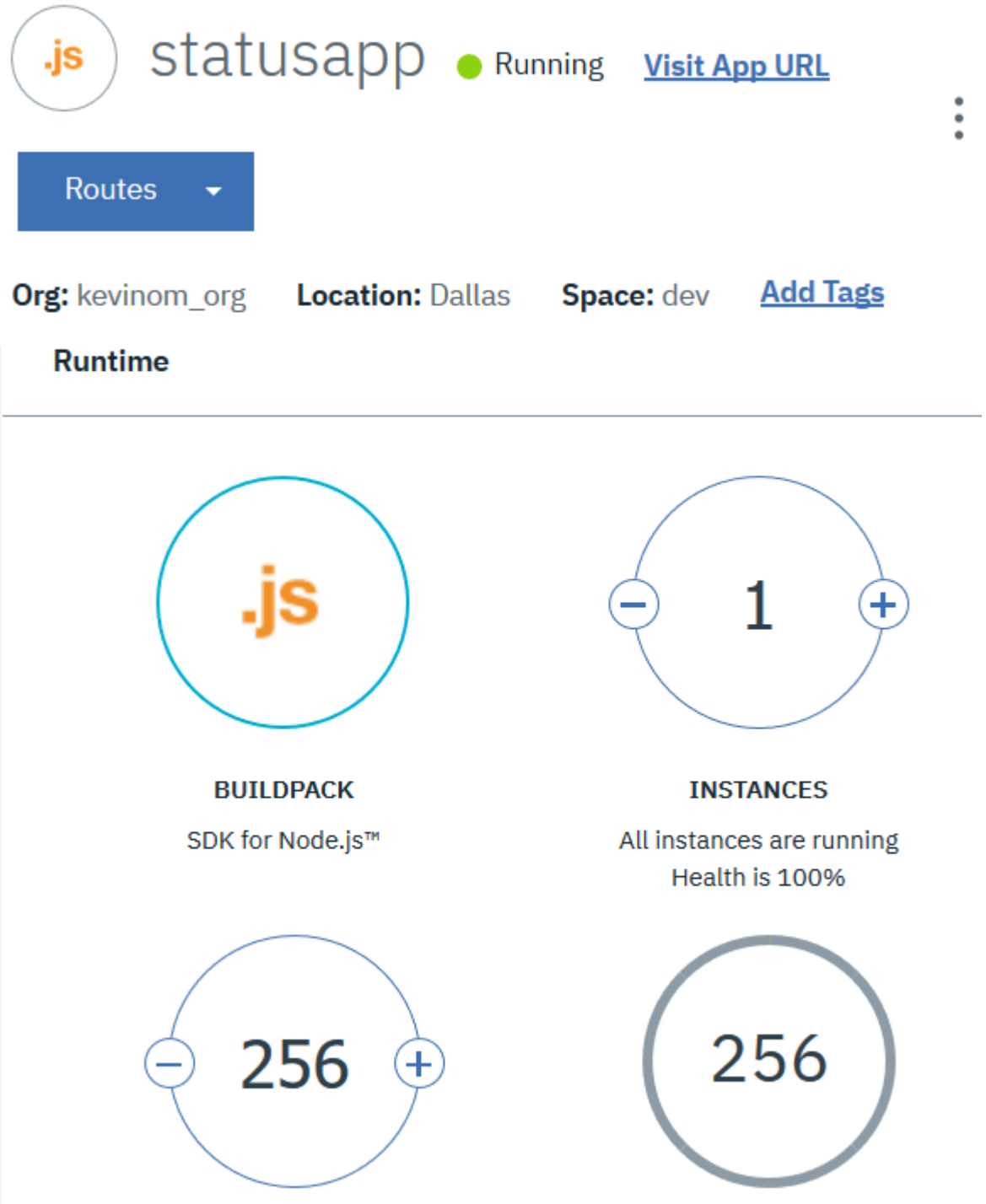
__ d. From the Dashboard page, select the **Cloud Foundry Apps** category from the list of resources.



__ 2. Examine the state of the `statusapp` Node application.

__ a. Select the **statusapp** application in the list.

__ b. Review the runtime status and resource usage.



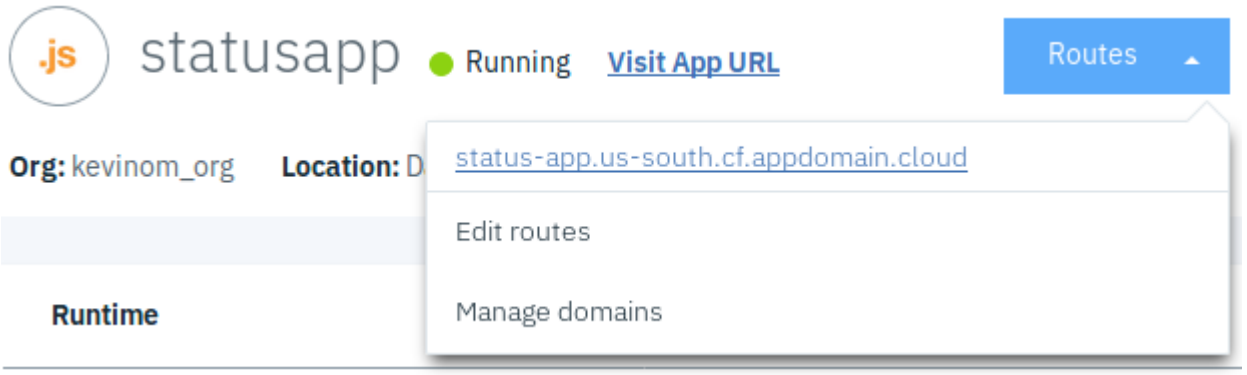
Information

The **Cloud application dashboard** displays the same pertinent information that you reviewed in the IBM Cloud Foundry command-line interface (ibmcloud cf).

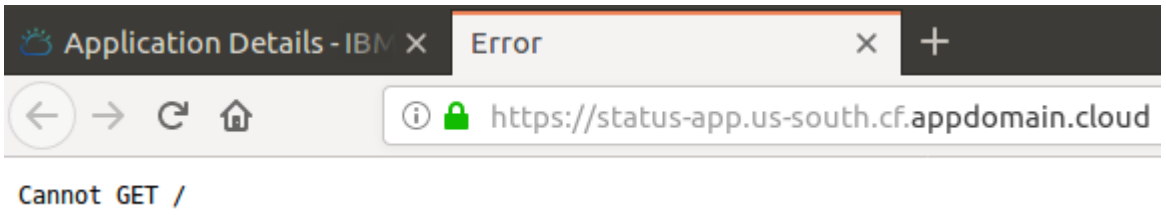
- The **buildpack** lists the runtime environment for your application. You set the exact Node.js runtime version in the **package.`json` engines** section.
- The **instances** section displays the number of processes and status of the processes.
- The **MB per instance** setting displays the memory space for the application.
- The **Total MB allocation** displays the total amount of space available to all instances of the application.

__ 3. Run the application from your application page.

__ a. Click the **Routes** icon from the application. Then, click the Route URL to open the page.

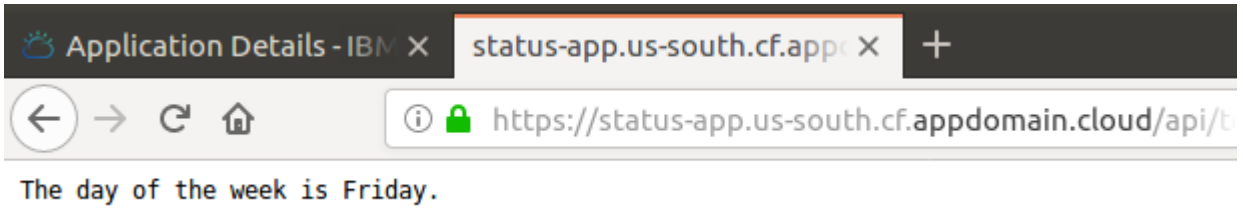


__ b. A page is displayed with the message "`Cannot GET /`".

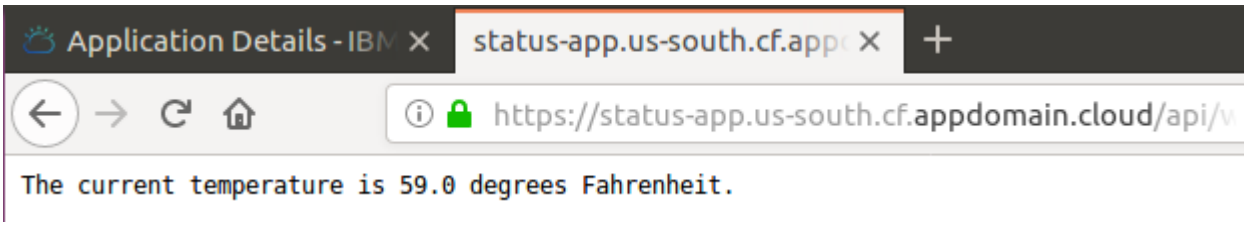


__ c. Append the value '`/api/today`' at the end of the route in the browser address area.

__ d. The page returns a successful call to the application.

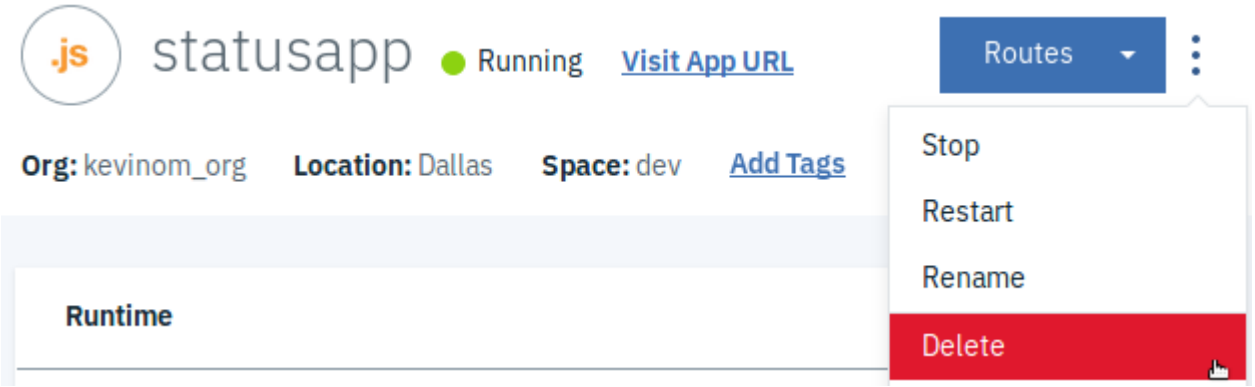


__ e. Instead of the value '`/api/today`' at the end ot the route, append the value '`api/weather/KSEA`' to the end of the route in the browser address area. The page returns a successful call to the weather application.

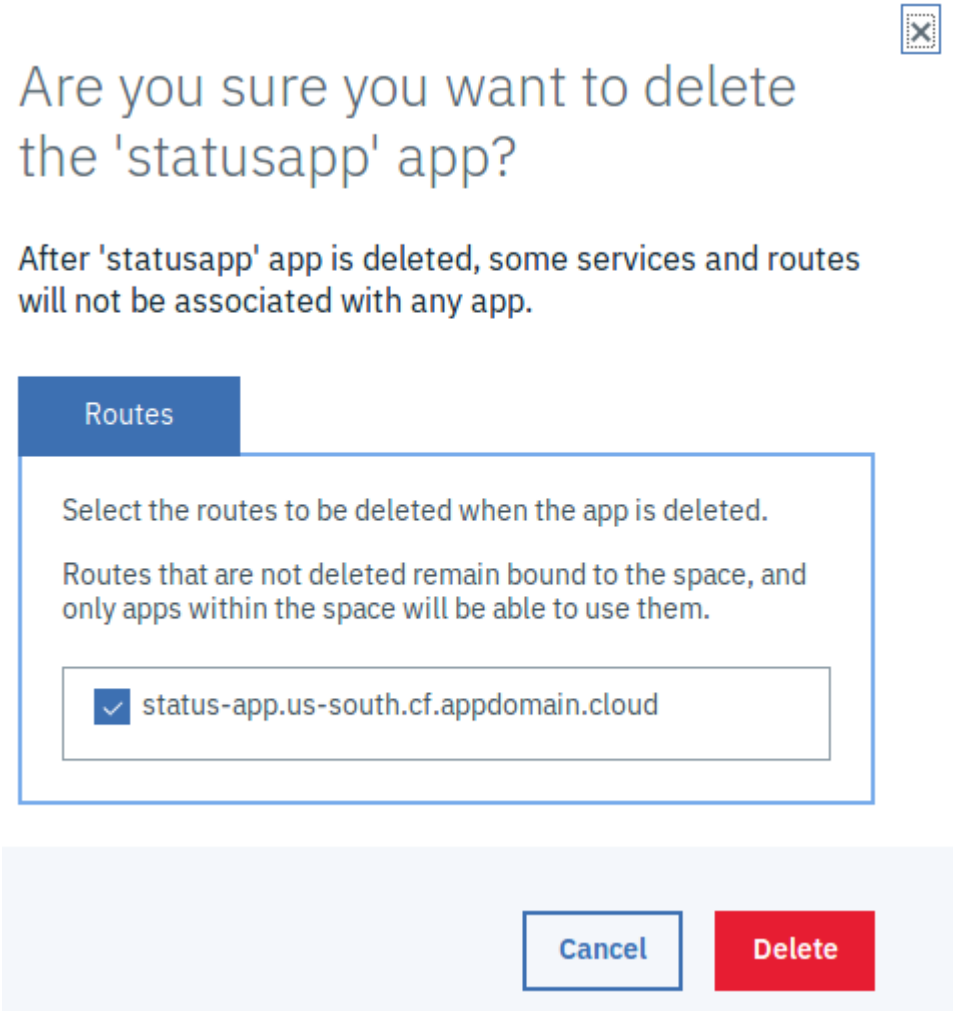


__ 4. Delete the application in IBM Cloud when you are finished.

__ a. From the application view in IBM Cloud, select the options ellipsis. Then, select **Delete**.



__ b. Select the route to be deleted in the confirmation dialog.



Then, click **Delete**.

__ c. The application and route are deleted from the Cloud account.

__ 5. Log out of the Cloud dashboard.

End of exercise

Solution

`status-app/manifest.yml` generated by the enable step:

```
---
applications:
- instances: 1
  timeout: 180
  name: statusapp
  buildpack: sdk-for-nodejs
  command: npm start
  memory: 256M
  domain: not-used.net
  host: not-used
```

`status-app/manifest.yml` updated by the deploy step:

```
---
applications:
- buildpack: sdk-for-nodejs
  command: npm start
  domain: null
  host: null
  instances: 1
  memory: 256M
  name: statusapp
  random-route: true
  routes:
  - route: status-app.us-south.cf.appdomain.cloud
  timeout: 180
```

status-app/package.json:

```
{
  "name": "status-app",
  "version": "1.0.4",
  "description": "Return Node runtime status information",
  "main": "server.js",
  "scripts": {
    "lint": "echo 'Running ESLint now' && eslint *.js",
    "start": "node server.js",
    "test": "mocha test",
    "build": "echo 'Running build' && npm run lint && npm run test"
  },
  "author": "John Doe <jdoe@example.com>",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "request": "^2.88.0",
    "xml2js": "^0.4.19"
  },
  "engines": {
    "node": "10.16.0",
    "npm": "6.9.0"
  },
  "devDependencies": {
    "chai": "^4.2.0",
    "eslint": "^5.16.0",
    "mocha": "^6.1.4",
    "supertest": "^4.0.2"
  }
}
```

status-app/today.js:

```
/*eslint no-undef: "error"*/
/*eslint-env node*/
module.exports = function(date) {
  var days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday'];
  return days[ date.getDay() ];
}
```

status-app/server.js:

```
/*eslint no-undef: "error"*/
/*eslint-env node*/
var port = process.env.PORT || 3000;
var parse = require('xml2js').parseString;
var today = require('./today');
var request = require('request');
var express = require('express');
var app = express();

app.get('/api/today', function(req, res) {
  var date = req.query.date == null ? new Date() : new Date(req.query.date);
  var body = "The day of the week is "+ today(date)+ ".";
  res.type('text/plain');
  res.set('Content-Length', Buffer.byteLength(body));
  res.status(200).send(body);
});

app.get('/api/weather/:location', function(req, res) {
  var options = {
    method: 'GET',
    uri: 'http://weather.gov/xml/current_obs/'
      + req.params.location + '.xml',
    headers: {
      'User-agent': 'weatherRequest/1.0'
    }
  };
  var callback = function(error, response, body) {
  if (error) {
    res.status(500).send(error.message);
  }
  parse(body, function(err, result) {
    var message =
      'The current temperature is ' +
      result.current_observation.temp_f[0] +
      ' degrees Fahrenheit.';
    res.type('text/plain');
    res.set('Content-Length', Buffer.byteLength(message));
    res.status(response.statusCode).send(message);
  });
};
  request(options, callback);
});

app.listen(port, function() {
  console.log('Listening on port %s.', port);
});
```

Exercise review and wrap-up

In this exercise, you installed the IBM Cloud Developer Tools. You used the BM Cloud Developer Tools command to enable the application for IBM Cloud and generate an application manifest.yml file to configure the Cloud application environment. You used the Cloud deploy command to push the application as a Cloud Foundry application to IBM Cloud. You reviewed the log files from the command-line interface. Then, you changed the server.js file to include the Cloud process environment in the port variable. You redeployed the application to Cloud and the application ran on the IBM Cloud.

IBM Training