ELECTRICAL AND INFORMATION ENGINEERING
University of the Witwatersrand, Johannesburg
Software Development III

Laboratory 4 — Testing and Deployment

In this lab, we will implement testing for the class list app from Lab 3 and deploy it to Azure using Travis which is a Continuous Integration (CI) service.

1 Testing

In order to ensure that we don't break or regress features as we code, it is important to test our code regularly. There are many testing frameworks for Javascript and Node. A simple test framework is given below.

1.1 Jest

Jest is a very simple, easy to use testing framework for Node. We will be using Jest for writing tests because of its BDD (Behaviour Driven Development) assertion syntax.

BDD-style assertions: In C++, using doctest, you were familiarised with TDD (Test Driven Development) and the assertion format of:

```
assert.equals(a, "b")
```

while functional, these assertions can be difficult to read. BDD aims to solve this by using verbose expression syntax. Thus, the same assertion in BDD-style is:

```
expect(a).toEqual("b")
```

In addition to its simple syntax, Jest is useful because it allow for easy generation of code coverage reports. To get code coverage reports, just add the flag `--coverage` when running your tests.

Install Jest as development dependancies:

```
npm install --save-dev jest
```

Now let's write our first test. Create a folder called `test` and, in it, a file called `classList.test.js` with the following:

```
'use strict'

test('Hello World: hello should greet the world', () => {
  let hello = 'world'
  expect(hello).toEqual('world')
})
```

Note: All test files must have the `.test` term in their name

A bit of a contrived test, but let's see if it works. In your `package.json`, change the value of `scripts.test` to `"jest"`. Now run

```
npm run test
```

You should have 1/1 tests passing!

Ok, now let's create something that we can really test. It's good practice to abstract your data storage from the rest of your code with a proxy class so that your business logic does not become dependant on your specific database or version. We are going to do this with our class list app.

At the moment, the class list is just stored in an array at the top of the router file. We are going to create a module that presents an interface for adding, deleting, editing and viewing students from the class list.

Create a file called `classList.js` with the following in it:

```
// Private
var list = []

// Public
module.exports = {
  add: function (student) {
    list.push(student)
  },
  edit: function (student, index) {
    list[index] = student
  },
  get: function (index) {
    return list[index]
  },
  delete: function (index) {
    list.splice(index, 1) // remove one element starting from index
  }
}
```

Now edit `classRoutes.js` to require and use your new module instead of the array. Add functions to `classList.js` if necessary.

Exercise 1

Using the documentation for Jest, implement some real tests for your newly-created module. **Jest cheatsheet:** https://devhints.io/jest

`Majestic` is a GUI for Jest, that makes visualising and running tests very easy. If you want to use it, it can be found at https://github.com/Raathigesh/majestic

1.2 Tape

An alternative to Jest is Tape (https://github.com/substack/tape).

Tape tests are very simple. An example of the same Hello World test given above but with Tape is given below.

```
"use strict";

let test = require("tape")

test("Hello World: hello should greet the world", function(t){
    let hello = "world"
    t.equal(hello, "world")
    t.end() // This is necessary at the end of every test function
})
```

To use Tape you need to first install it, as shown below:

```
npm install --save-dev tape
```

One other change is to replace the test script in your `package.json` file to the following.

```
"test": "node ./test/*.js",
```

Now you can run your tests using `npm test`. Navigate to https://github.com/dwyl/learn-tape for a very good example of how to use Tape to test a simple program. You can add linting checks to your tests using the `tape-standard` NPM package. Check it out at https://www.npmjs.com/package/tape-standard.

2 Deployment

Manually watching your git repository for changes, running tests and then deploying to your production server is time-consuming. Instead most companies opt to have an automated deployment pipeline.

We are going to set up our own automated deployment platform using Travis CI and Azure.

Firstly, ensure that you have a student account with both Travis CI and Azure. The best method for getting these is using the GitHub Student Pack (https://education.github.com/pack).

3 Travis CI

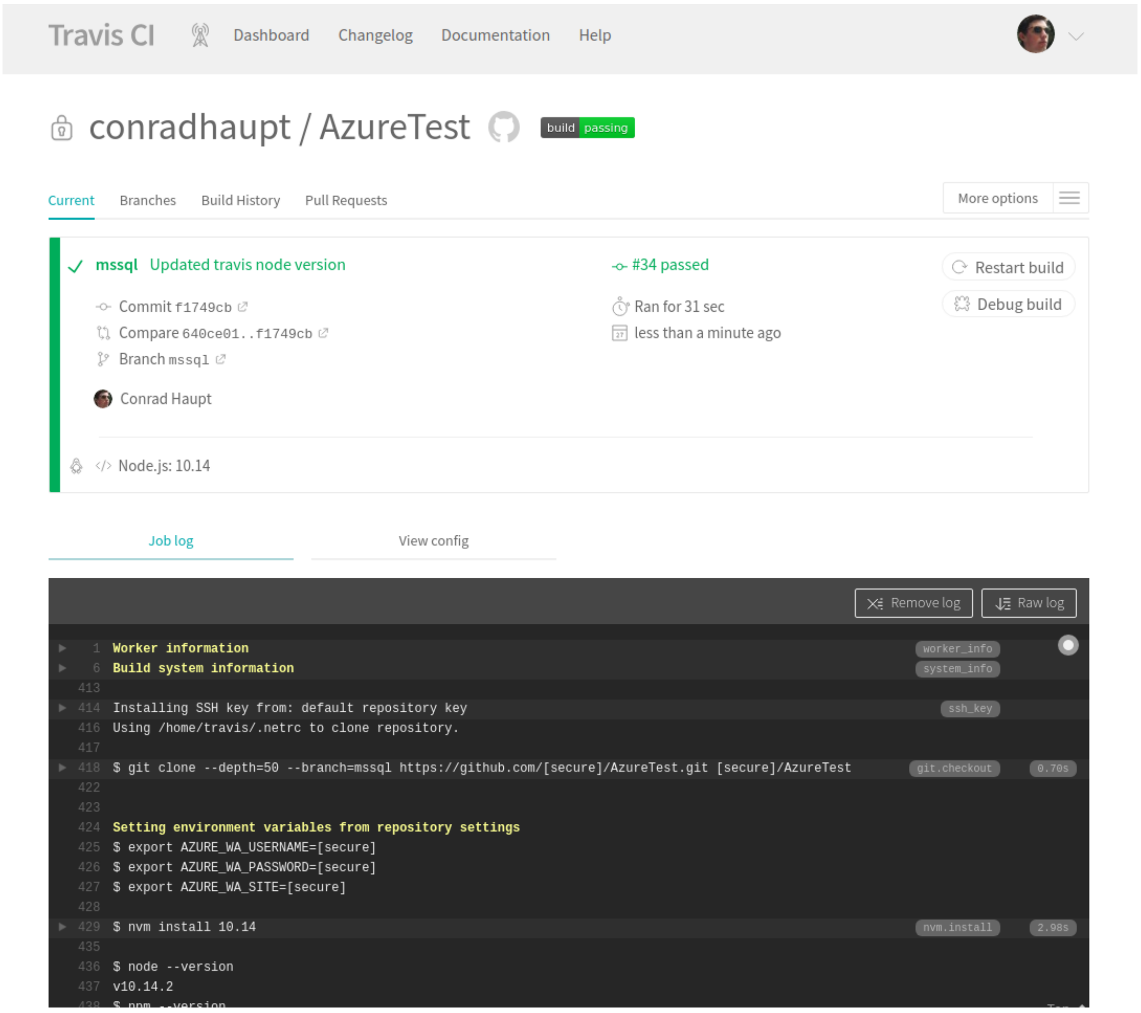
Navigate to `https://travis-ci.com` and register using your Github account. You should be redirected to Github.com where you should follow the prompts.

Now add a file called `.travis.yml` with the following to the root of your git repository:

```yaml
language: node_js
node_js:
  - "10.14"
```

Commit and push the file to your repository.

If you then open Travis CI, you should see your project being built:



mssql Updated travis node version

#34 passed

Commit f1749cb

Compare 640ce01..f1749cb

Branch mssql

Conrad Haupt

Ran for 31 sec

less than a minute ago

Node.js: 10.14

```
Worker information                                                              worker_info
Build system information                                                        system_info

Installing SSH key from: default repository key                                 ssh_key
Using /home/travis/.netrc to clone repository.

$ git clone --depth=50 --branch=mssql https://github.com/[secure]/AzureTest.git [secure]/AzureTest    git.checkout    0.70s


Setting environment variables from repository settings
$ export AZURE_WA_USERNAME=[secure]
$ export AZURE_WA_PASSWORD=[secure]
$ export AZURE_WA_SITE=[secure]

$ nvm install 10.14                                                             nvm.install    2.98s

$ node --version
v10.14.2
$ npm --version
```

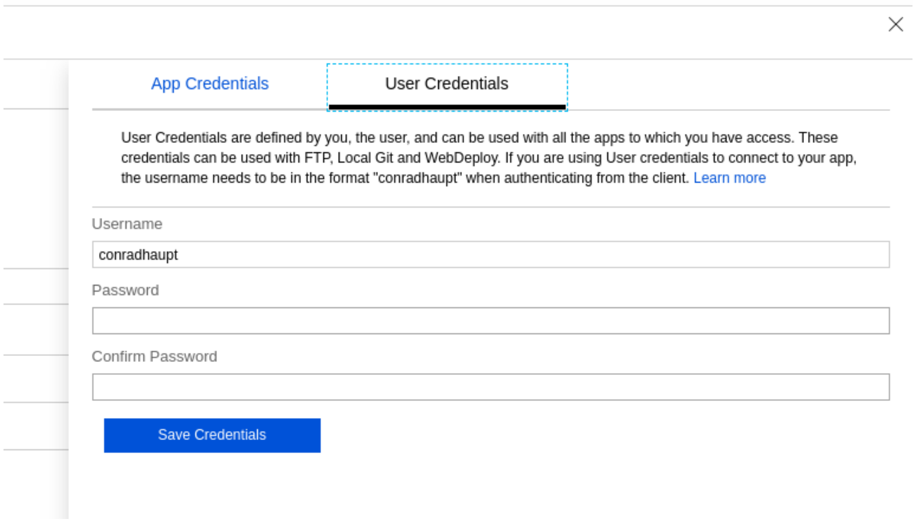
Figure 1: A successful build in Travis CI

4 Azure Deployment from Travis CI

Right now your Azure setup will deploy any new commit on the master branch while Travis CI checks all tests independently. What you really want is for deployments to occur IF AND

ONLY IF the most recent commit has passed all tests successfully. To do this we need to move the automatic deployment functionality from Azure to Travis CI.

Fortunately Travis CI has a built in provider for Azure deployment. To configure and enable it, first you must remove the old Github Deployment Option on your Azure instance. Navigate to your website's configuration page and then to the `Deployment Centre` sub-menu. `Disconnect` your current config and click `Setup`. In the new window select `Local Git Repository` as the new `source` then click `OK` until the deployment change is complete. Once you're back at the main portal window for your website, select `Deployment Credentials`. Select `User Credentials` at the top of the sidebar and fill in `UNIQUE` details for the username and password.



User Credentials are defined by you, the user, and can be used with all the apps to which you have access. These credentials can be used with FTP, Local Git and WebDeploy. If you are using User credentials to connect to your app, the username needs to be in the format "conradhaupt" when authenticating from the client. Learn more

Username: conradhaupt

Password:

Confirm Password:

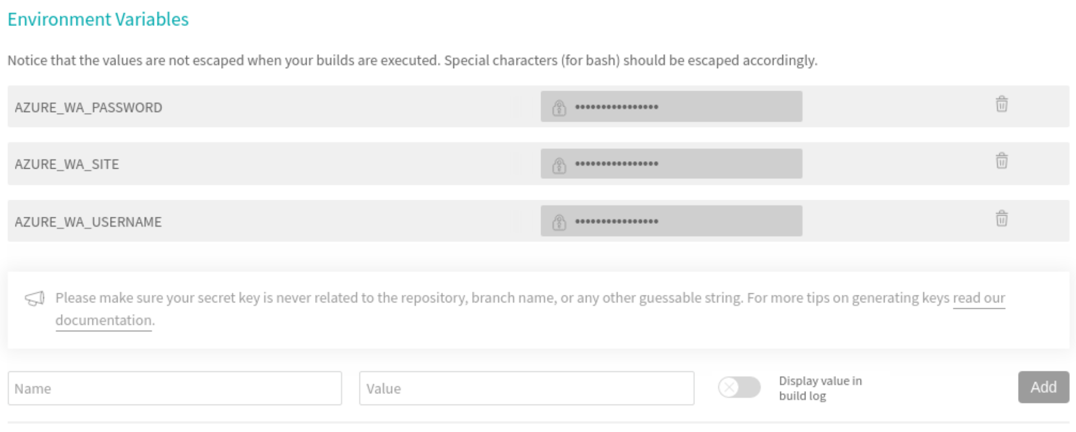
Figure 2: Example App Credentials for Azure Deployment

Storing authentication information in Git and Github is very insecure. Travis CI allows us to store environment variables in the build information to ensure we don't have any security risks with our code. Navigate to your Travis CI page for your Github project and click on `More Options` and then `Settings`. In the window that appears, scroll down to the section titled `Environment Variables`. Add the environment variables listed in the table below to Travis CI. Ensure that the variable names are in all-caps.

Table 1: Travis CI Environment Variables necessary for Azure Deployment

Variable Name	Variable Value
`AZURE_WA_USERNAME`	The username you setup in the deployment credentials page
`AZURE_WA_PASSWORD`	The password you setup in the deployment credentials page
`AZURE_WA_SITE`	The name of your Azure project

After adding those three environment variables, your settings page should look like the figure below.


Environment Variables

Notice that the values are not escaped when your builds are executed. Special characters (for bash) should be escaped accordingly.

AZURE_WA_PASSWORD	••••••••••••••••
AZURE_WA_SITE	••••••••••••••••
AZURE_WA_USERNAME	••••••••••••••••

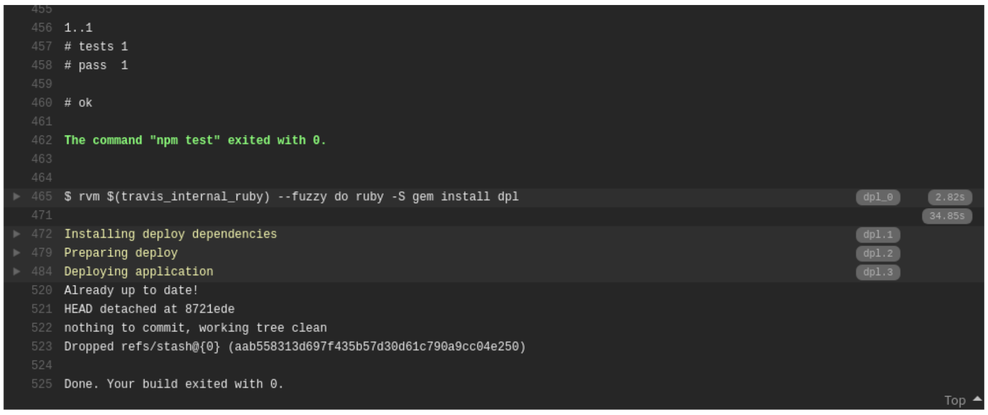
Please make sure your secret key is never related to the repository, branch name, or any other guessable string. For more tips on generating keys read our documentation.

Figure 3: Travis CI Azure Deployment Environment Variables settings section

The final change to make is to add the following code to the end of your `.travis.yml` file in your repo, commit, and push to Github.

```
deploy:
    provider: azure_web_apps
```

After Travis CI has run the tests for your new commit, you should see something similar to the following output in the `Job Log`. If you navigate to your Azure Instance's URL you will be viewing an automatically deployed version of your repository that is controlled by Travis CI.



```
1..1
# tests 1
# pass  1

# ok

The command "npm test" exited with 0.


$ rvm $(travis_internal_ruby) --fuzzy do ruby -S gem install dpl                dpl_0    2.82s
                                                                                        34.85s
Installing deploy dependencies                                                  dpl.1
Preparing deploy                                                                dpl.2
Deploying application                                                           dpl.3
Already up to date!
HEAD detached at 8721ede
nothing to commit, working tree clean
Dropped refs/stash@{0} (aab558313d697f435b57d30d61c790a9cc04e250)

Done. Your build exited with 0.
```

Figure 4: Travis CI Job Output for Azure Deployment

5 Browser LocalStorage

With new developments in browsers and the internet at large, websites can start to take advantage of device capabilities that weren't available 5 years ago. An example is `LocalStorage`. Browsers can now store information and data that is needed by the website without having to deal with cookies or server-side storage.

Lets say that your website needs to store a variable that describes the theme of your website for each user. You have three options:

- Require the user to change the theme every time they navigate to your website.
- Store a session cookie on the user's browser and the matching theming variable on a server-side database. The website would then be themed by the server using the cookie value.
- Store the theming variable in LocalStorage on the user's browser, requiring only javascript on the client-side.

The third option is the most modern and advised for new websites. Obviously, older browser version may not support this but most do. To check if a client has LocalStorage capabilities, you can run the code below.

```
if(typeof(Storage) !== "undefined"){
// LocalStorage is available
} else {
// LocalStorage is NOT available
}
```

If the client's browser supports it, then you can do fancy things like storing key-value pairs.

```
// The user would like the website to use the dark theme
window.localStorage.setItem("theme", "dark")

console.log(window.localStorage.getItem("theme"))
```

You can also use named member variables of localStorage.

```
window.localStorage.theme = "dark"

console.log(window.localStorage.theme)
```

You can delete entries in LocalStorage using `window.localStorage.removeItem("theme")`. The life of the variables stored in LocalStorage is dependent on the browser but they will last for longer than the tab is open.

If you want the variable to be deleted when the tab is closed, replace `localStorage` with `sessionStorage` in the above examples. This way, the variables will be deleted once the browser stops navigating to your website. You can verify if a variable exists by writing `if(window.localStorage.<variable name>)`. The condition will return true if the variable exists and false if it doesn't.