Workshop

Hackerlog: Build and Deploy Node.js Apps with Microsoft Azure

v.0.0.1

Welcome to MLH Localhost:
Build and Deploy Node.js Apps with Microsoft Azure

Wifi Network:
[Network]

Wifi Password:
[Password]

Event Hashtag:
#MLHLocalhost

Twitter Handle:
@MLHacks

Welcome! My name is Kyle O'Brien

1. I'm here to lead this session & help you learn something new today!
2. I'm a Junior at UC Santa Cruz.
3. My favorite programming language / tool is Docker.

1

Using your Web Browser,
Open this URL & Fill out the Form:

http://mlhlocal.host/checkin

2

Afterwards, Check your Email to Find:

- Setup Instructions
- An Invite to the MLH Slack
- The Code Samples
- A Workshop FAQ
- These Workshop Slides
- More Learning Resources

MLH

MAJOR LEAGUE HACKING

Our Mission is to Empower Hackers.

65,000+
HACKERS

12,000+
PROJECTS CREATED

3,000+
SCHOOLS

We hope you learn something awesome today!

Find more resources: http://mlh.io/

What will you learn today?

1. The basics of Node.js and JavaScript Development
2. How to use Visual Studio Code with Git
3. How to Deploy a Web App with Microsoft Azure

Table of Contents

What's Web Development?

Web developers design, build, and maintain websites.
Two main categories:

Front End, or Client-Side:

- Directly user-interactive website parts
- Technologies
  - HTML: Content
  - CSS: Layout
  - Javascript: Interaction

Back End, or Server-Side:

- User-hidden functionality.
- Basically any language, but mostly Node.js, Ruby and Python

Note:

Full Stack developer: Someone who is familiar with both kinds of development, and can work on a website from start to finish!

What You'll Be Building

http://mlhlocal.host/azure-webapp

How It Works

1. User types app URL in browser
2. Browser sends request to app's server, which is hosted on Microsoft Azure
3. Server returns all client-side files for app and browser displays app to user
4. User types update and password
5. Browser sends information back to app
6. Server checks password and adds updates to database
7. Server updates website with changes now visible to users

Client Side
(Front End)

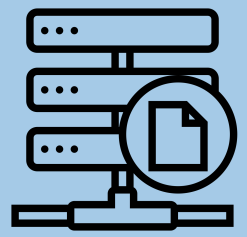
Server Side
(Back End)

Database

Client Side (Front-End)

1. User types app URL
2. Server returns app pages to display
3. User enters password & updates
4. Server checks password and adds updates to DB
5. Server updates the website with new changes

Server Side (Back-End)



Database

How You'll Build It

We'll be using several tools to make building this app as easy as possible.

- Node.js
- Express.js
- MongoDB and Mongoose
- Microsoft Azure
- Visual Studio Code
- Git

Now that you understand how the app works, let's get started!

We learned that our app will be hosted on Microsoft Azure. Why are we doing that? How will we do that?

Table of Contents

Key Advantages of Microsoft Azure

Microsoft Azure is a cloud platform with more than 100 services and tools for developers.

- 95% of the Fortune 500 companies run on Azure (and learning these skills can help you land your first job!)
- Easily deploy Web Apps, Mobile Apps, Databases, Data Science and AI Apps, Blockchain, Analytics with pre-built configurations
- Supports a wide range of
  - Open source libraries (Node, .NET core, spark, hadoop, kafka)
  - Programming languages (Python, C#, JavaScript, Java, Ruby)
  - Operating systems (iOS, Android, Windows, MacOS, Linux)

Please Note:

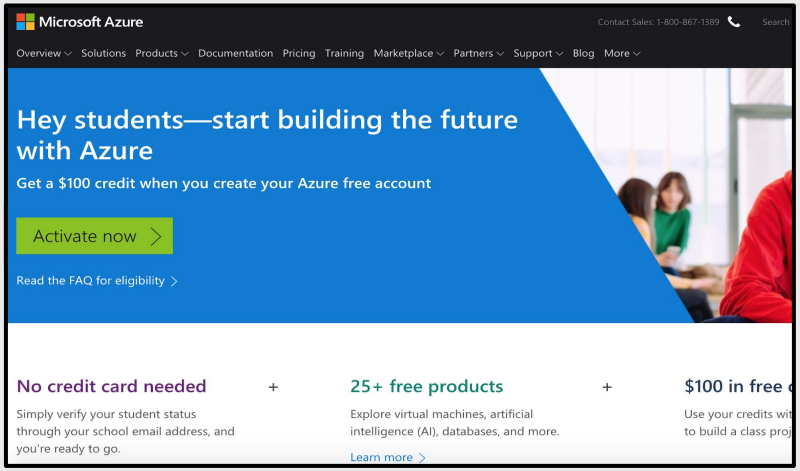
We recommend completing this entire workshop in an Incognito Window in Chrome.

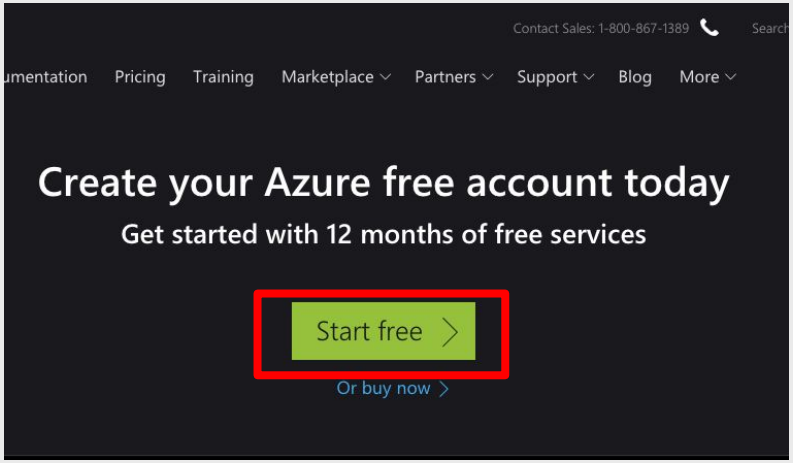
Create Microsoft Azure Account

http://mlhlocal.host/azure-student

http://mlhlocal.host/azure-nonstudent

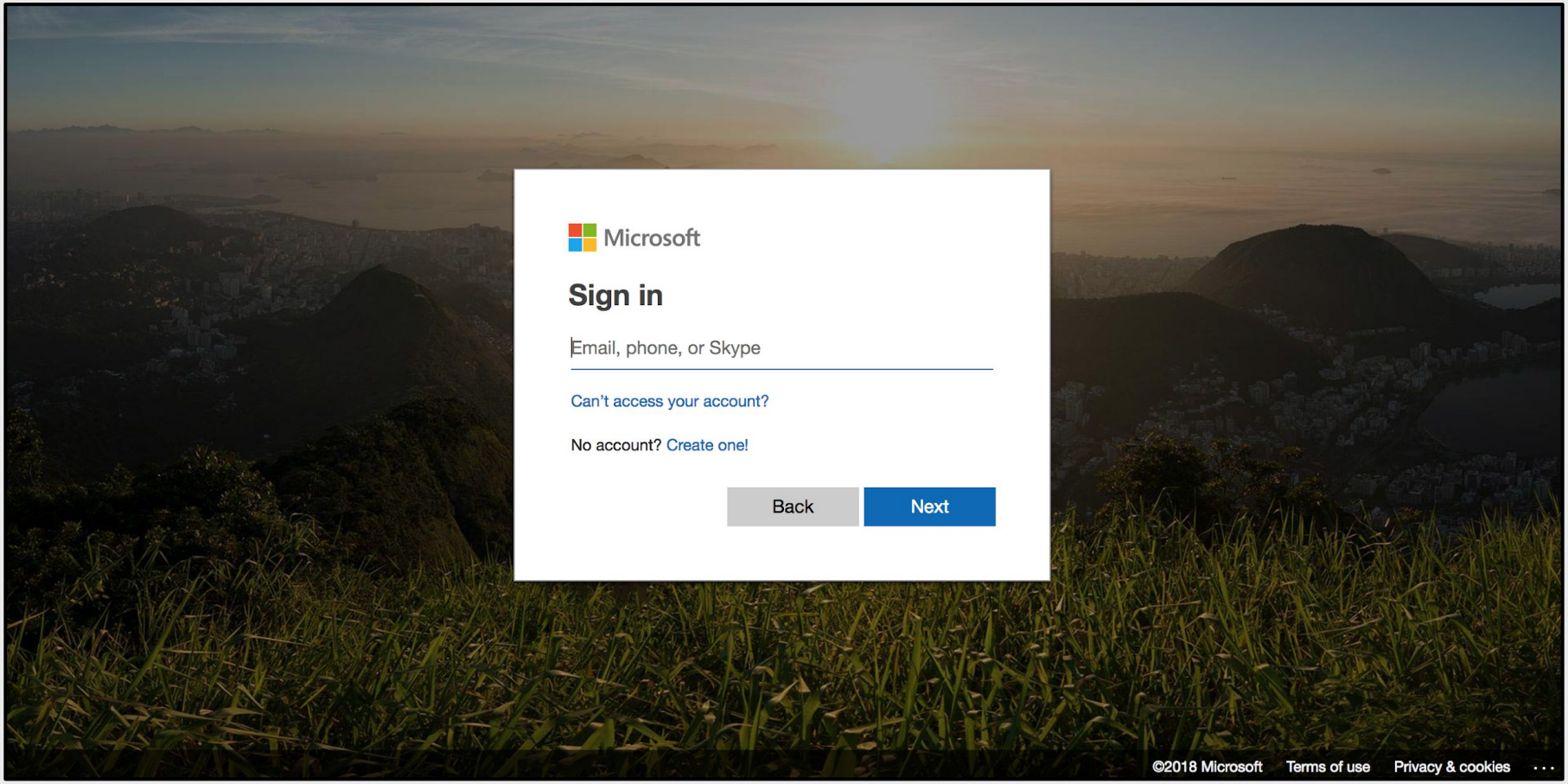
Navigate to the top URL if you have a student email address and the bottom one if you don't. Select Activate now or Start free .





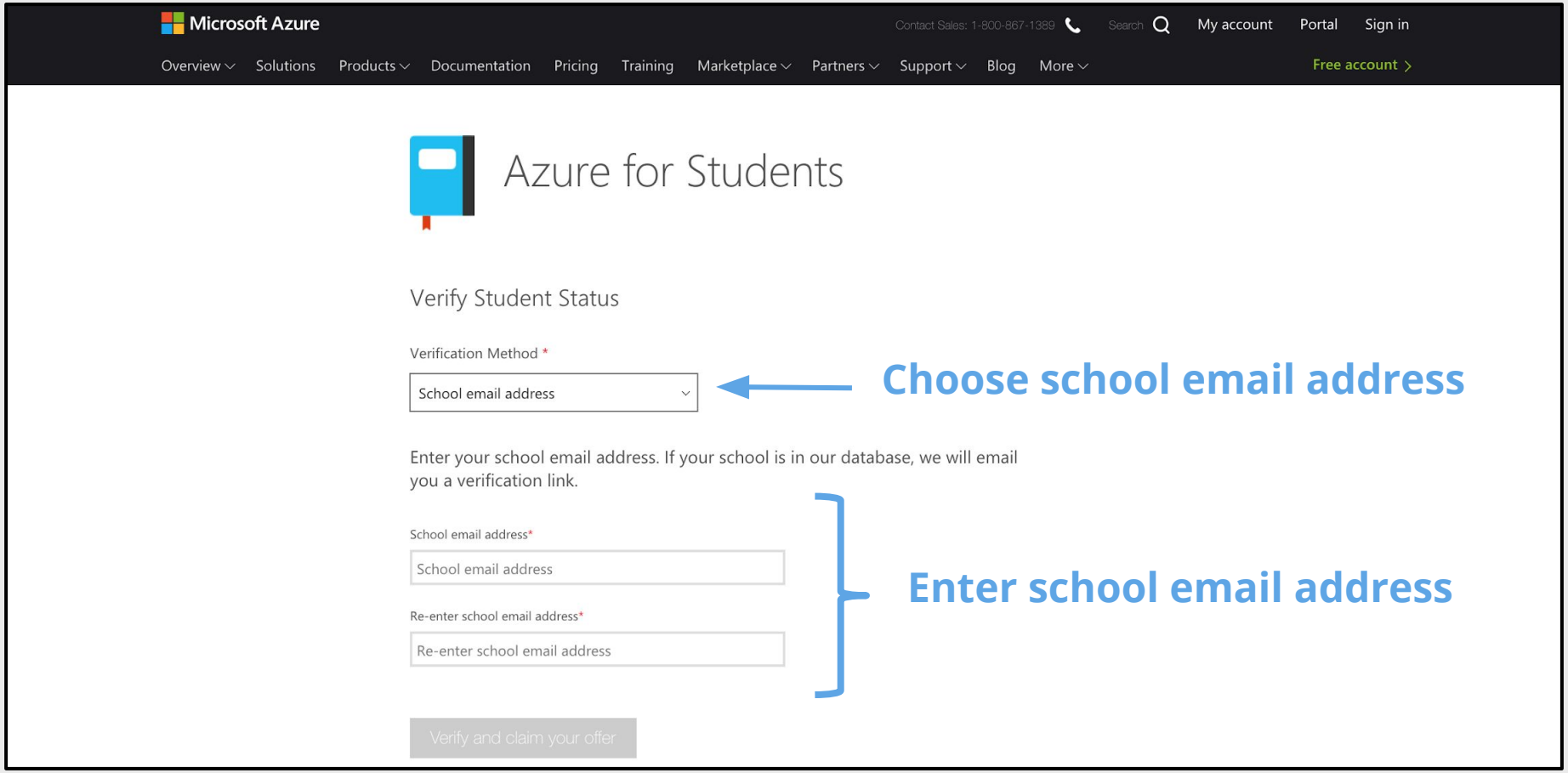
Create Microsoft Azure Account

Log in (or Sign up) with a Microsoft account



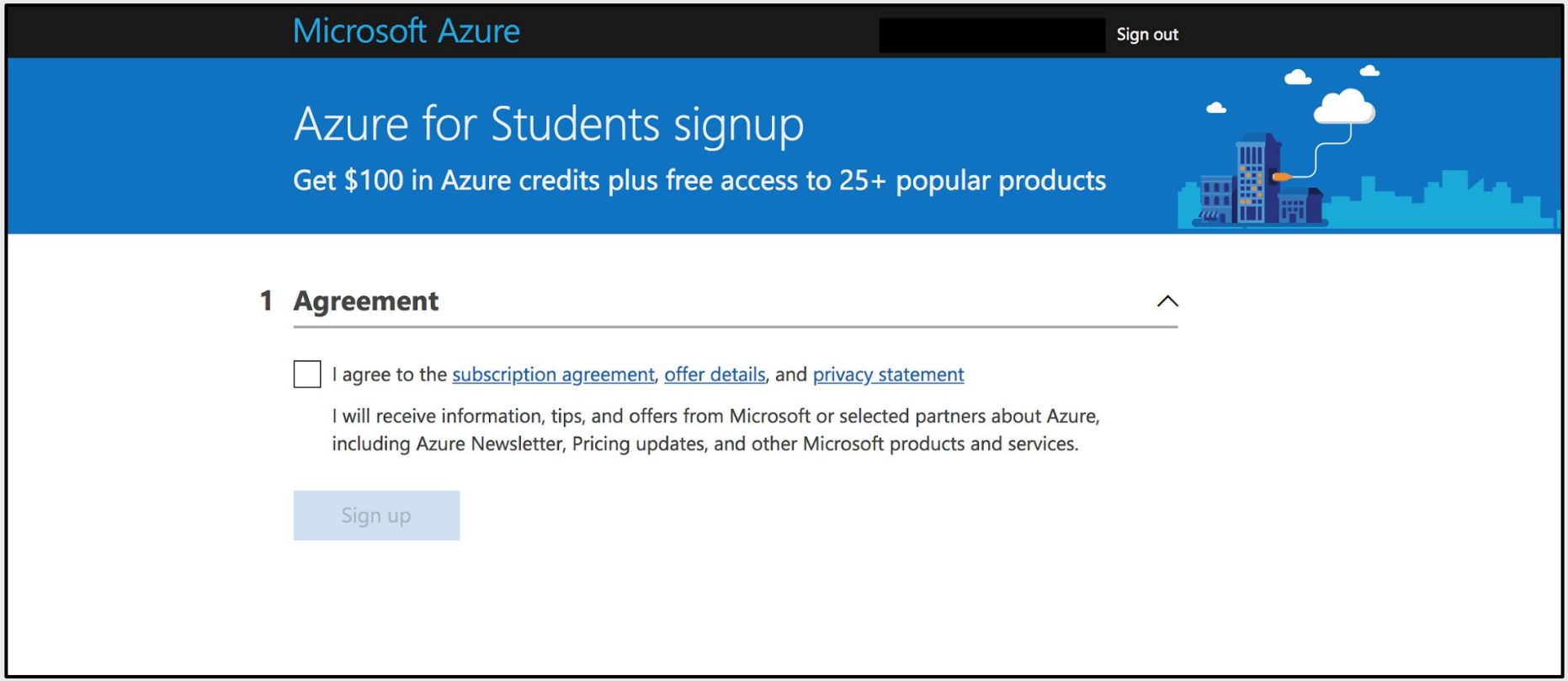
Create Microsoft Azure Account

Enter your school email address (ending in .edu), then follow the verification URL in the email sent to that address.



Create Microsoft Azure Account

Accept the terms of service and start using Microsoft Azure!



Microsoft Azure is a cloud platform with awesome developer tools. However, we're not going to write our code on Azure; that's what Visual Studio Code is for!

Key Advantages of Visual Studio Code

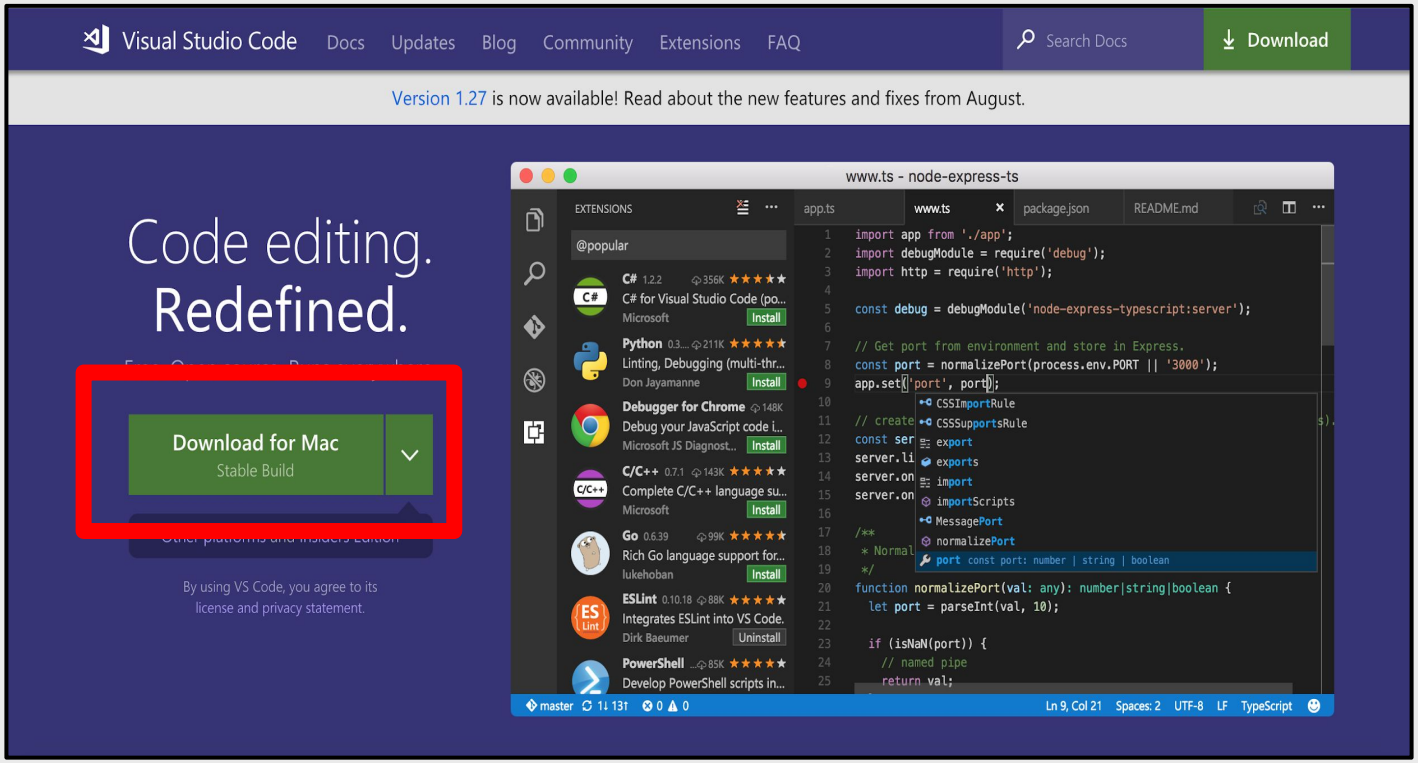
Visual Studio Code is a simple, code-centric integrated development environment (IDE).

- Available for Mac, Linux, and Windows
- Key features include debugging, version control with Git, and intelligent code completion
- Supports 36 programming languages and offers customized tools and plug-ins to make your dev environment your own!

Download Visual Studio Code

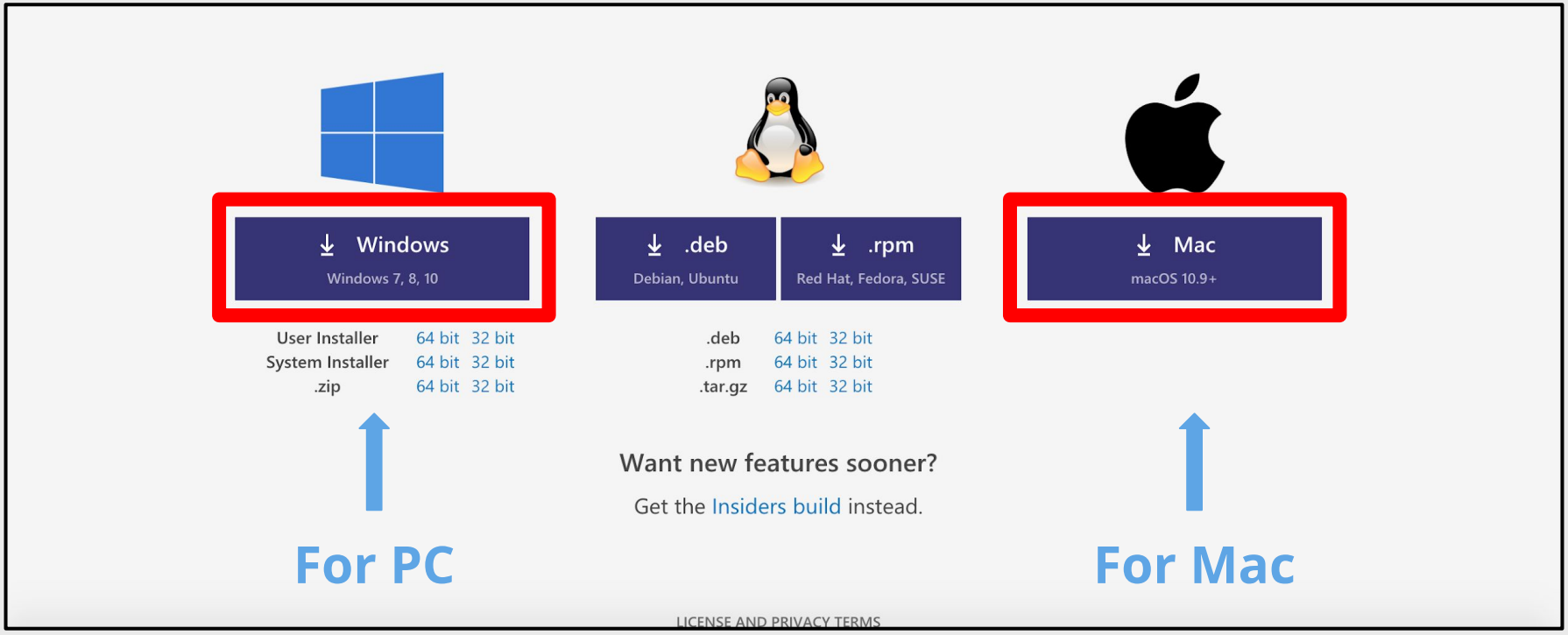
http://mlhlocal.host/vscode

Follow the URL and select Download.



Download Visual Studio Code

(If you're having trouble, scroll to the bottom of the page and manually select the correct package for your system)



Download and Install Visual Studio Code

Open the VSCode .zip file after it has finished downloading to set up Visual Studio Code.

Your browser may let you open the file directly from the download page...

...or you may have to open your Downloads folder and select the .zip file:

Visual Studio Code has many excellent features such as integrating directly with Git for version control.

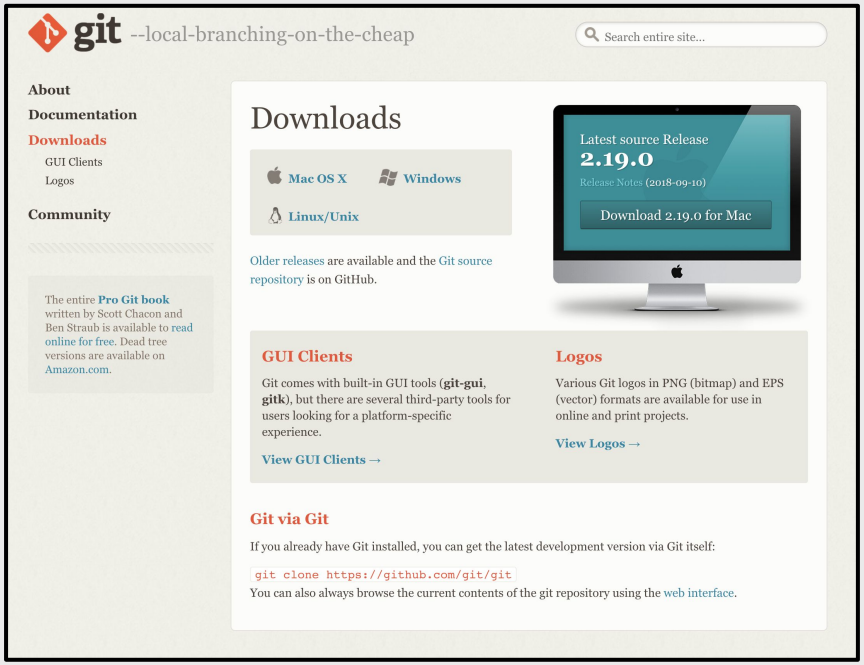
Let's get set up with Git!

Hello, Git!

Git is a version control system that we'll use to save changes and deploy our app with Azure. We need to install Git!

http://mlhlocal.host/install-git

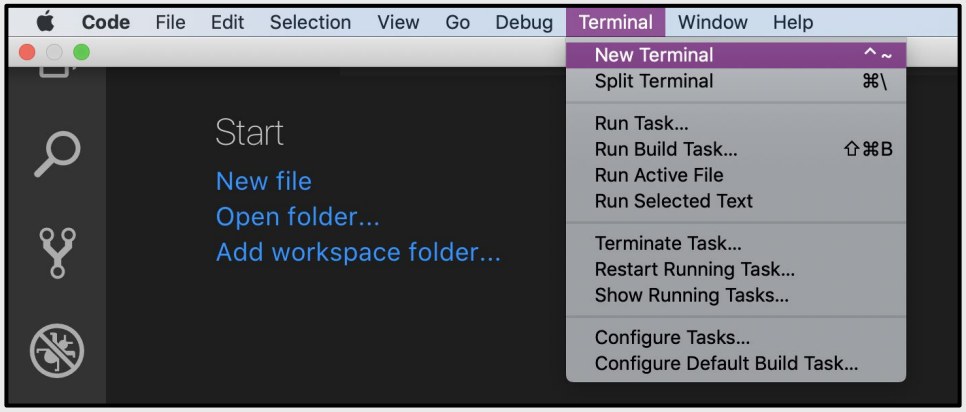
1. Visit the URL above.
2. Select your operating system.
3. Follow the prompts to install Git!
4. If Git asks you which default editor to use, we recommend VSCode!



Configure Git Locally

We need to run two commands so Visual Studio Code can use Git.

1. Open Visual Studio Code.
2. Open a New Terminal.



3. Type these commands but substitute your name and email address :

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

@localhost > git --version
git version 2.17.1 (Apple Git-112)
@localhost > git config --global user.name "MLHLocalhost"
@localhost > git config --global user.email "localhost@mlh.io"
@localhost >
```

Great! You set up your Azure account, downloaded Visual Studio Code and configured Git!

Now let's get a better understanding of the technologies that our HackerLog app uses.

Table of Contents

Hello, Node.js!

Remember from earlier that HackerLog is a Node app. What's Node?

- Node.js is server-side JavaScript allowing us to create applications that send and receive information. You get to use the same language on the front-end and back-end!
- Node.js is designed to quickly deal with multiple tasks (for example, receiving messages, updating databases, processing data) simultaneously.
- It's open-source with thousands of packages available for use!

Install Node

Navigate to the URL that suits your computer and follow the installation instructions:

http://mlhlocal.host/node-mac — Download for Mac

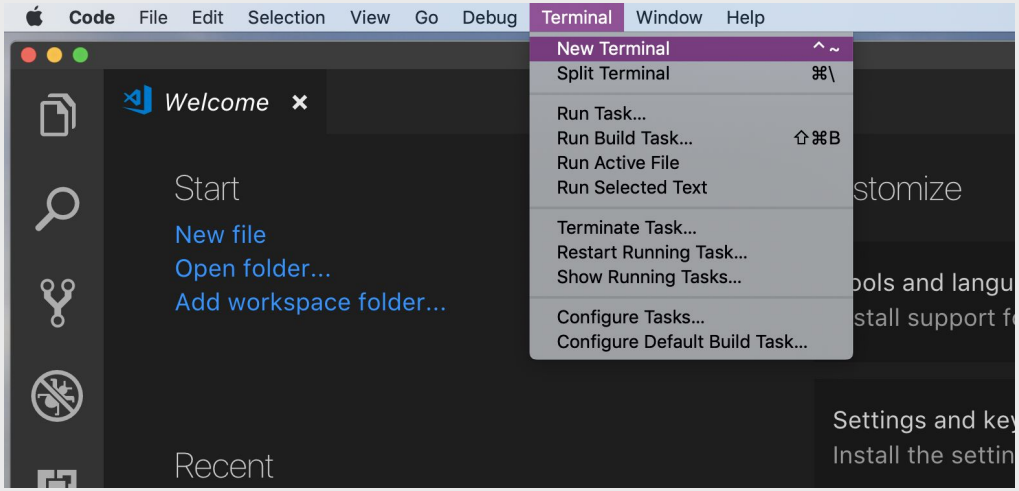
http://mlhlocal.host/node-pc — Download for PC

http://mlhlocal.host/node-linux — Download Page to Select Linux Version

Check Node.js Installation

Instructions

1. Open Visual Studio Code
2. Select **Terminal -> New Terminal**.
3. Type `node --version` in your terminal.



Hello, Express!

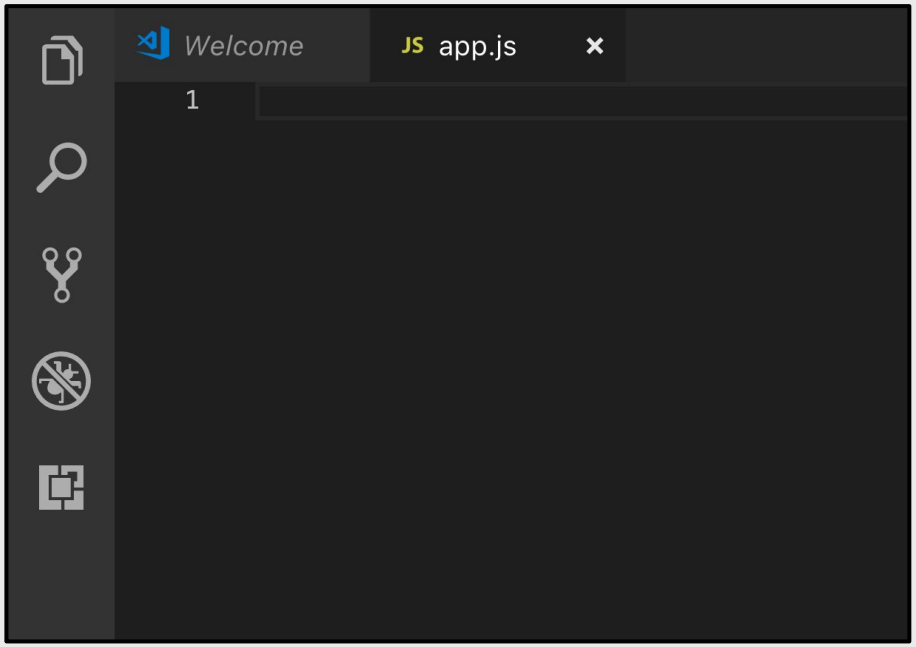
Express is a very simple web framework for Node.js.

- 2 lines of code allow you to use Express in any JavaScript app!
- Express makes it easy to handle app↔server communication via HTTP, such as **`get`** and **`post`** requests.
- We won't install Express just yet; more on that later.

Hello, World!

Now that you have a basic understanding of Node and Express, let's develop the simplest app we can to get some experience.

1. Return to Visual Studio Code.
2. Click **File** -> **New File**.
3. Save your file as **`app.js`**.



Hello, World!

4. Add the following code to the file:

app.js

```
1  const express = require('express');
2  const app = express();
3  const port = 3000;
```

What does this code mean?

- **Line 1:** Allows our file to use the Express package
- **Line 2:** Creates a new express app
- **Line 3:** Saves `3000` as the local port on which we'll later run the app

Hello, World!

5. Add the following code to the file:

```
5   app.get('/', (req, res) => {
6       res.send('Hello World!');
7       console.log('Success!');
8   })
```

What does this code mean?

- **Line 5:** Call the `.get()` method on our app. `'/'` refers to the app home page (usually **`index.html`**). When someone requests the home page of our app, do the following with the request (`req`) and response code (`res`):
- **Line 6:** Use the `.send()` method to send `Hello World!` as the response.
- **Line 7:** Print `Success` in the console.

Hello, World!

```
5   app.get('/', (req, res) => {
6       res.send('Hello World!');
7       console.log('Success!');
8   })
```

What does this code do?

- This block of code is called a **route handler**.
- A route handler requires a method – `.get()` – and a route – `'/'`
- A route handler also takes a **callback function** to tell the app what to do with the request and response

Hello, World!

6. Add the following code to the file:

```
10    app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

What does this code mean?

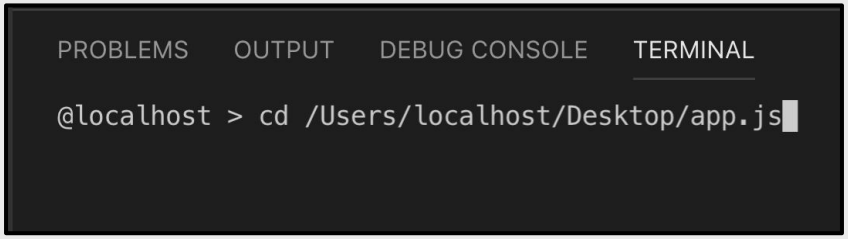
- **Line 10:** On the local port that we saved earlier, listen for requests. Log `Example app listening on port 3000!` to the console.

Now let's run our app!

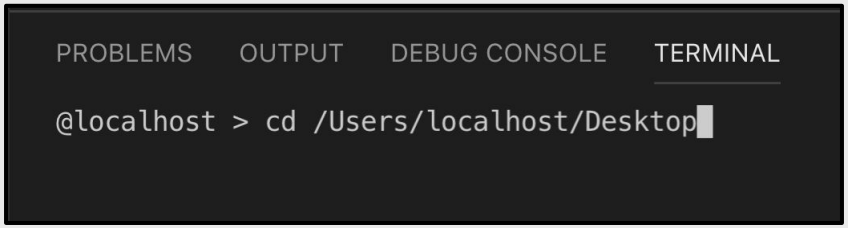
Hello, World!

7. Right click `app.js` and select **Copy Path**.

8. In your terminal, type `cd` and paste the path you just copied.

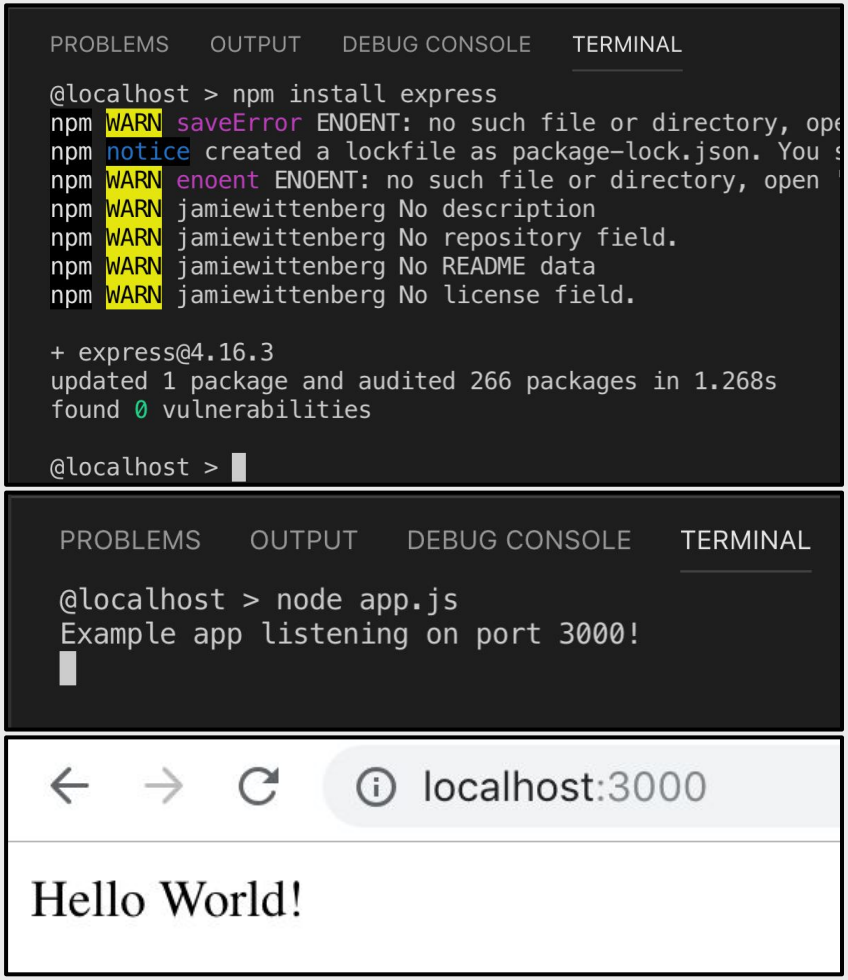


9. Delete `app.js` from the end before hitting [Enter].



Hello, World!

10. Type `npm install express` so that we can use Express on our computer.
11. Type `node app.js` to run our file.
12. Visit **`localhost:3000`** in your browser.
13. Go back to your terminal. What's changed?
14. Enter [CTRL C] to kill the server.



Express.js

In that simple app, you created a route handler using the **`.get()`** method. You also used **`.listen()`**. Let's learn two more, **`.post()`** and **`.redirect()`**!

`.post()`

- An HTTP method allowing the app to update information in the app's database based on information sent from the user
- It requires the path of the route and a callback function.

`.redirect()`

- Tells the app to send the user to another page. In our app, we'll use it to return the home page to the user after they submit an update, or return an error page if something is wrong
- It requires the endpoint of the page you want to redirect to as a string

Node.js creates a the server for your app, and Express handles the requests and responses.

The information in those requests and responses needs to be saved to a database. Let's learn about how this app achieves that!

MongoDB and Mongoose

We won't cover MongoDB and Mongoose in depth in this workshop, but here's what you need to know:

mongoose

- mongoDB is a non-relational database
- The updates that users **`post`** to our app will be stored in a mongoDB database
- Instead of installing MongoDB, we'll use a hosted version of it on Microsoft Azure

- mongoose is a framework for structuring data in a mongoDB database.

Now you have experience with Node.js and Express and a base level understanding of MongoDB and Mongoose.

Let's download our project!

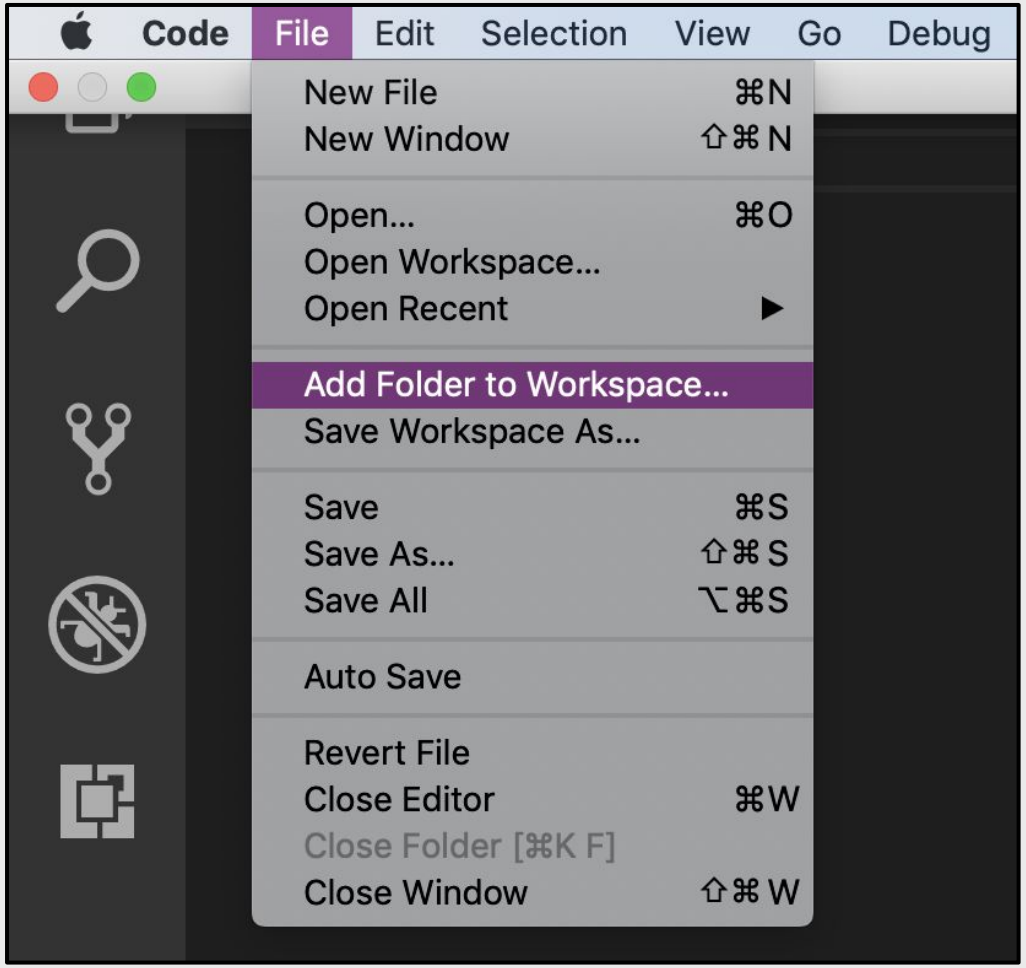
Download the Hacker Log Starter Code

http://mlhlocal.host/azure-web-code

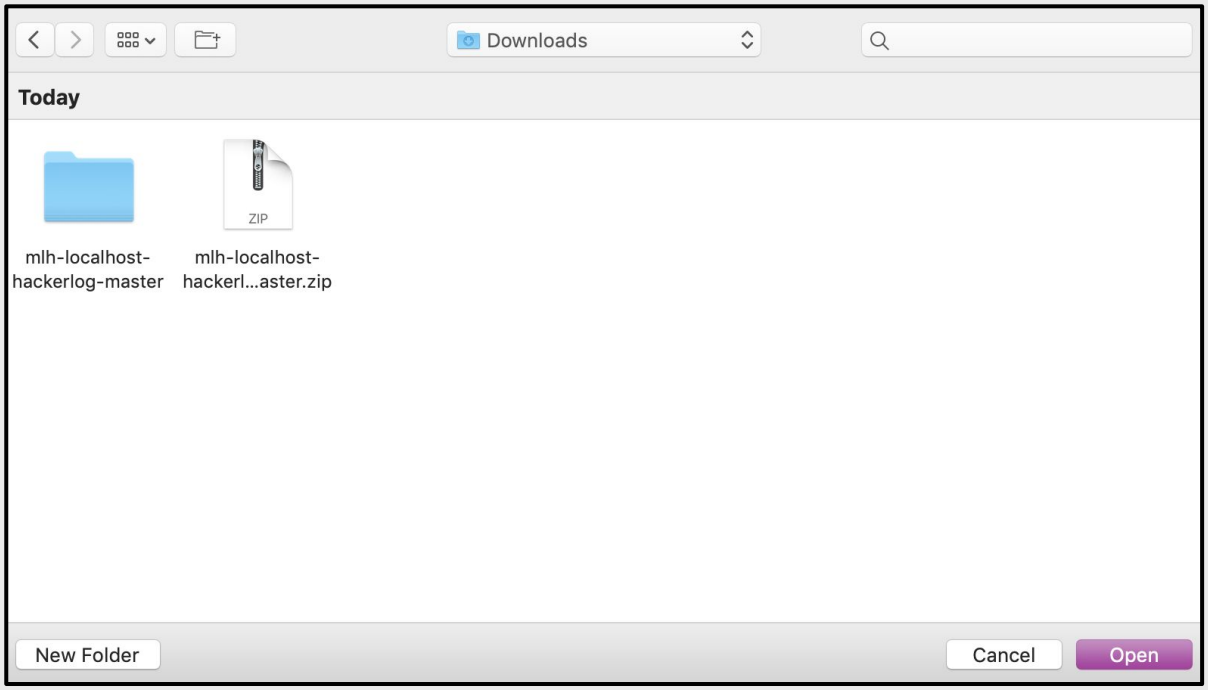
1. Navigate to the URL above.
2. A zip file of the code will be downloaded.
3. We need to unzip the file and open it in Visual Studio Code. These steps are different for MacOS and Windows – please follow the next steps **very carefully** depending on your operating system!

Add Project to Visual Studio Code: MacOS

1. Click the downloaded zip file. It will be automatically unzipped.
2. In Visual Studio Code, Click **File** -> **Add Folder to Workspace**.

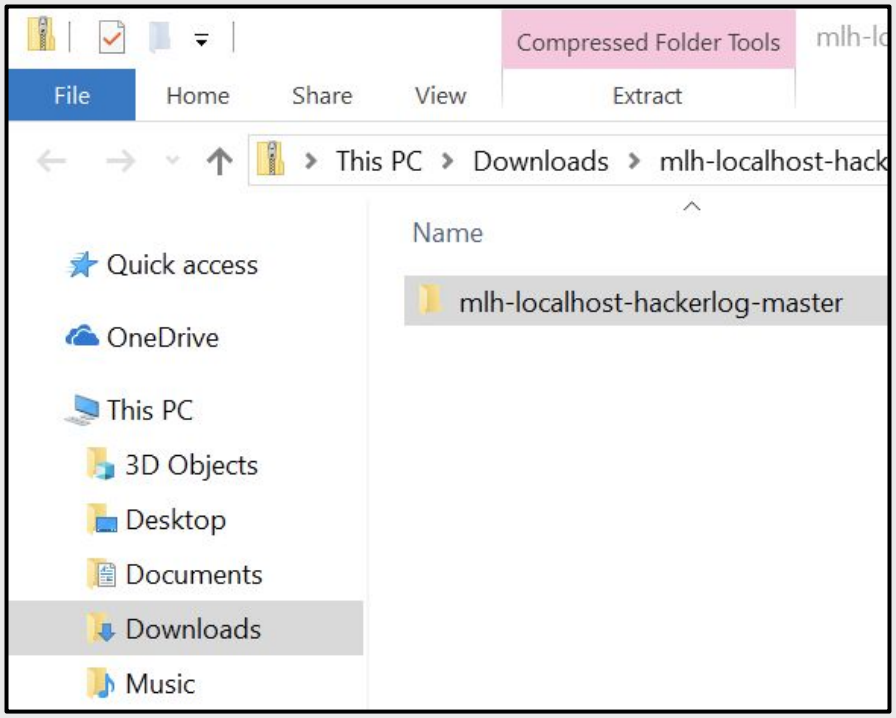


Add Project to Visual Studio Code: MacOS



3. In the file navigator modal, navigate to Downloads.
4. Click **`mlh-localhost-hackerlog-master`**.
5. Click **Add**.

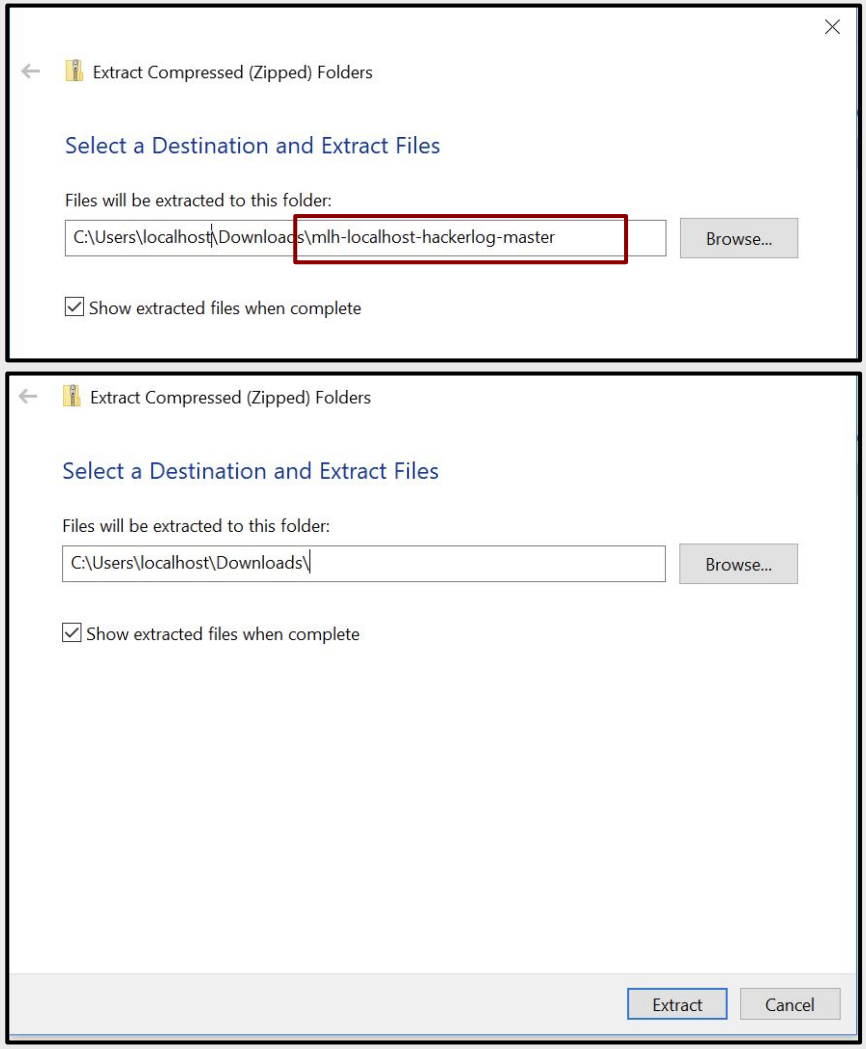
Add Project to Visual Studio Code: Windows



1. On a Windows computer, open the Downloads folder.
2. Click **`mlh-localhost-hackerlog-master`**.
3. Click **Extract** at the top.
4. Click **Extract All**.

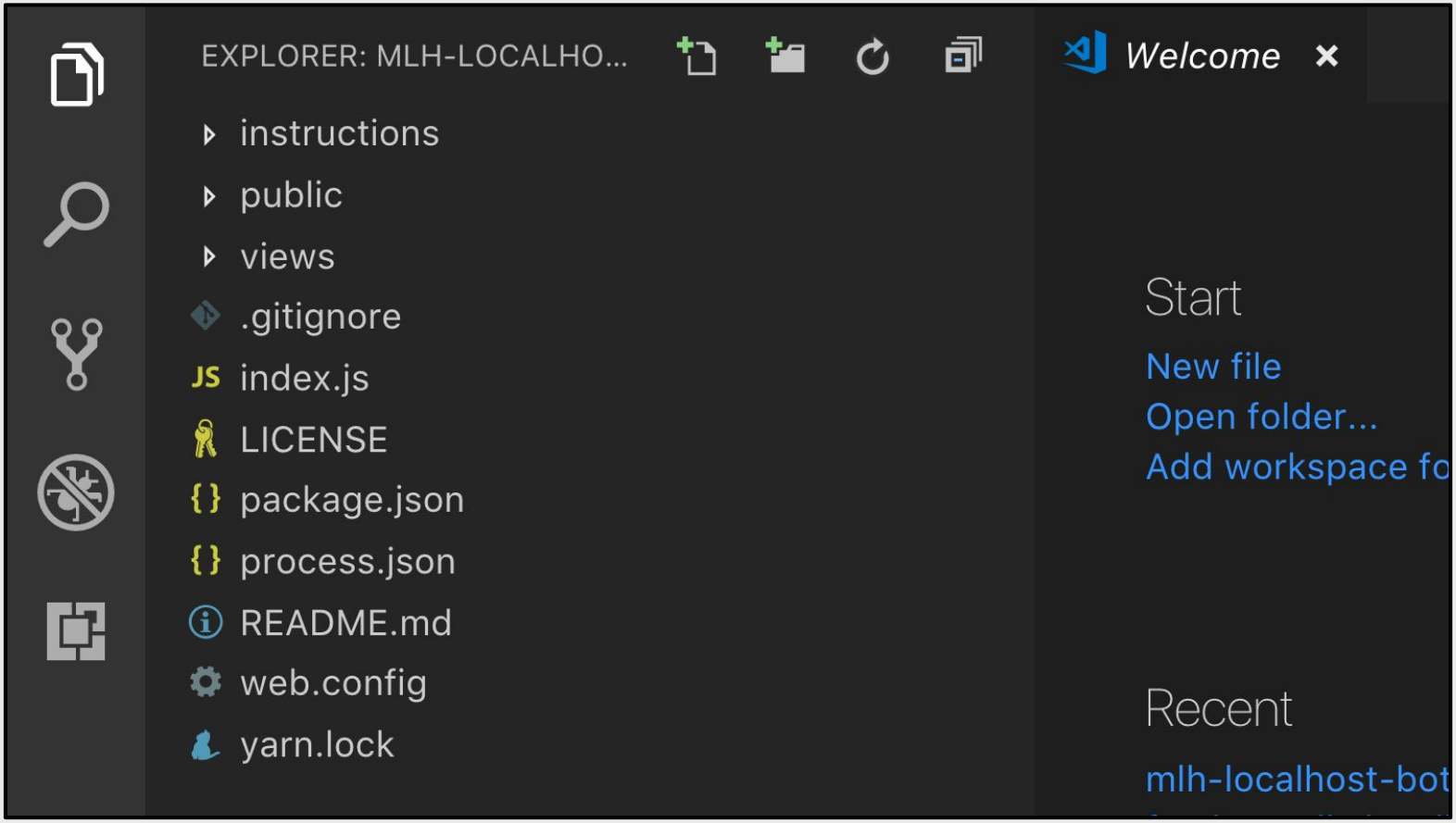
Add Project to Visual Studio Code: Windows

5. This screen should pop up. Delete **`mlh-localhost-hackerlog-master`**. Click **Extract**.
6. In Visual Studio Code, click **File** -> **Add Folder to Workspace**.
7. Select this folder from your Downloads.



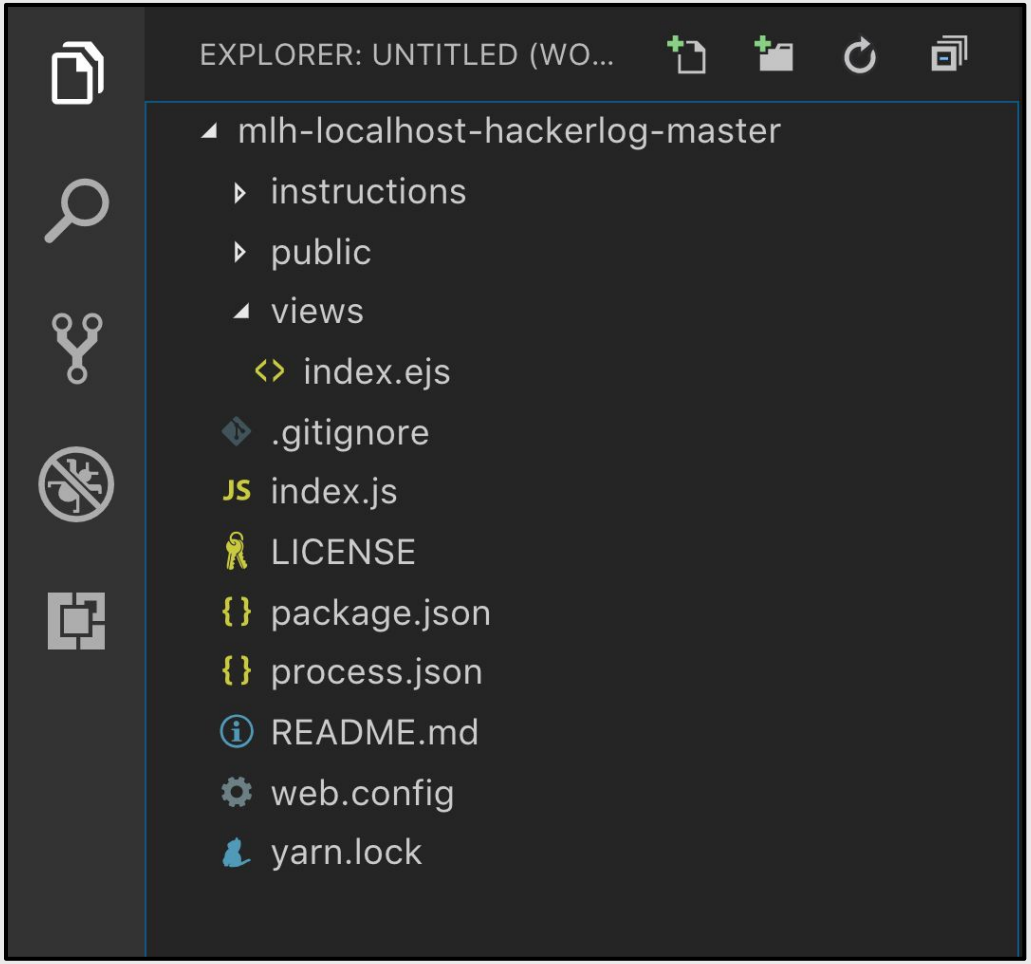
Add Project to Visual Studio Code

On both Mac and PC, Visual Studio Code looks like this:



What's in this project?

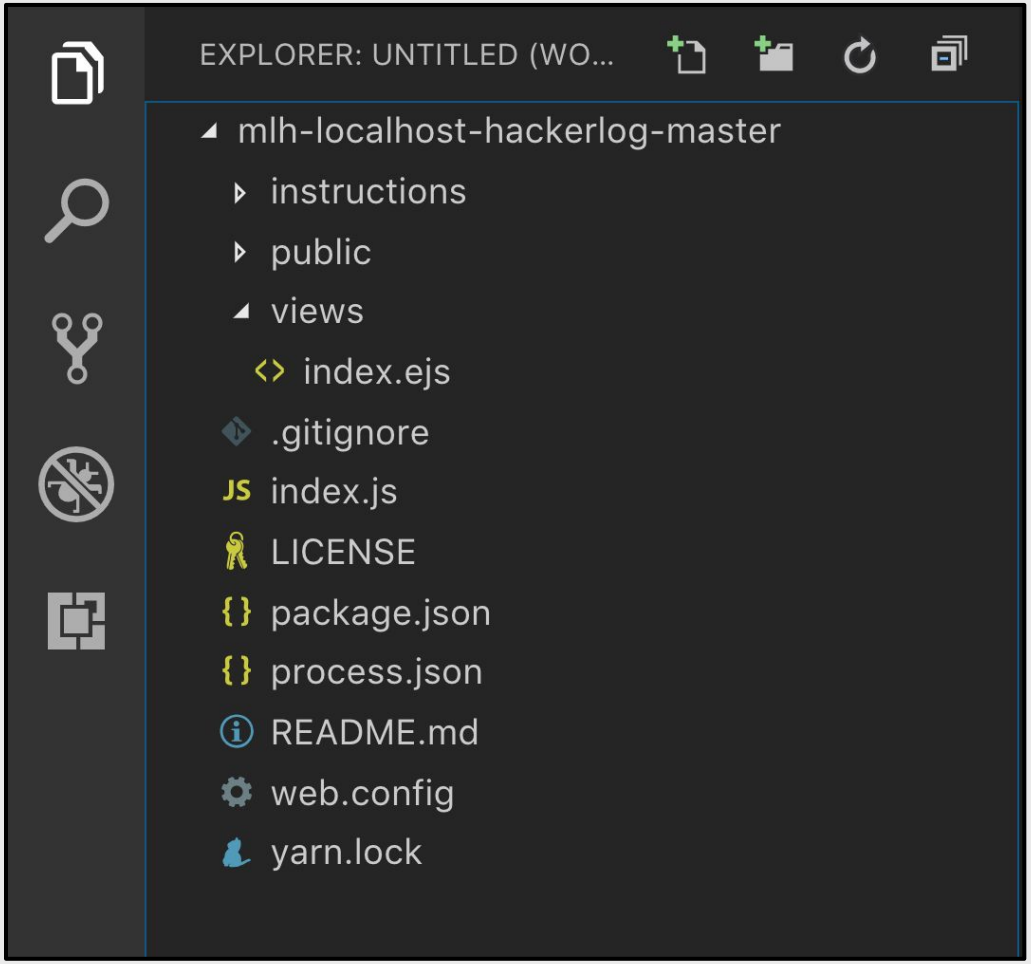
What are these files?



- **`public/`** – this directory contains files for the browser to display your web app, such as CSS and fonts
- **`views/index.ejs`** – an Embedded JavaScript file, the main landing page of our app

What's in this project?

What are these files?



- **`index.js`** - this file does the hard work of our app; it tells the app how to handle requests and connect to the database
- **`package.json`** - this file tells the project all of the necessary node packages for the app to run

Code Review

Open **`index.js`** in Visual Studio Code and let's review!

index.js

```
1   const express = require('express');
2   const mongoose = require('mongoose');
3   const bodyParser = require('body-parser');
4   const moment = require('moment');
5   const app = express();
```

- **Lines 1 - 4** allow the app to use the listed Node packages.
- **Line 5** creates an Express app, which will handle routing, or how the app receives and sends information

Code Review

index.js

```
7   // Setup express
8   app.set('view engine', 'ejs');
9   app.set('views', __dirname + '/views');
10  app.use(express.static(__dirname + '/public'));
11  app.use(bodyParser.urlencoded({ extended: false }));
12  app.use(bodyParser.json());
```

- **Lines 8 - 12** tell the Express app where to find and how to use the files for view rendering and presentation, as well as how to read communication from the browser

Code Review

index.js

```
14  // Get env variables
15  const port = process.env.PORT || 3000;
16  const perPage = process.env.PAGE_SIZE || 10;
17  const mongoUrl = process.env.MONGODB_URI || 'mongodb://localhost:27017/hackerlog';
18  const defaultPassword = process.env.HACKERLOG_PASSWORD || 'P@ssw0rd!';
```

- **Lines 15 - 18** tell the app to retrieve environment variables (env) from where the app is being run. If no environment variable is available, it sets it to a default value (the value to the right of the || pipes).
- **Line 18** contains the default password, which you can change.

Code Review

index.js

```
21  const updateSchema = mongoose.Schema({
22    name: { type: String, required: true },
23    update: { type: String, required: true }
24  }, {
25    timestamps: true
26  });
27  const Update = mongoose.model('update', updateSchema);
```

- **Line 21** saves the process of creating a schema (structure/format of data) into a variable called **`updateSchema`**.
- **Lines 22 - 23** create a schema (a blueprint for entries in a database) with the information submitted to the form on the HackerLog app
- **Line 27** saves the process of adding a new entry to the database to a variable called **`Update`**.

Code Review

index.js

```
29  // Routes
30  app.get('/', (req, res) => {
31    const page = Math.max(0, req.query.page);
32    const wrongPassword = req.query.wrongPassword;
33    Update.find().limit(perPage).skip(perPage * page).sort({ createdAt: 'desc' }).exec().then((updates) => {
34      Update.count().exec().then((count) => {
35        const pages = count / perPage;
36        res.render('index', { title: 'HackerLog', updates, wrongPassword, page, pages, moment });
37      }).catch(() => {
38        res.redirect('/error');
39      });
40    }).catch(() => {
41      res.redirect('/error');
42    });
43  });
```

- **Line 30** tells the app that when the URL for the webpage is entered, the app should run the code that comes after the open bracket ({) at the end of the line (and before the closing bracket (}) on line 43), and what to do with the request (**req**) and the response (**res**)

Code Review

index.js

```
29  // Routes
30  app.get('/', (req, res) => {
31    const page = Math.max(0, req.query.page);
32    const wrongPassword = req.query.wrongPassword;
33    Update.find().limit(perPage).skip(perPage * page).sort({ createdAt: 'desc' }).exec().then((updates) => {
34      Update.count().exec().then((count) => {
35        const pages = count / perPage;
36        res.render('index', { title: 'HackerLog', updates, wrongPassword, page, pages, moment });
37      }).catch(() => {
38        res.redirect('/error');
39      });
40    }).catch(() => {
41      res.redirect('/error');
42    });
43  });
```

- **Lines 33 - 36** tell the app to retrieve data from the database, calculate how many pages to display (10 entries per page), and render the data on the page

Code Review

index.js

```
29  // Routes
30  app.get('/', (req, res) => {
31    const page = Math.max(0, req.query.page);
32    const wrongPassword = req.query.wrongPassword;
33    Update.find().limit(perPage).skip(perPage * page).sort({ createdAt: 'desc' }).exec().then((updates) => {
34      Update.count().exec().then((count) => {
35        const pages = count / perPage;
36        res.render('index', { title: 'HackerLog', updates, wrongPassword, page, pages, moment });
37      }).catch(() => {
38        res.redirect('/error');
39      });
40    }).catch(() => {
41      res.redirect('/error');
42    });
43  });
```

- **Lines 37 - 42** handle any errors in the process by taking the user to the error page

Code Review

index.js

```
45  // Posting update
46  app.method('where', (paramOne, paramTwo) => {
47    const { body: { name, update, password } } = req;
48    if (!name || !update) {
49      res.redirect('/error');
50    } else if () {
51      const userUpdate = new Update({ name, update });
52      userUpdate.save().then(() => {
53        // do a redirect here
54      }).catch(() => {
55        // do a redirect here
56      });
57    } else {
58      // do a redirect here
59    }
60  });
```

You'll write the code for these later!

- We'll come back to lines 46 - 60, since you'll be filling in some of it!

Code Review

index.js

```
66  mongoose.connect(mongoUrl, { useNewUrlParser: true }).then(() => {
67    console.log("Connected to MongoDB!");
68    console.log("Starting webserver..");
69    app.listen(port, '0.0.0.0', () => console.log(`HackerLog app listening on port ${port}!`));
70  }).catch(() => {
71    console.log("Could not connect to MongoDB server! Shutting down...");
72    process.exit(1);
73  });
```

- **Line 66** connects the app to the mongo database, then:
- **Lines 67 and 68** log success messages onto the console
- **Line 69** starts the app using Node
- **Lines 70 - 73** handles any errors that may occur

Index.js handles the requests our app receives and the responses that it sends. These requests come from the browser and are entered into index.ejs. The result is HTML to send back to the browser.

Code Review

index.ejs

```
13  <form class="update-form" action="/update" method="post">
14    <% if (wrongPassword) { %>
15      <div class="error-message">
16        <span class="message">That password is not correct!</span>
17      </div>
18    <% } %>
19    <section class="auth">
20      <div class="field">
21        <label class="input-label" for="name">Name</label>
22        <input class="input-field" type="text" id="name" name="name" required>
23      </div>
24      <div class="field">
25        <label class="input-label" for="password">Password</label>
26        <input class="input-field" type="password" id="password" name="password" required>
27      </div>
28    </section>
29    <section class="update-section">
30      <div class="update-field">
31        <label class="input-label" for="update">Update</label>
32        <input class="input-field" type="text" id="update" name="update" required>
33      </div>
34      <div class="field">
35        <div class="spacer"></div>
36        <input class="input-field submit-button" id="submit" type="submit" value="SUBMIT" required>
37      </div>
38    </section>
39  </form>
```

- **Lines 13-39** create the form that you see on the app

Note:

This file is dense, so we'll only be touching on the code that is critical for our purposes!

Code Review

index.ejs

```
13  <form class="update-form" action="/update" method="post">
14    <% if (wrongPassword) { %>
15      <div class="error-message">
16        <span class="message">That password is not correct!</span>
17      </div>
18    <% } %>
19    <section class="auth">
```

- **Line 13** has a `method` attribute with the value `"post"`. This means that when someone submits the form, it sends a post request to the app. It also has an `action` attribute with the value `"/update"`. This is how we separate the different kinds of requests.

Code Review

index.ejs

```
13  <form class="update-form" action="/update" method="post">
14    <% if (wrongPassword) { %>
15      <div class="error-message">
16        <span class="message">That password is not correct!</span>
17      </div>
18    <% } %>
19    <section class="auth">
```

- **Lines 14 - 18** post an error message if the wrong password is submitted.

Code Review

index.ejs

```
20      <div class="field">
21        <label class="input-label" for="name">Name</label>
22        <input class="input-field" type="text" id="name" name="name" required>
23      </div>
24      <div class="field">
25        <label class="input-label" for="password">Password</label>
26        <input class="input-field" type="password" id="password" name="password" required>
27      </div>
28    </section>
29    <section class="update-section">
30      <div class="update-field">
31        <label class="input-label" for="update">Update</label>
32        <input class="input-field" type="text" id="update" name="update" required>
33      </div>
```

- **Lines 20 - 23** create the name field on the form.
- **Lines 24 - 28** create the password field on the form. The `name` attributes in this section are used to form the database schema in **`index.js`**
- **Lines 29 - 33** create the update field on the form.

Now that you understand the code that's written in the app, let's write the missing code!

Your Turn!

Let's go back to **lines 46 - 60** in **index.js**. You're going to fill in the incomplete event handler under `// Posting Update`!

index.js

```
45  // Posting update
46  app.method('where', (paramOne, paramTwo) => {
47    const { body: { name, update, password } } = req;
48    if (!name || !update) {
49      res.redirect('/error');
50    } else if () {
51      const userUpdate = new Update({ name, update });
52      userUpdate.save().then(() => {
53        // do a redirect here
54      }).catch(() => {
55        // do a redirect here
56      });
57    } else {
58      // do a redirect here
59    }
60  });
```

Your Turn!

index.js

Lines 46 - 60

This function should be able to:

1. Line 46 sends a post request. paramOne and paramTwo are the wrong words.
2. On line 46, the request should go to '/update' instead of '/'.
3. Line 50 verifies that the password the user enters matches defaultPassword.
4. Line 53 should call the redirect method on the res and redirect the user to '/'.
5. Line 55 should call the redirect method on the res and redirect the user to '/error'.
6. Line 58 should call the redirect method on the res and redirect the user to '/?wrongPassword=true'.

Want to test your code before you see the solution?

1. Save your changes by clicking **File > Save** or **[Command] + S**.
2. Open the Integrated Terminal by clicking **View > Terminal**. The terminal allows you to communicate directly with your computer.
3. Install the app dependencies with **npm install**
4. Run the program with **node index.js**
5. Open a browser and visit the URL **http://localhost:3000**
6. If your app isn't working, type **[CTRL] + C** to stop it, try to fix it and start it again with **node index.js**!

Are you ready to see if you did it?

Your Turn - Write the POST Function

1. Send a `post` request.
2. The request should go to `'/update'` instead of `'/'`.
3. Line 50 should make sure the password is correct.

index.js

```
45  // Posting update
46  app.post('/update', (req, res) => {
47    const { body: { name, update, password } } = req;
48    if (!name || !update) {
49      res.redirect('/error');
50    } else if () {
51      const userUpdate = new Update({ name, update });
52      userUpdate.save().then(() => {
53        // do a redirect here
54      }).catch(() => {
55        // do a redirect here
56      });
57    } else {
58      // do a redirect here
59    }
60  });
```

Your Turn - Write the POST Function

1. Send a `post` request.
2. The request should go to `'/update'` instead of `'/'`.
3. Line 50 should make sure the password is correct.

index.js

```
45  // Posting update
46  app.post('/update', (req, res) => {
47    const { body: { name, update, password } } = req;
48    if (!name || !update) {
49      res.redirect('/error');
50    } else if () {
51      const userUpdate = new Update({ name, update });
52      userUpdate.save().then(() => {
53        // do a redirect here
54      }).catch(() => {
55        // do a redirect here
56      });
57    } else {
58      // do a redirect here
59    }
60  });
```

Your Turn - Write the POST Function

1. Send a `post` request.
2. The request should go to `'/update'` instead of `'/'`.
3. Line 50 should make sure the password is correct.

index.js

```
45  // Posting update
46  app.post('/update', (req, res) => {
47    const { body: { name, update, password } } = req;
48    if (!name || !update) {
49      res.redirect('/error');
50    } else if (password === defaultPassword) {
51      const userUpdate = new Update({ name, update });
52      userUpdate.save().then(() => {
53        // do a redirect here
54      }).catch(() => {
55        // do a redirect here
56      });
57    } else {
58      // do a redirect here
59    }
60  });
```

Your Turn - Write the POST Function

index.js

```
45  // Posting update
46  app.post('/update', (req, res) => {
47    const { body: { name, update, password } } = req;
48    if (!name || !update) {
49      res.redirect('/error');
50    } else if (password === defaultPassword) {
51      const userUpdate = new Update({ name, update });
52      userUpdate.save().then(() => {
53        res.redirect('/');
54      }).catch(() => {
55        // do a redirect here
56      });
57    } else {
58      // do a redirect here
59    }
60  });
```

4. Line 53 should call the redirect method on the res and redirect the user to '/'.
5. Line 55 should call the redirect method on the res and redirect the user to '/error'.
6. Line 58 should call the redirect method on the res and redirect the user to `'/?wrongPassword=true'`.

Your Turn - Write the POST Function

index.js

```
45  // Posting update
46  app.post('/update', (req, res) => {
47    const { body: { name, update, password } } = req;
48    if (!name || !update) {
49      res.redirect('/error');
50    } else if (password === defaultPassword) {
51      const userUpdate = new Update({ name, update });
52      userUpdate.save().then(() => {
53        res.redirect('/');
54      }).catch(() => {
55        // do a redirect here
56      });
57    } else {
58      // do a redirect here
59    }
60  });
```

4 Line 53 should call the redirect method on the res and redirect the user to '/'.

5 Line 55 should call the redirect method on the res and redirect the user to '/error'.

6 Line 58 should call the redirect method on the res and redirect the user to '/?wrongPassword=true'.

Your Turn - Write the POST Function

index.js

```
45  // Posting update
46  app.post('/update', (req, res) => {
47    const { body: { name, update, password } } = req;
48    if (!name || !update) {
49      res.redirect('/error');
50    } else if (password === defaultPassword) {
51      const userUpdate = new Update({ name, update });
52      userUpdate.save().then(() => {
53        res.redirect('/');
54      }).catch(() => {
55        // do a redirect here
56      });
57    } else {
58      // do a redirect here
59    }
60  });
```

4 Line 53 should call the redirect method on the res and redirect the user to '/'.

5 Line 55 should call the redirect method on the res and redirect the user to '/error'.

6 Line 58 should call the redirect method on the res and redirect the user to '/?wrongPassword=true'.

Our Code is Complete!

Now that our code is complete, let's try running our app again!

index.js

```
45  // Posting update
46  app.post('/update', (req, res) => {
47    const { body: { name, update, password } } = req;
48    if (!name || !update) {
49      res.redirect('/error');
50    } else if (password === defaultPassword) {
51      const userUpdate = new Update({ name, update });
52      userUpdate.save().then(() => {
53        res.redirect('/');
54      }).catch(() => {
55        res.redirect('/error');
56      });
57    } else {
58      res.redirect('/?wrongPassword=true');
59    }
60  });
```

Try it One More Time!

1. Type **`node index.js`** in your terminal.
2. Visit `localhost:3000`!

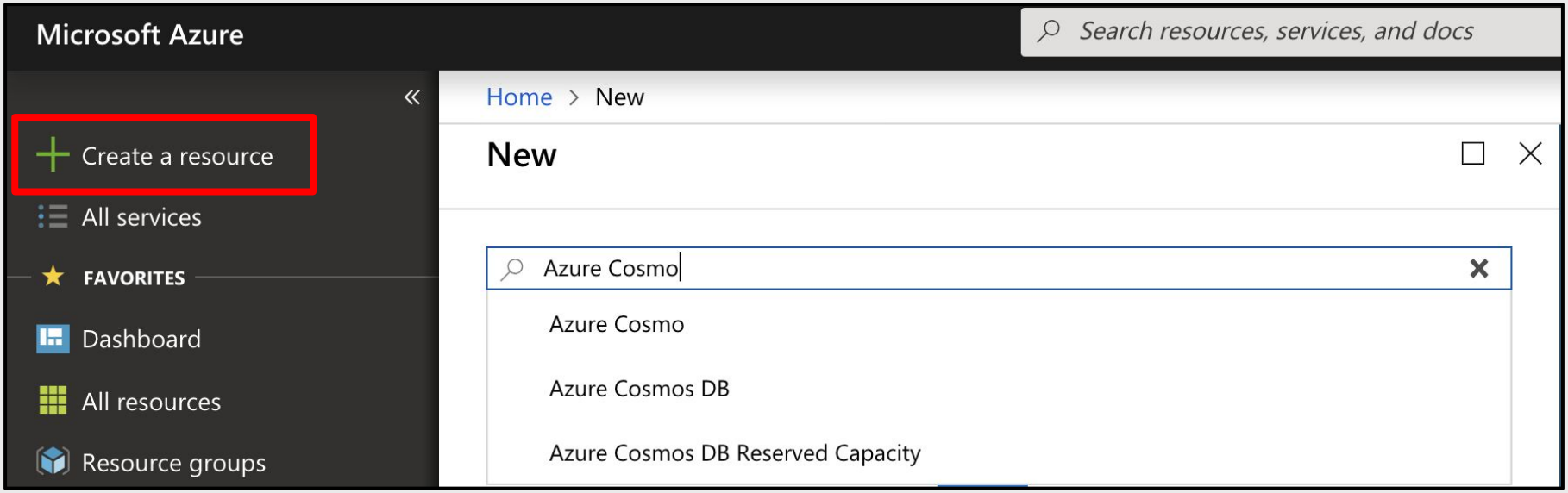
Amazing! You've learned about Node and Express, written code, and tested your app. Now, let's deploy it so others can use it!

Table of Contents

Create a Cosmos Resource

First, we'll create our database on Azure.

1. Navigate to your Azure dashboard and select **Create a Resource** then search for and select **Azure Cosmos DB**. Click **Create**.

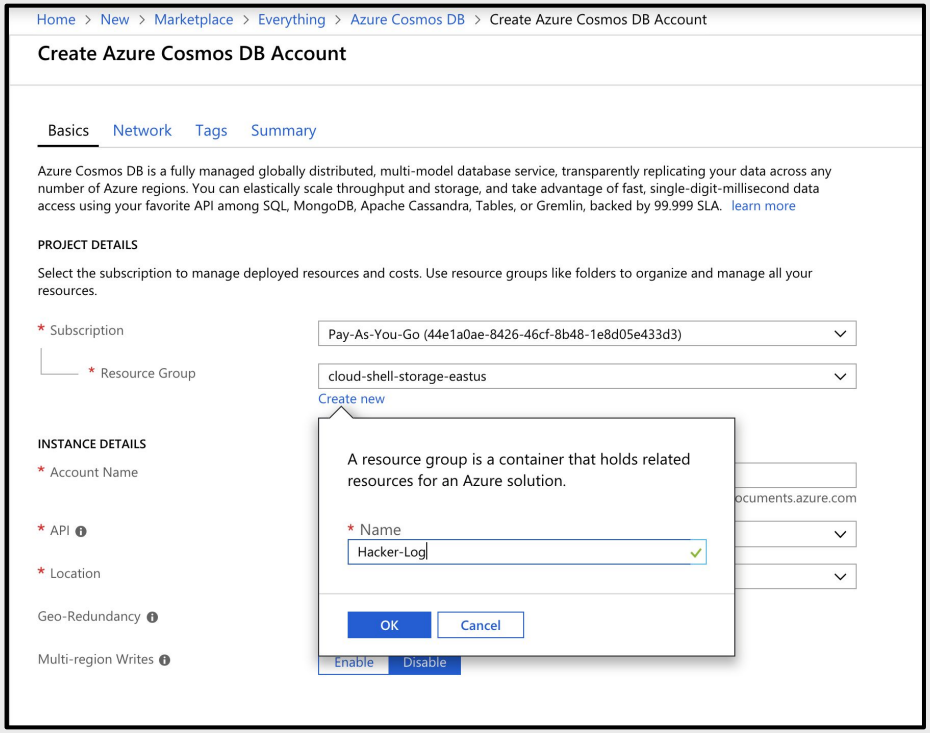


Create a Cosmos Resource

2. Select your subscription, then click **Create new** under Resource Group.

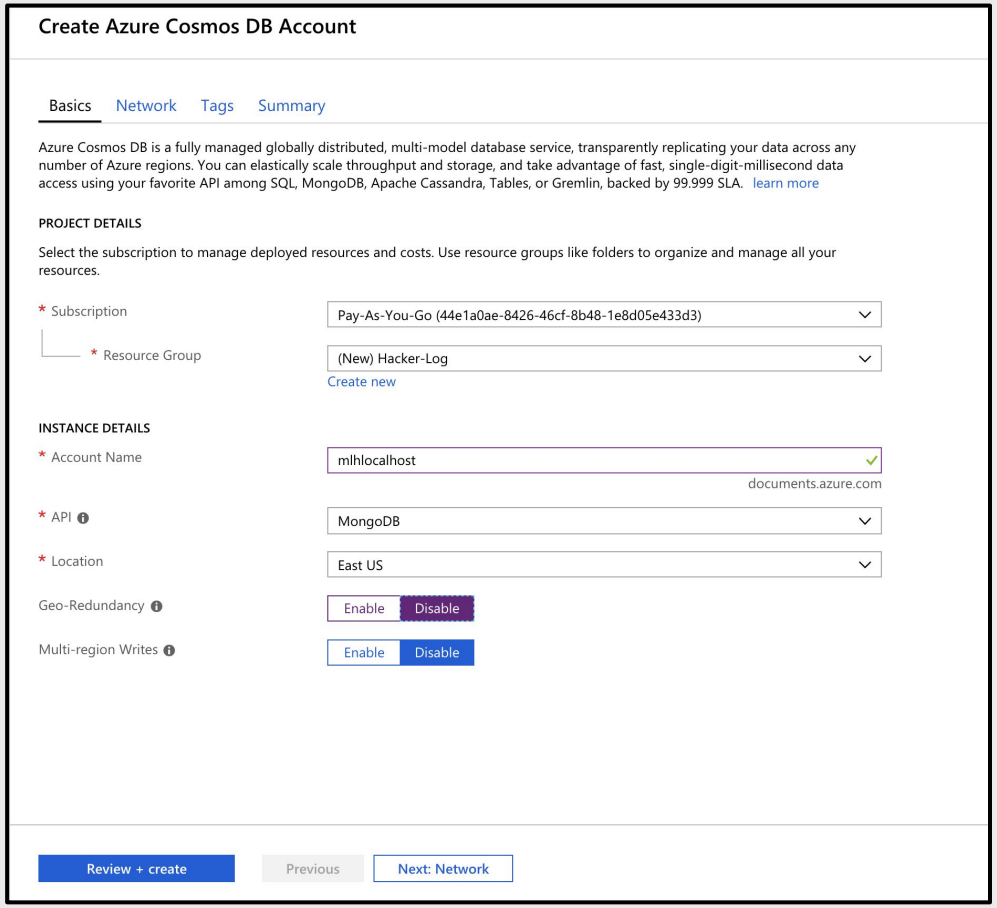
3. Name your Resource Group. You'll add your app to this group later, too.

4. Click **OK**.



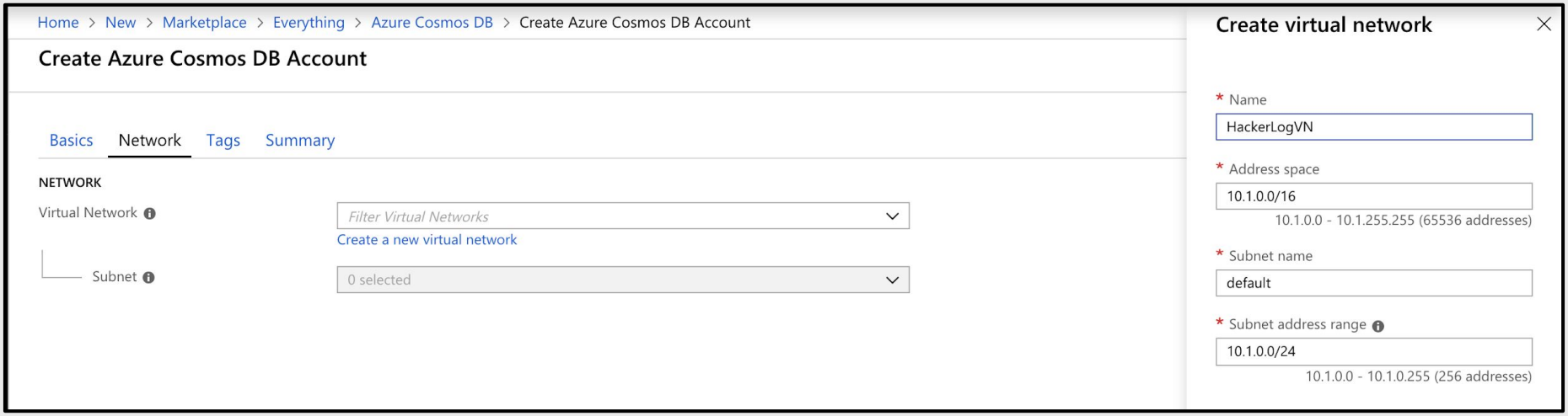
Create a Cosmos Resource

5. Create an account name.
6. Select MongoDB as the API.
7. Select your location.
8. Disable both options.
9. Click **Next: Network**.



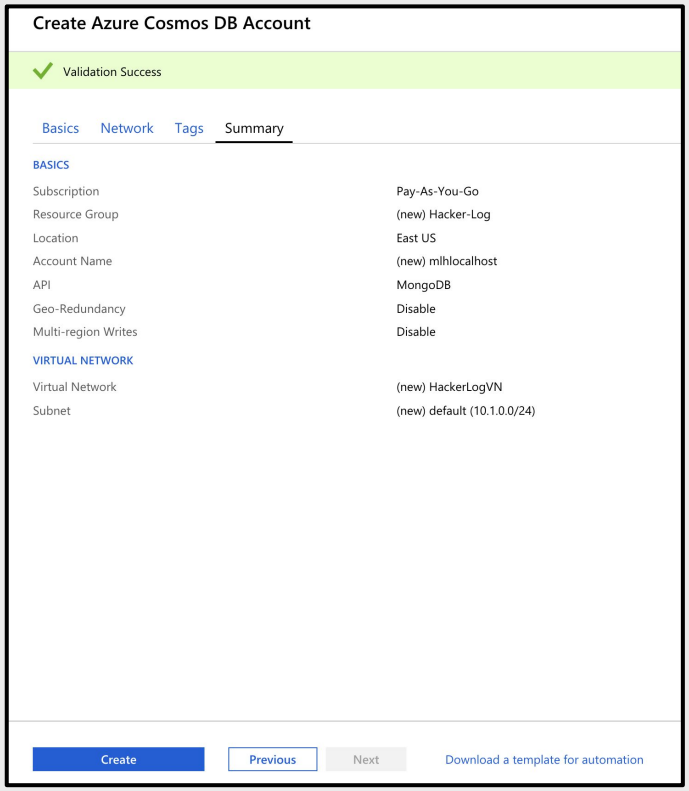
Create a Cosmos Resource

10. Click **Create a new virtual network**.
11. Name it HackerLogVN.
12. **Click OK**.
13. Click **Review + create**

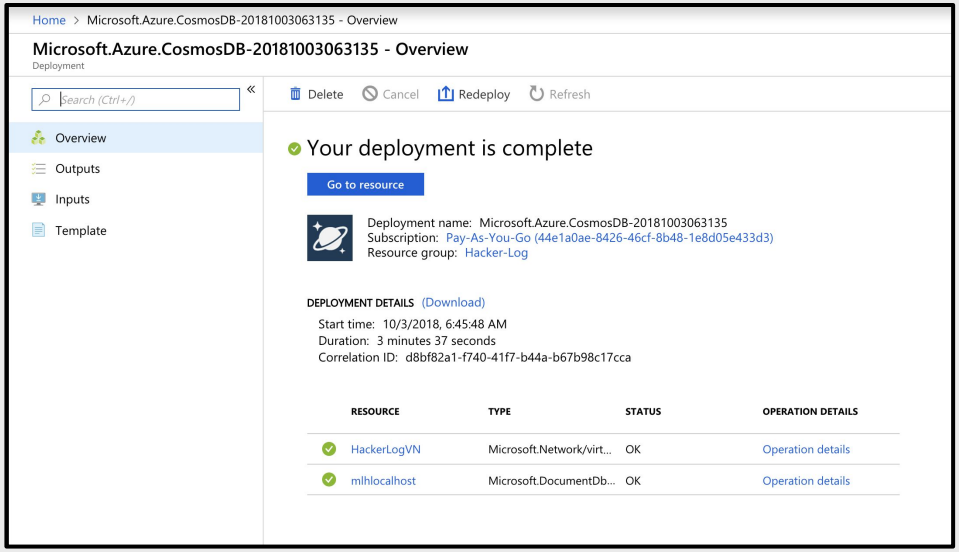


Create a Cosmos Resource

14. Click **Create** and wait for your database to deploy!

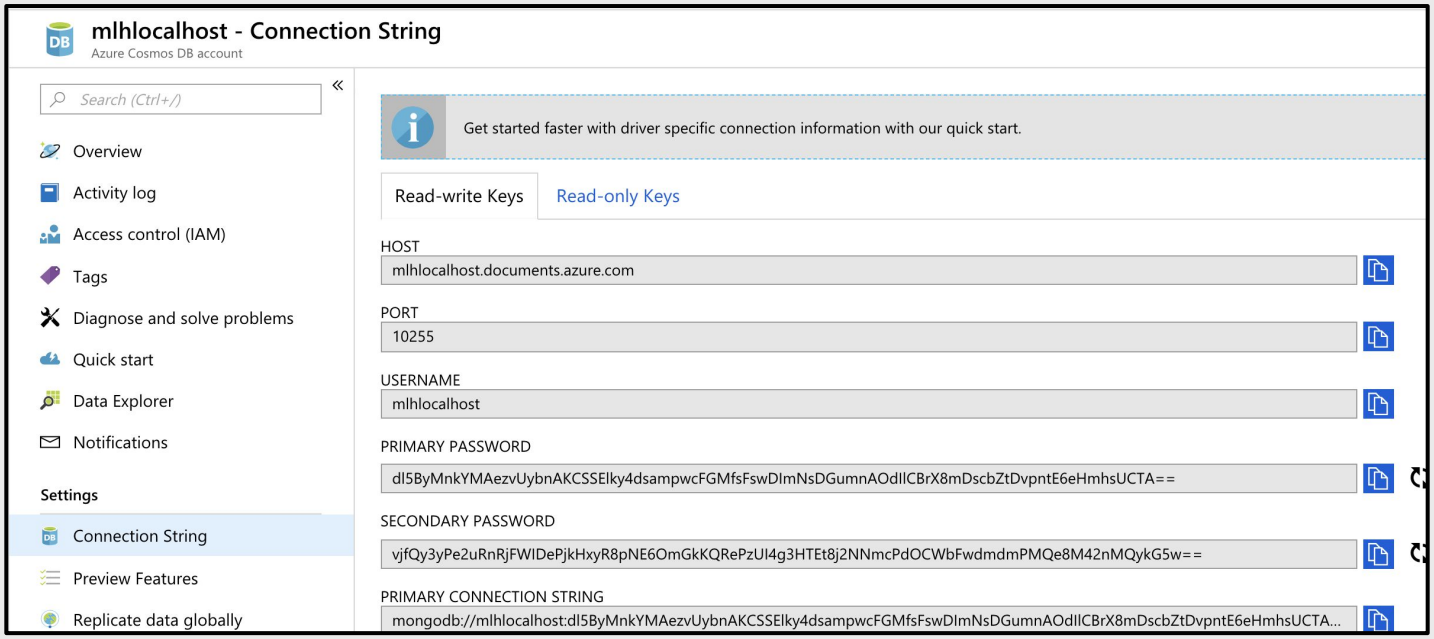


15. When the deploy is done, click **Go to resource**.



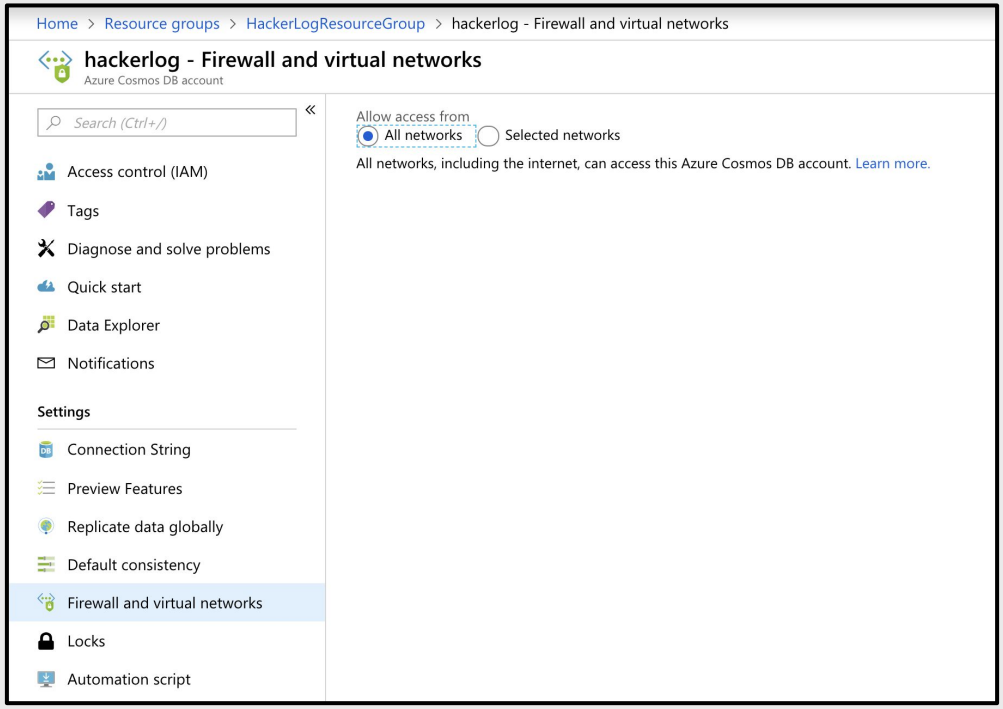
Create a Cosmos Resource

16. Open the **Connection String** menu, under the **Settings** category.
17. Copy PRIMARY CONNECT STRING. You'll need it later.



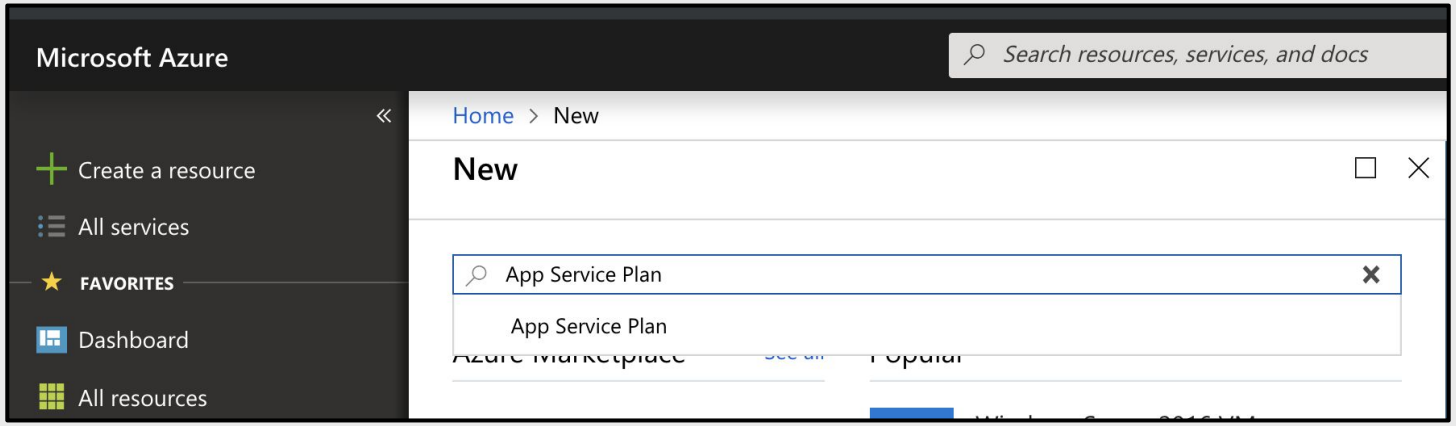
Create a Cosmos Resource

18. Go to the **Firewall and virtual networks** tab under the same **Settings** category.
19. For the purposes of this workshop, select **All networks** under **Allow access from:** (though you wouldn't necessarily want to do this in a production app).
20. Save!

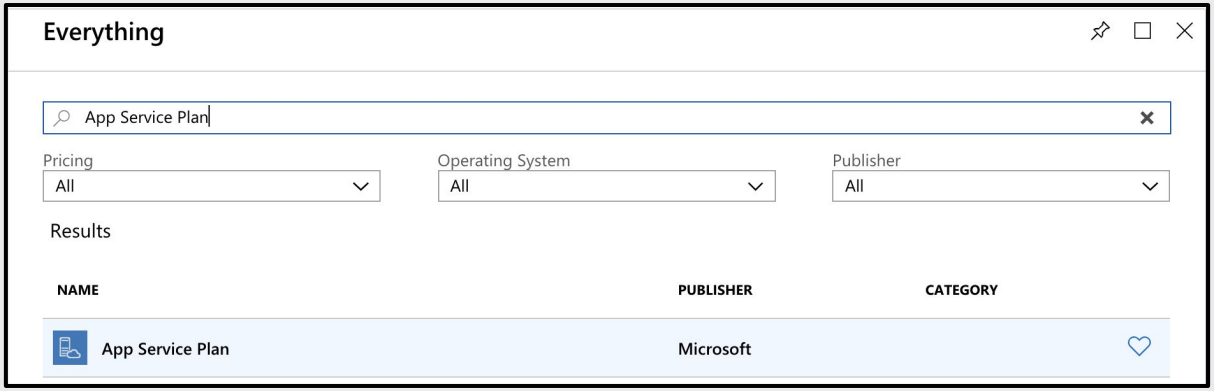


Create an App Service Plan

1. Return to portal.azure.com
2. Select **Create a Resource** then search for "App Service Plan".



3. Select **App Service Plan** then click **Create** in the panel that opens.



Create an App Service Plan

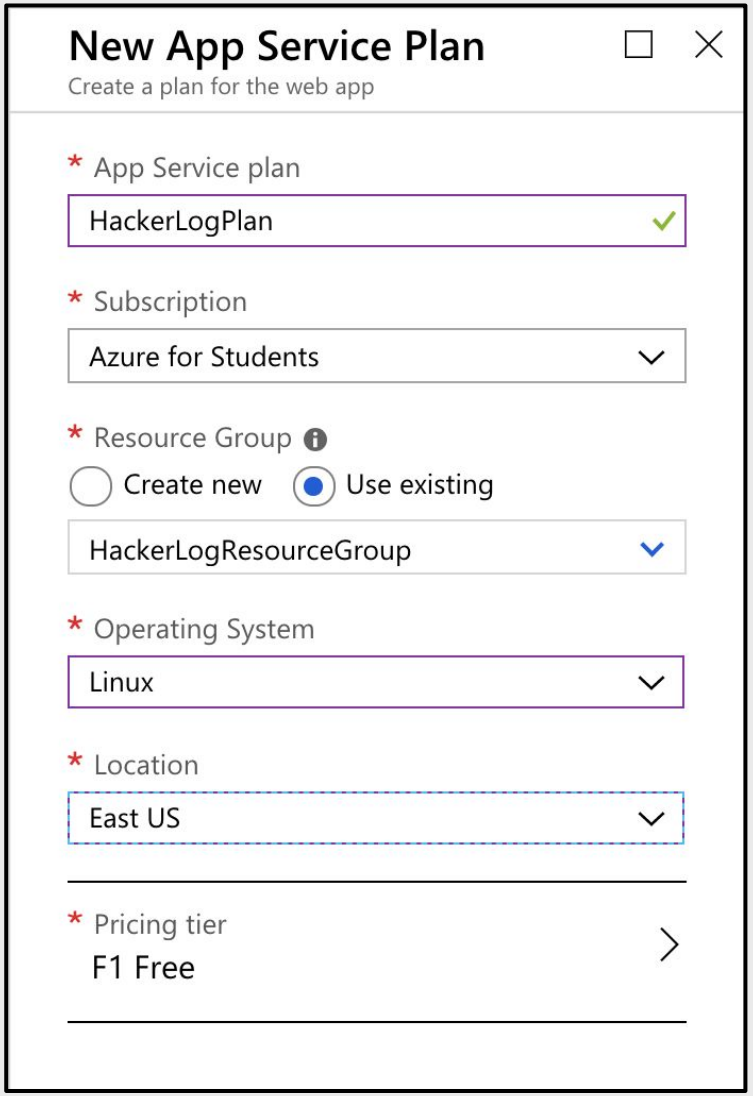
4. In the panel that opens, name your service plan HackerLogPlan.

5. Select the Resource Group you made previously.

6. Select **Linux** as your operating system.

7. Select **East US**.

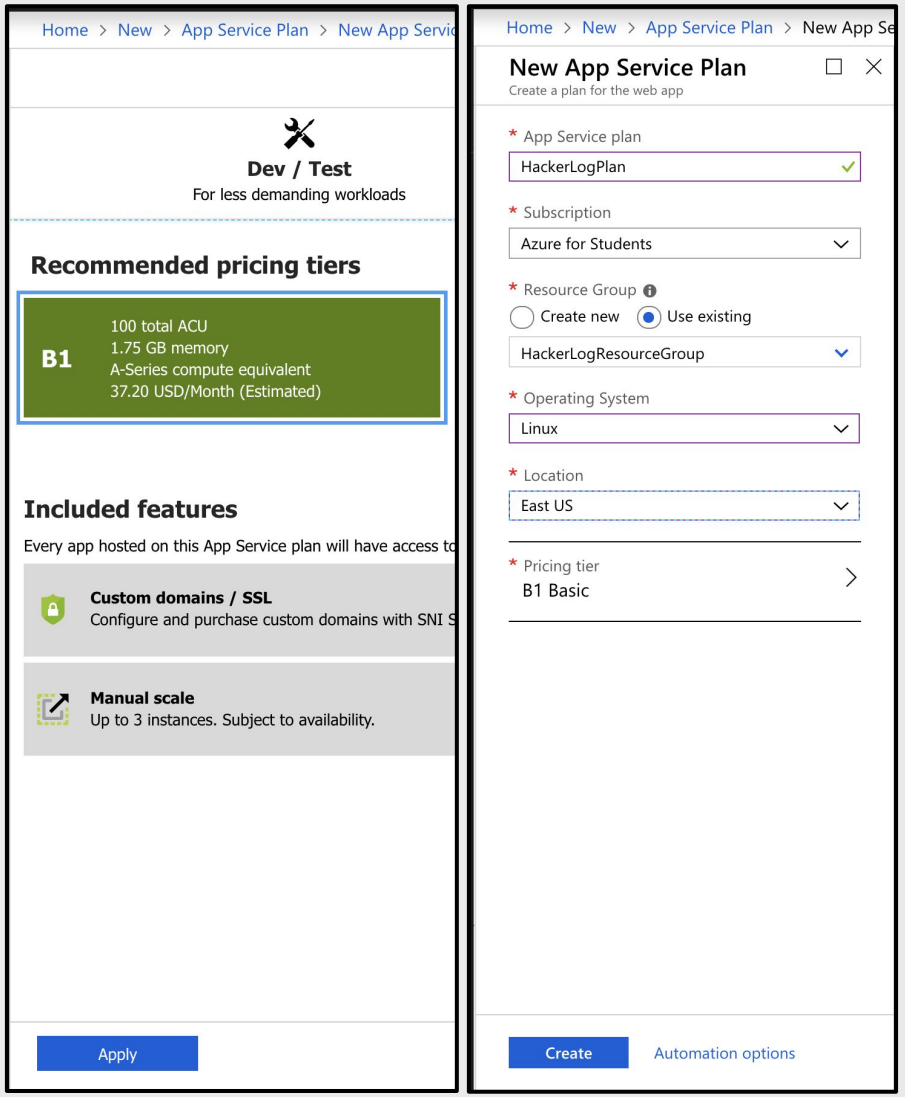
8. Click **Pricing tier**.



Create an App Service Plan

9. Select **Dev/Test**.
10. Select **B1**.
11. Select **Apply**.
12. When the previous screen returns, select **Create**.

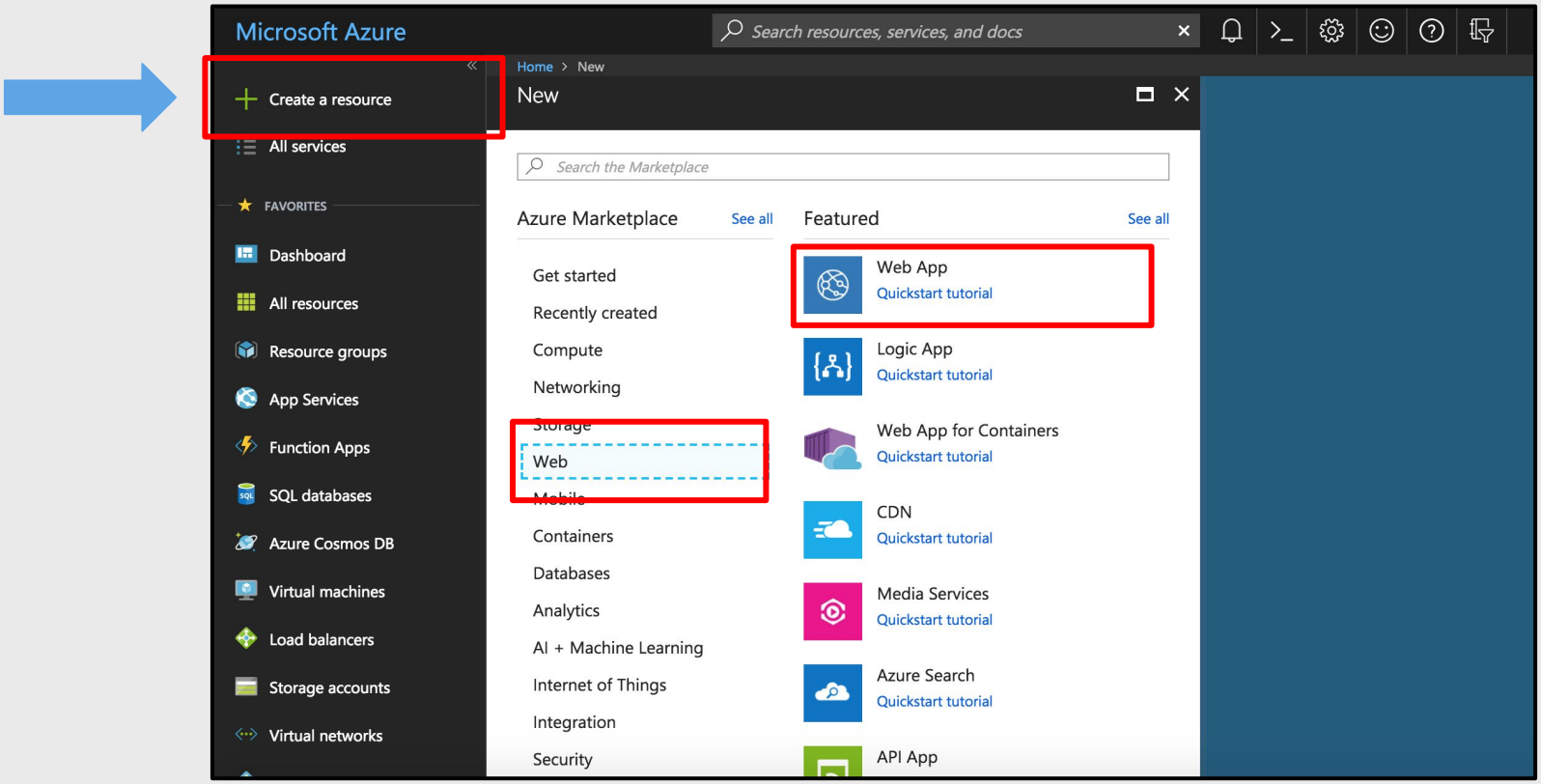
    Wait a minute or two for that to finish deploying, then we'll create the resource for the HackerLog Web App!



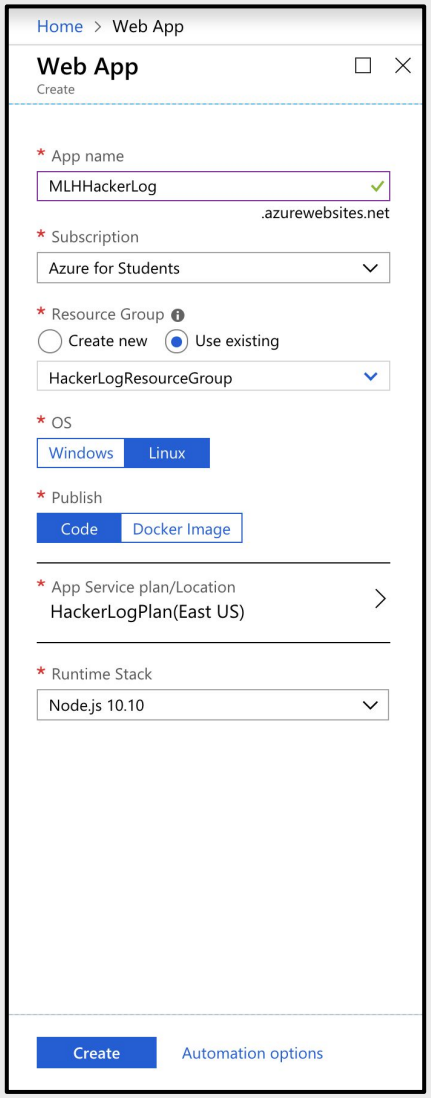
Deploy Your App to Microsoft Azure

Now, let's deploy the HackerLog app!

1. Navigate to your Azure dashboard and select **Create a resource** then **Web** then **Web App**.



Deploy Your App to Microsoft Azure



2. In the panel that opens, name your app something unique.

3. Select the Resource Group you made previously.

4. Select **Linux** as your operating system.

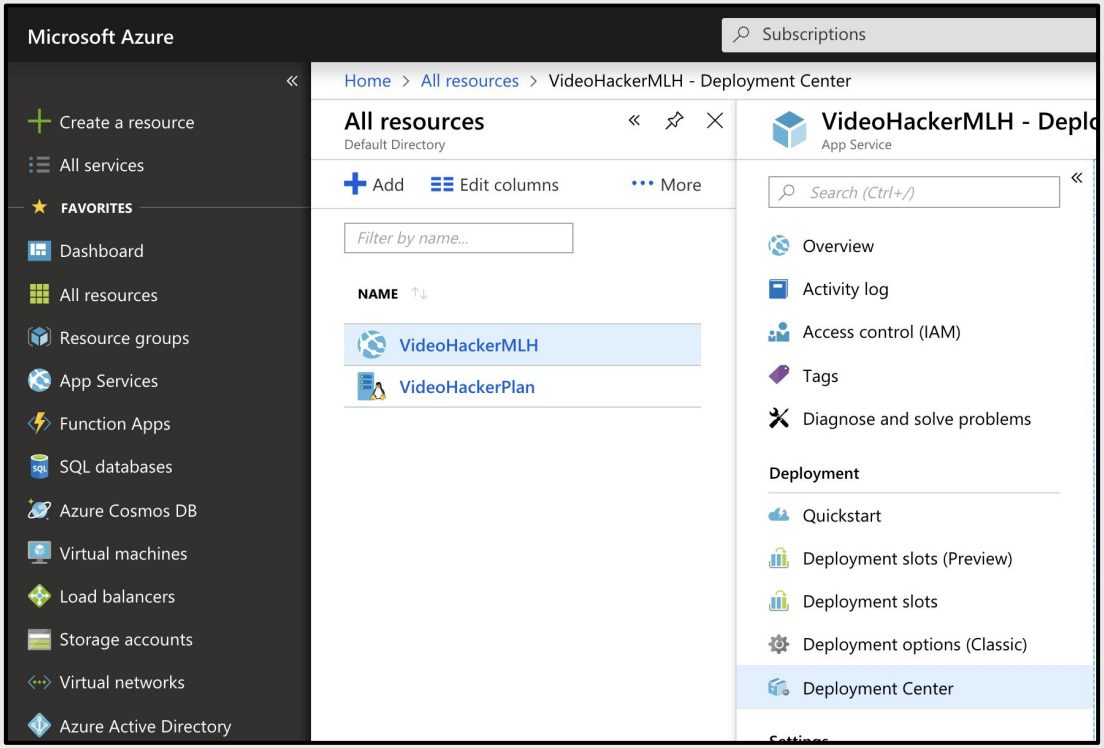
5. Select **Node.js 10.10** as your Runtime Stack.

6. Click **Create** at the bottom of the screen.

Wait a moment for the app to deploy, then follow the next steps.

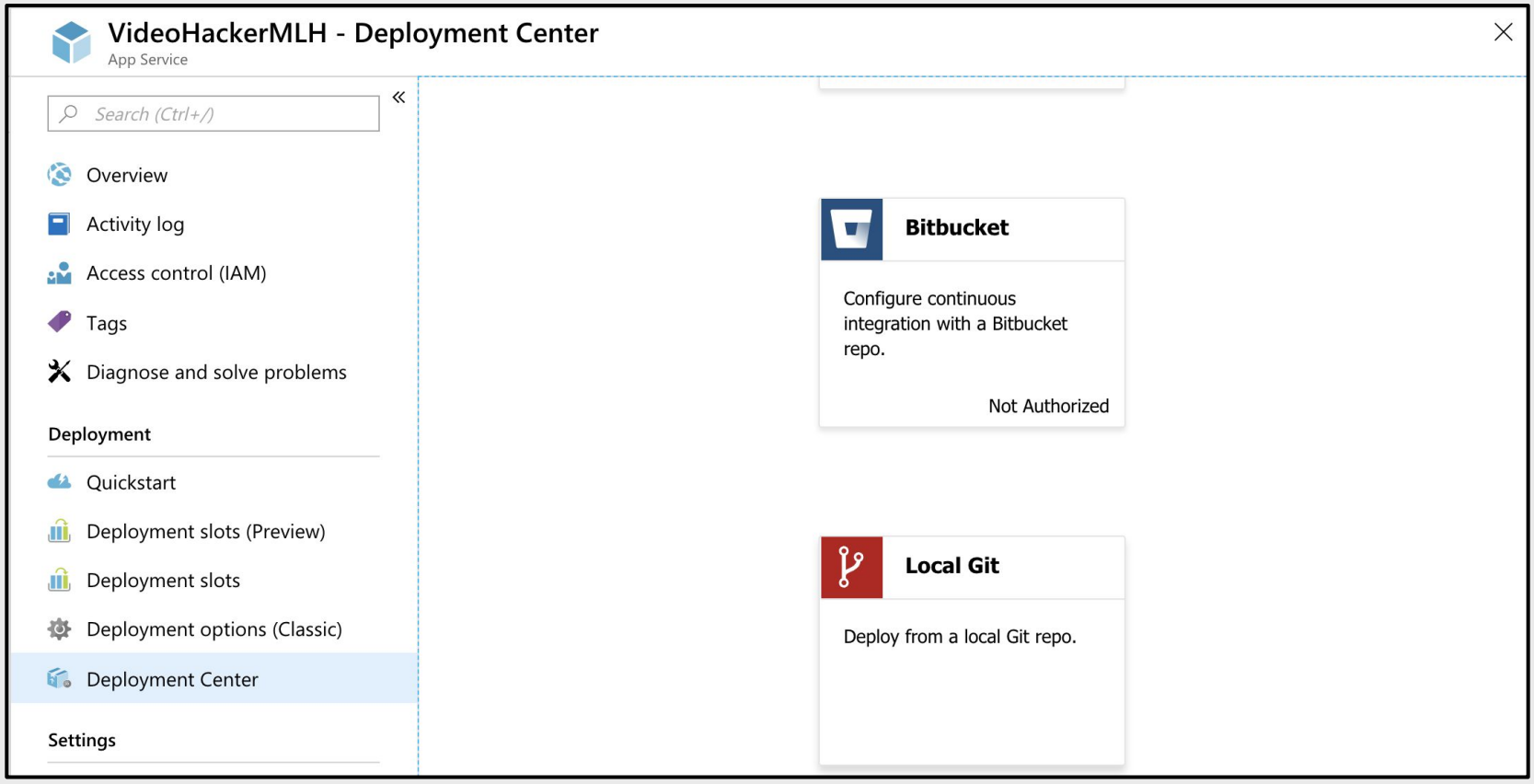
Deploy Your App to Microsoft Azure

1. On the left side of your Azure portal, select **All resources**.
2. Click the Web App you just made.
3. Select **Deployment Center**.



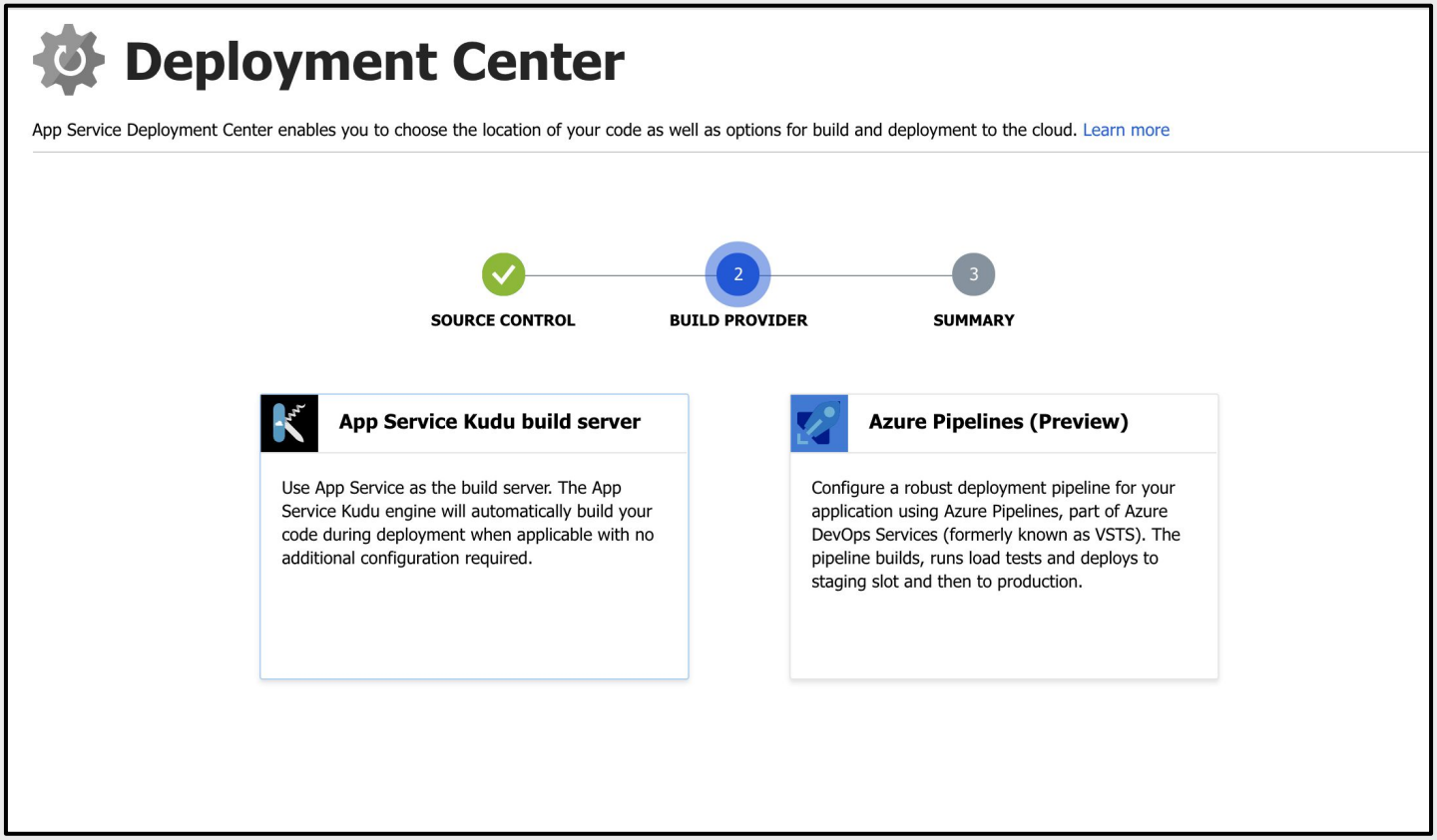
Deploy Your App to Microsoft Azure

4. Scroll to select **Local Git**. Select **Continue**.



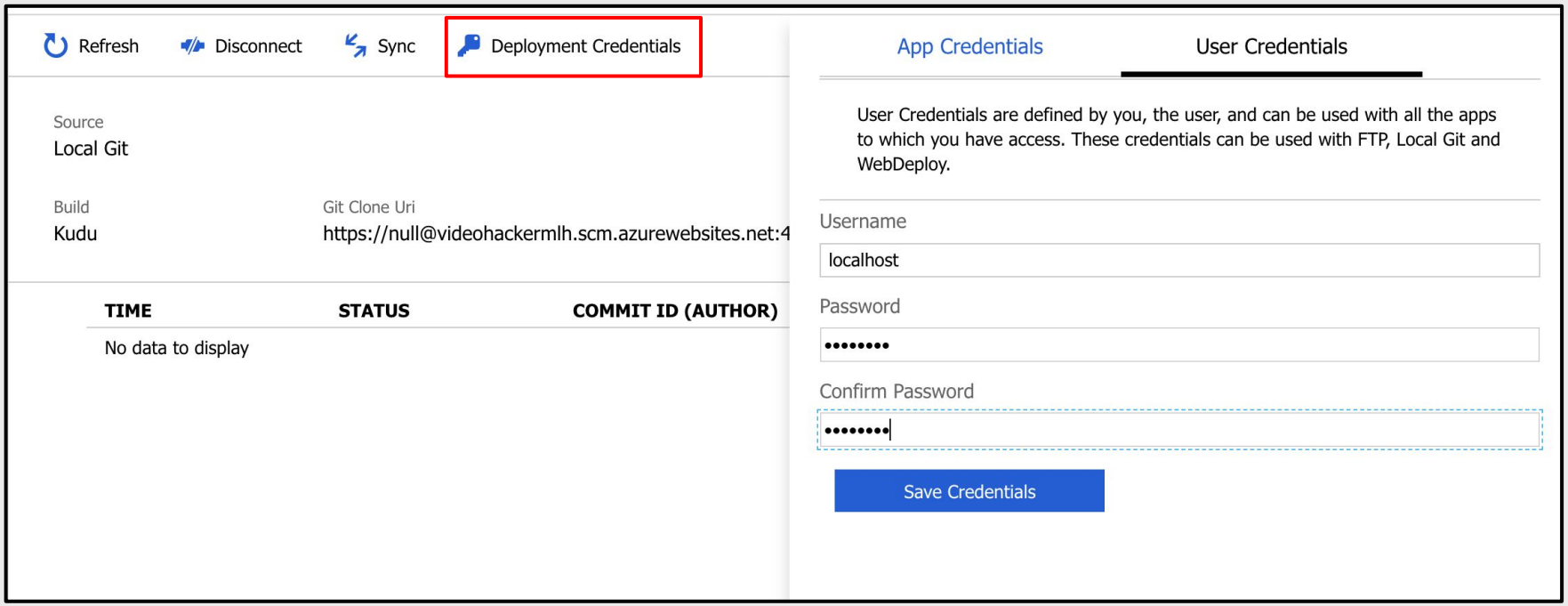
Deploy Your App to Microsoft Azure

5. Select App Service Kudu build server then select **Continue** then **Finish**.



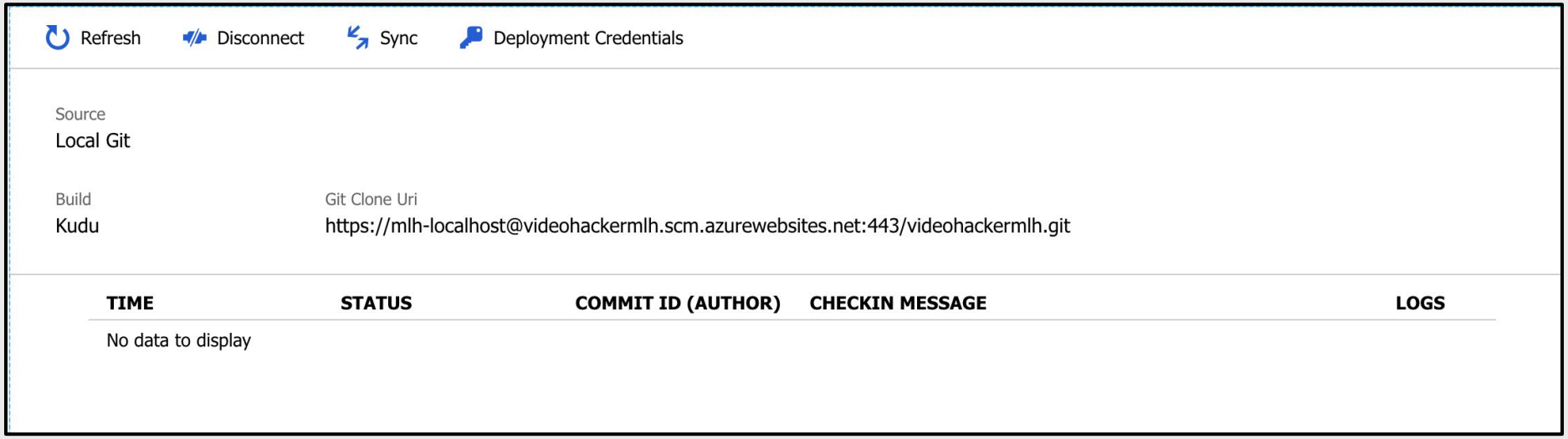
Deploy Your App to Microsoft Azure

6. Select **Deployment Credentials** then **User Credentials**.
7. Create a Username and Password.



Deploy Your App to Microsoft Azure

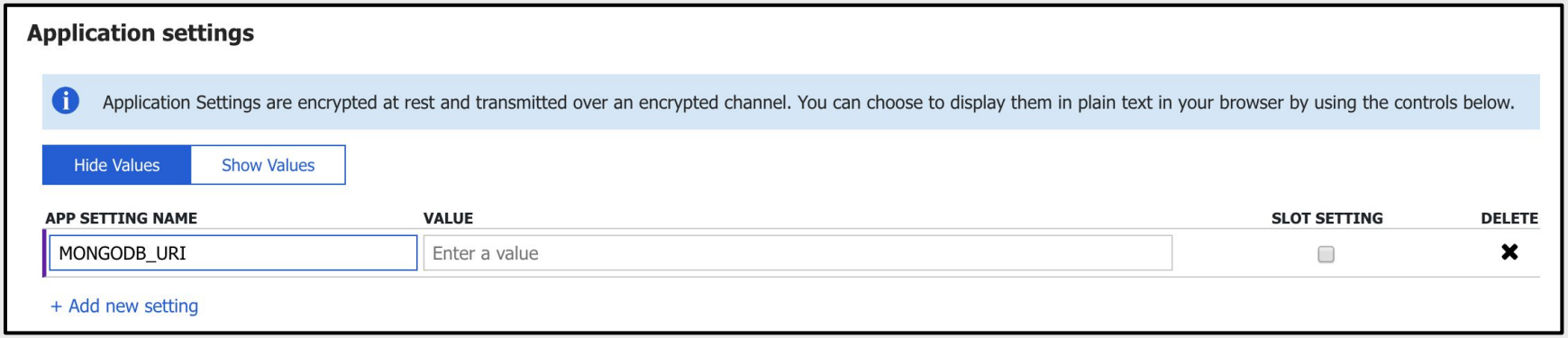
8. You will know the previous step was successful if the Uri has <your-new-username>@<rest of website>.



Deploy Your App to Microsoft Azure

9. Select **Application settings**.
10. Click **+ Add new setting**.
11. The name of the new setting is "MONGODB_URI"
12. The value of the new setting is the PRIMARY CONNECTION STRING you copied when you made the CosmosDB resource. This is how your app and the DB can talk to each other. It will look like this:

```
mongodb://<username>:<pimary_password>@<hostname>:<port>/?ssl=true&replicaSet=globaldb
```



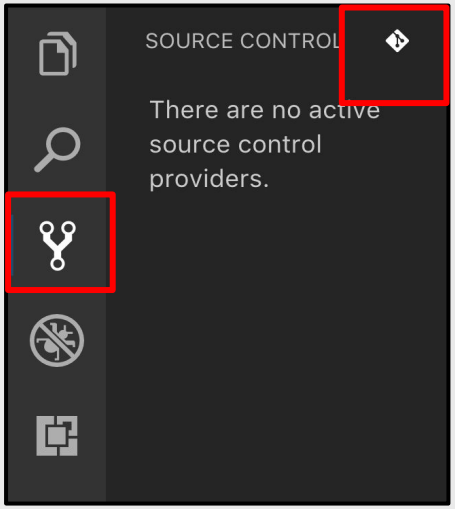
You just:

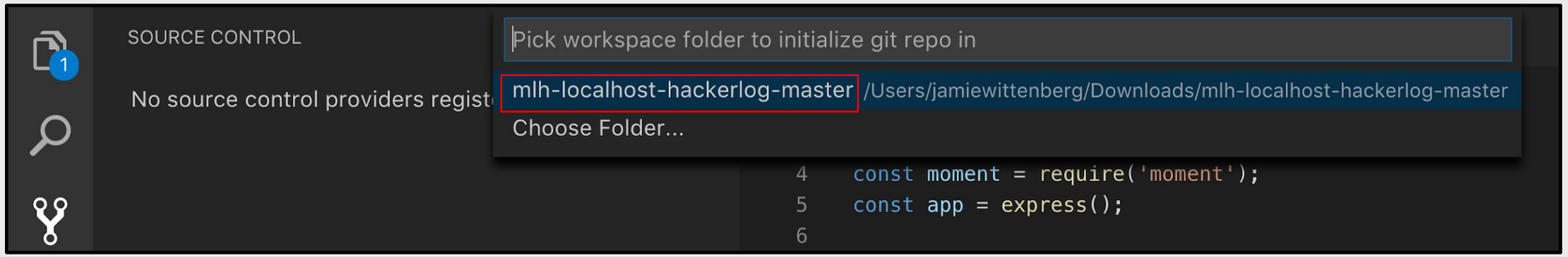
1. **Created a database on Microsoft Azure.**
2. **Created an App Service Plan.**
3. **Created a Node app environment to deploy your app to.**

Head back to Visual Studio Code for the rest!

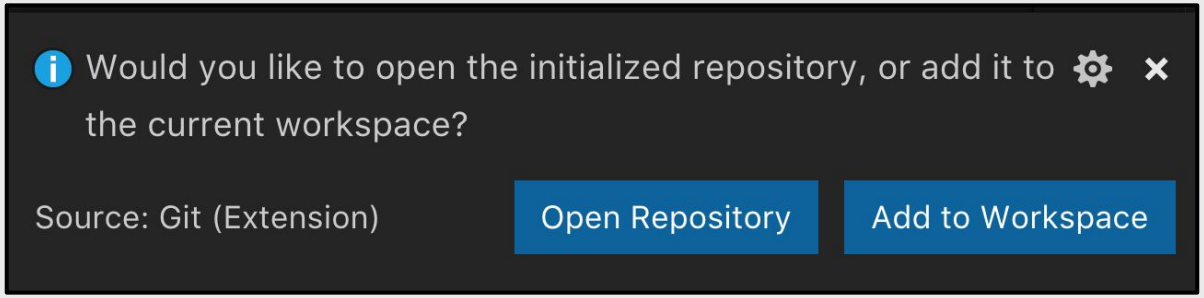
Deploy Your App to Microsoft Azure

1. In Visual Studio Code, click the source control symbol.
2. Then, click the symbol next to `SOURCE CONTROL`.
3. Be sure that the folder is **`mlh-localhost-hackerlog-master`** and click it.

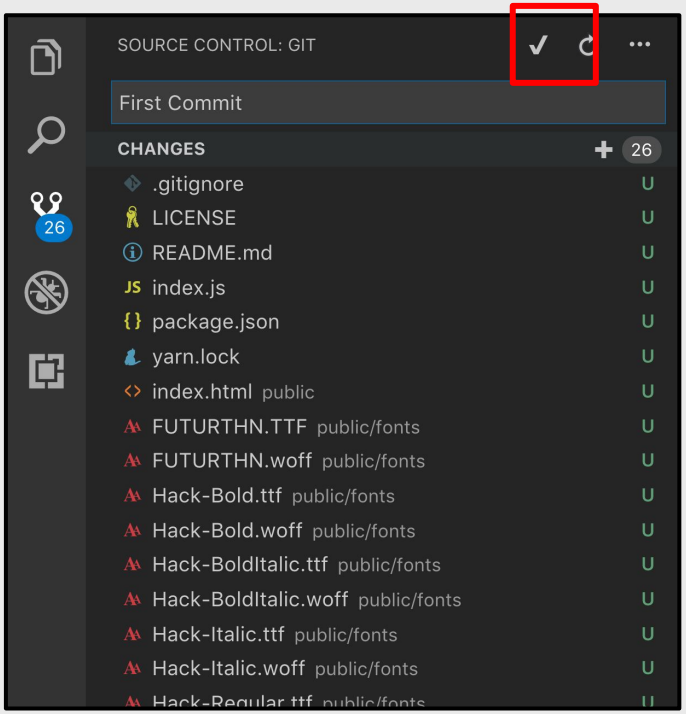




Deploy Your App to Microsoft Azure

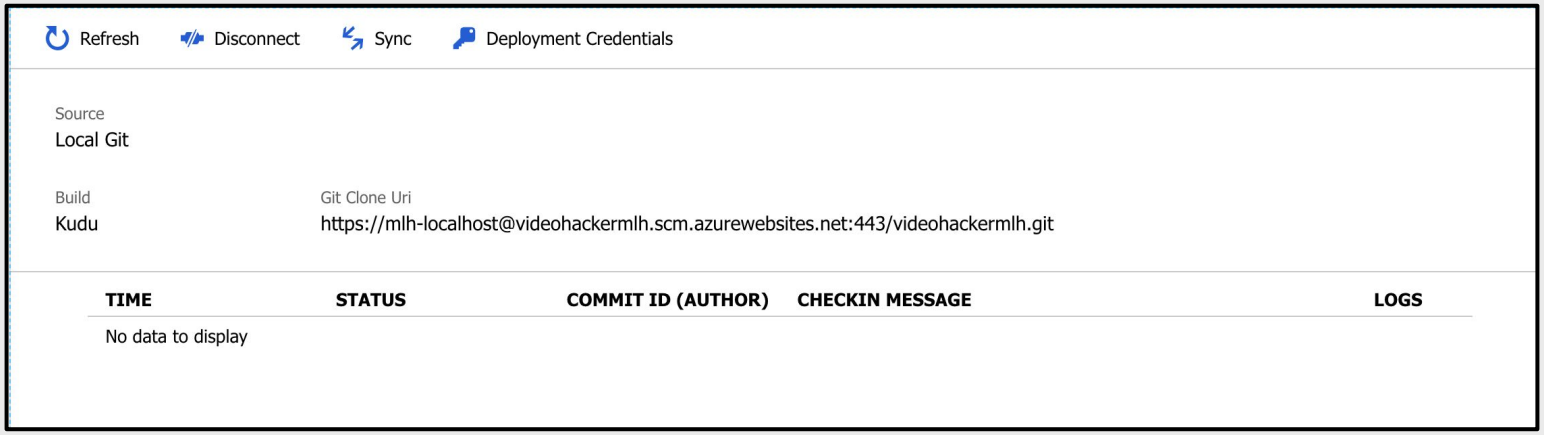


4. Select **Add to Workspace** if prompted.
5. Type "First Commit" above CHANGES.
6. Click the checkmark.
7. Click **Yes** if you get a modal asking about staging changes.

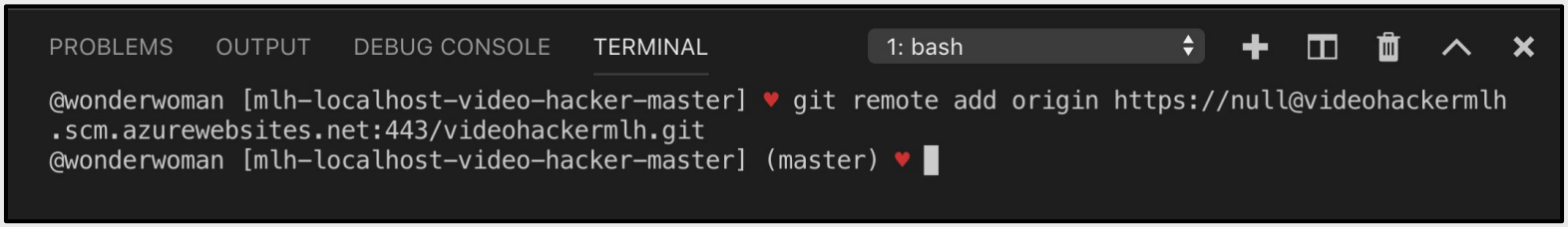


Deploy Your App to Microsoft Azure

8. In your Deployment Center tab, copy the Git clone Uri.

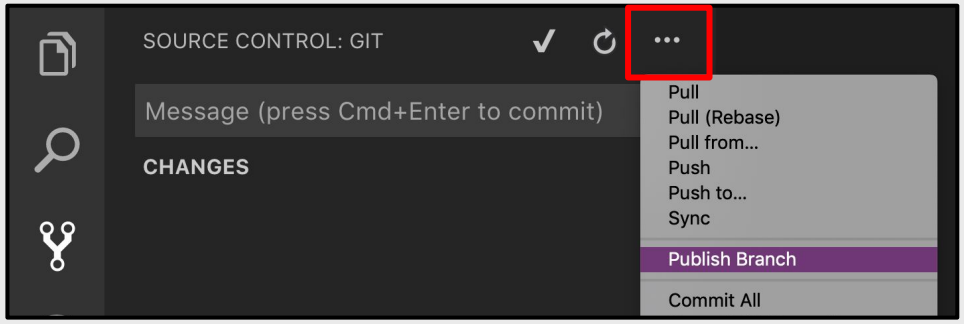


9. In your terminal, type **`git remote add origin`** and paste what you just copied.

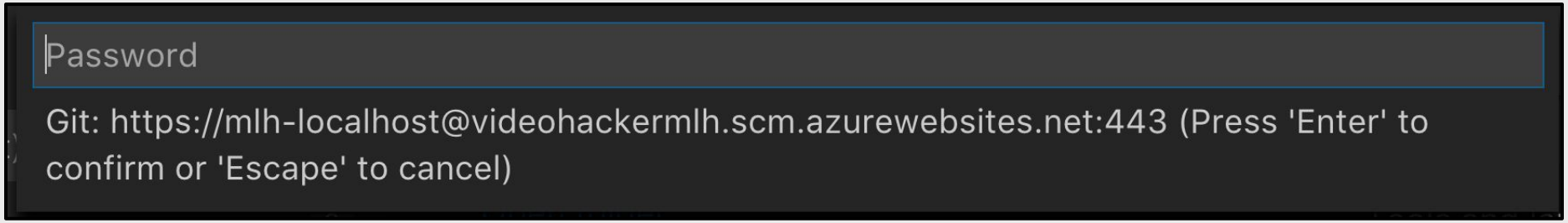


Deploy Your App to Microsoft Azure

10. Click the ellipses **(. . .)** near SOURCE CONTROL.
11. Select **Publish Branch**.



12. Enter the Password you created when prompted.



Deploy Your App to Microsoft Azure

13. Back in Azure Deployment Centre, you can see that your deploy is in progress!

Deploy Your App to Microsoft Azure

14. When the build has finished, select Overview.
15. Click the URL for your web app, and try it live!

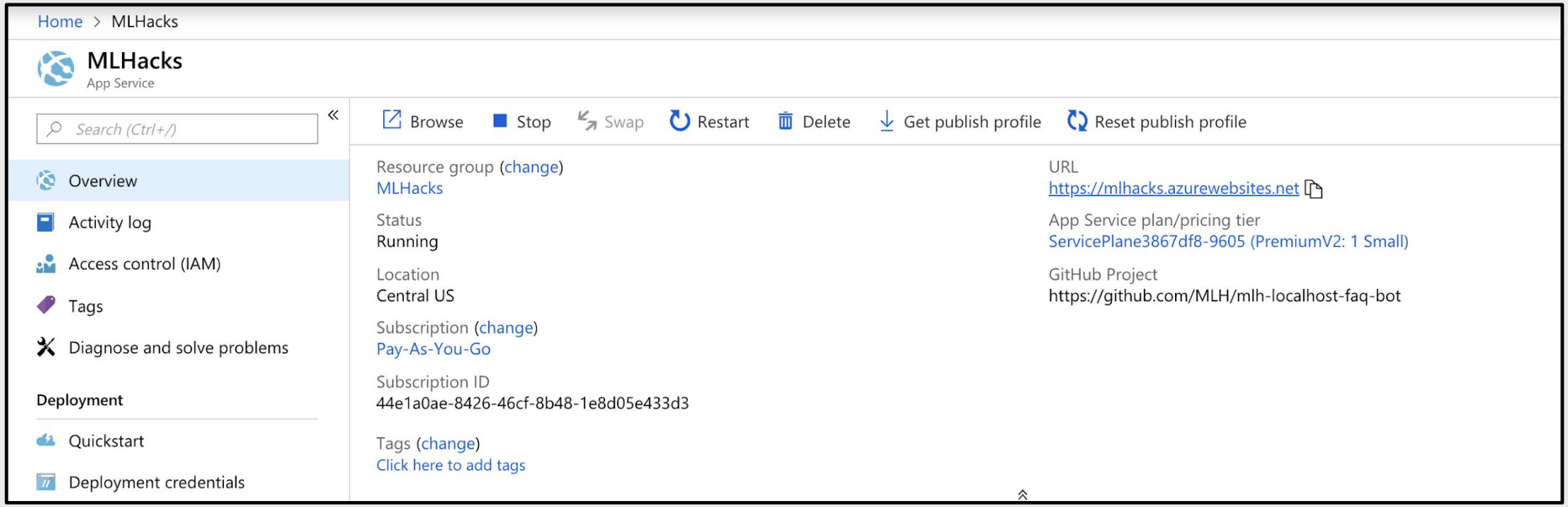


Table of Contents

Let's recap quickly...

1. Node.js is a server-side JavaScript runtime ideal for high performance, scalable network applications.
2. Express is a Node.js framework for back end web development.
3. Developing Web Apps on Azure is fast.

What did you learn today?

We created a fun quiz to test your knowledge and see what you learned from this workshop.

http://mlhlocal.host/quiz

Table of Contents

Keep Learning: Practice Problems for Later

Extra Practice Problem 1:

Add the password as an application setting and remove it from your code.

Extra Practice Problem 2:

Use the Azure Content Moderator to prevent people from posting negative content to your app!

Learning shouldn't stop when the workshop ends...

Check your email for access to:

- These workshop slides
- Practice problems to keep learning
- Deeper dives into key topics
- Instructions to join the community
- More opportunities from MLH!

Workshop

Hackerlog: Build and Deploy Node.js Apps with Microsoft Azure

v.0.0.1