

NAME

GENERIC NAME

UPDATE

PASSWORD

Workshop

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

SUSPENDISSE CURSUS TRISTIQUE MI NEC VESTIBULUM

9:31 AM

 localhost

IN DAPIBUS AUCTOR LACOS

5:20 PM

QUIS CONVALLIS NISL VARIUS AC

4:19 PM



MORBI A IACULIS SEM

3:08 PM

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



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

-  **1.** Introduction to Web Development
- 2.** Microsoft Azure and Visual Studio
- 3.** Node.js and Express
- 4.** Deploy Your Web App!
- 5.** Review & Quiz
- 6.** Next Steps

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:

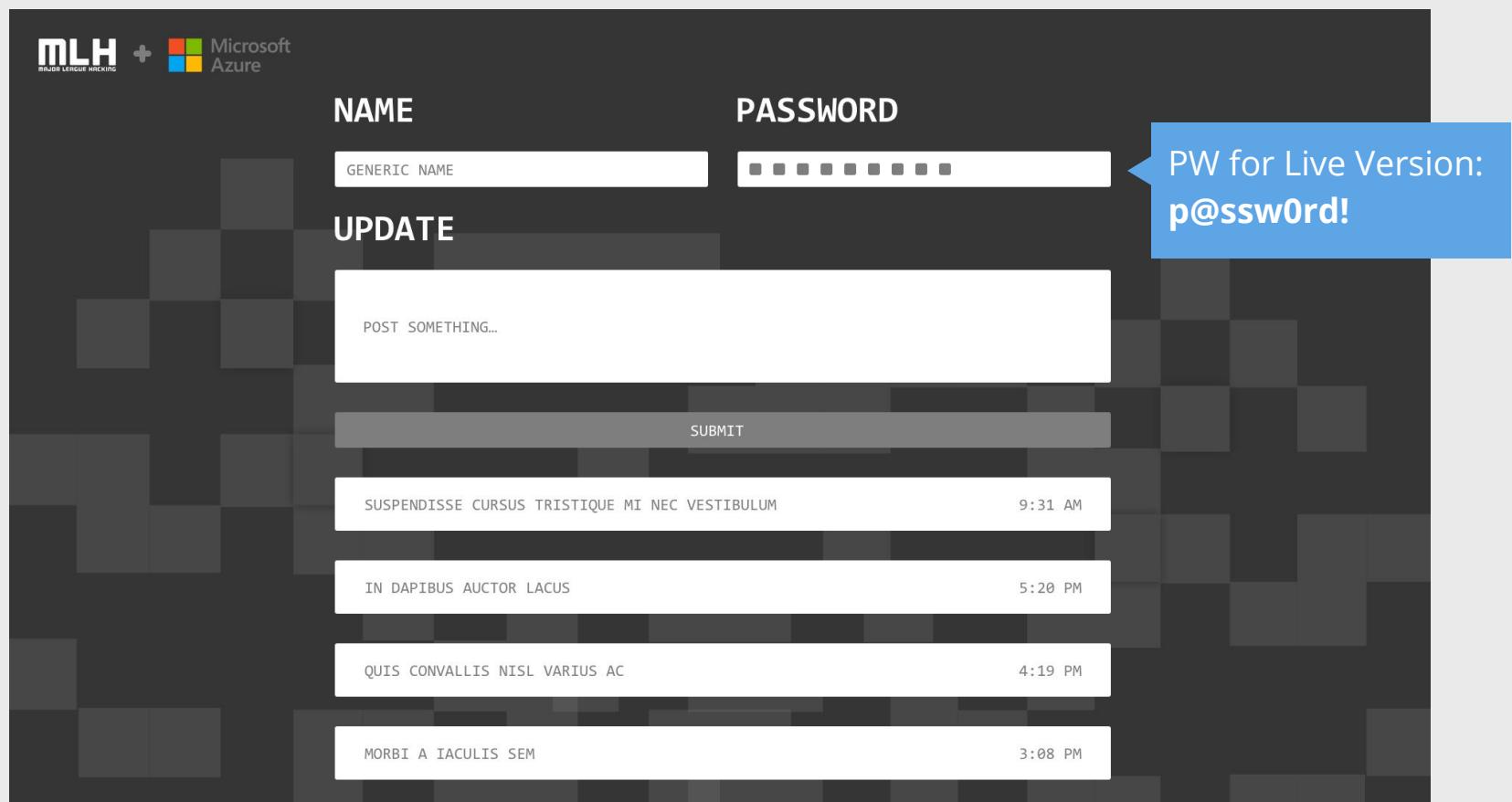
- User-hidden functionality.
- Basically any language, but mostly Node.js, Ruby and Python

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



Client Side
(Front End)

2

Browser sends request to app's server,
which is hosted on Microsoft Azure

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

Server checks password and adds updates to database

Server updates website with changes now visible to users

3



6



7

Client Side (Front-End)



1

User types
app URL

Server returns app
pages to display

2

3 User enters
password & updates

4 Server updates the
website with new changes

5

Server Side (Back-End)



4

5 Server checks
password and adds
updates to DB

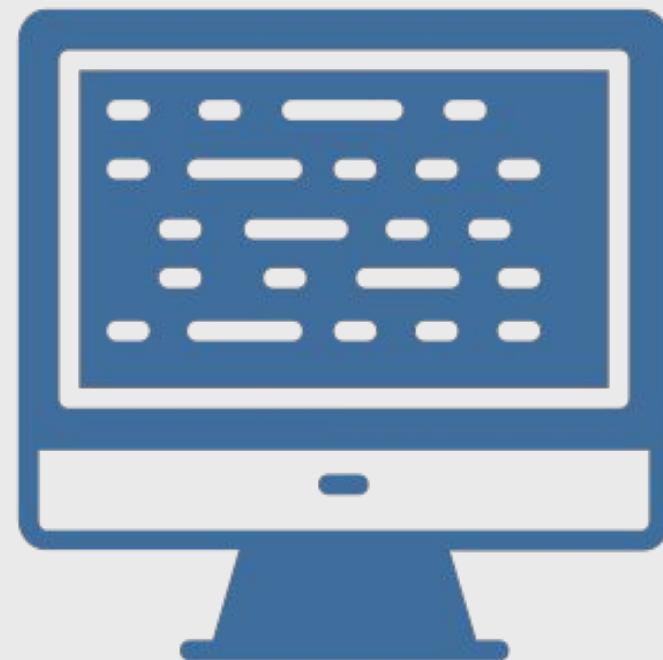


Database

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

- 1. Introduction to Web Development**
- 2. Microsoft Azure and Visual Studio**
- 3. Node.js and Express**
- 4. Deploy Your Web App!**
- 5. Review & Quiz**
- 6. Next Steps**

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:

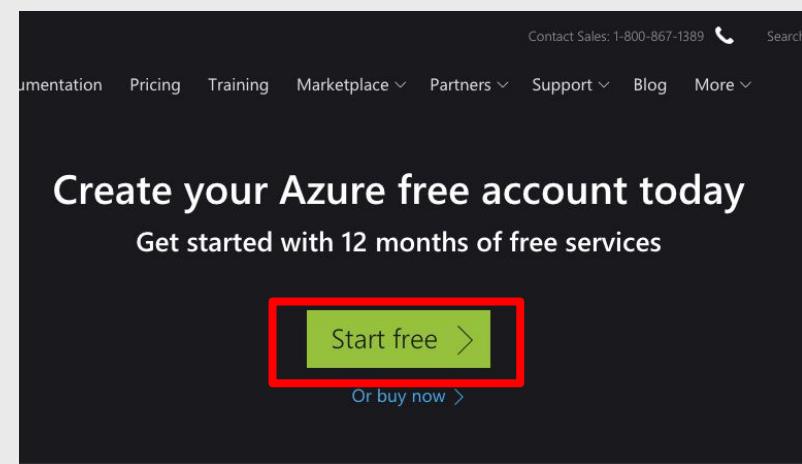
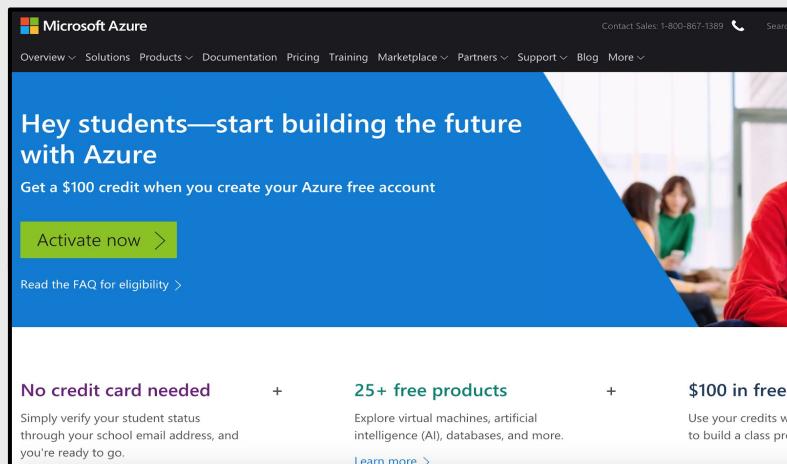
**We recommend completing this entire
workshop in an Incognito Window in
Chrome.**

Create Microsoft Azure Account

http://mlhlocal.host/azure-student

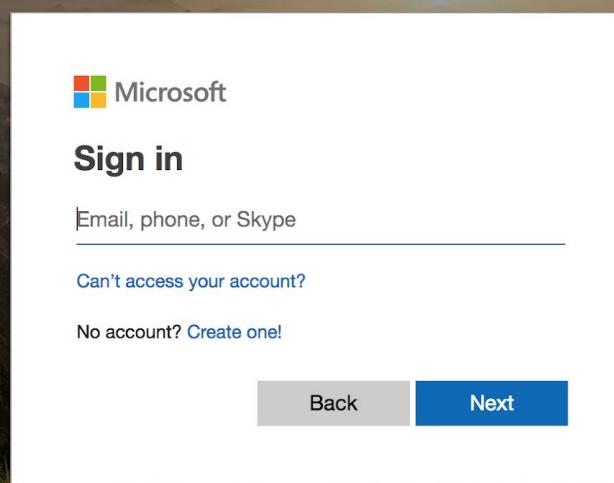
http://mlhlocal.host/azure-nonstudent

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



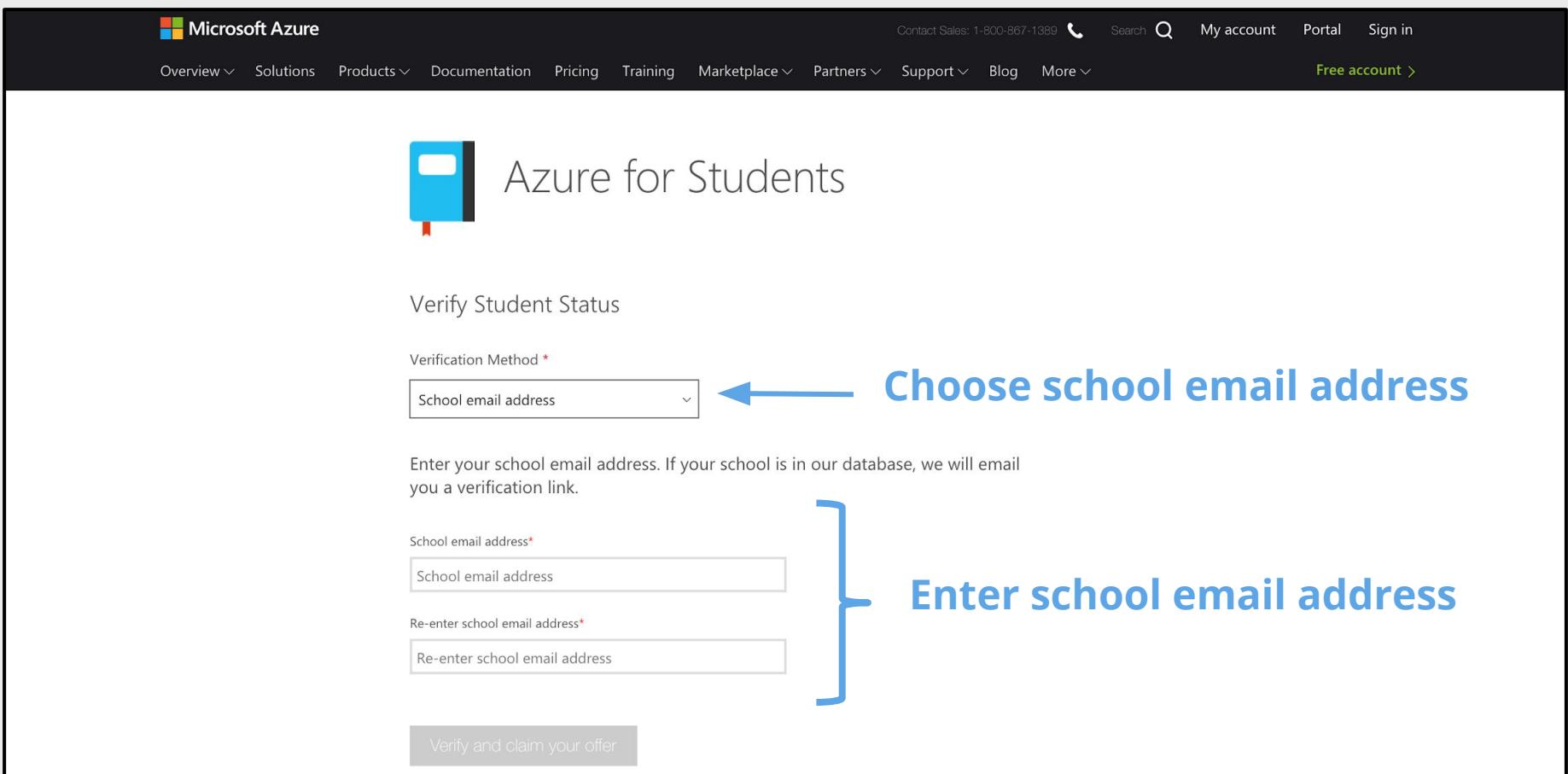
Create Microsoft Azure Account

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.



Azure for Students

Verify Student Status

Verification Method *

School email address

← Choose school email address

Enter your school email address. If your school is in our database, we will email you a verification link.

School email address*

School email address

Re-enter school email address*

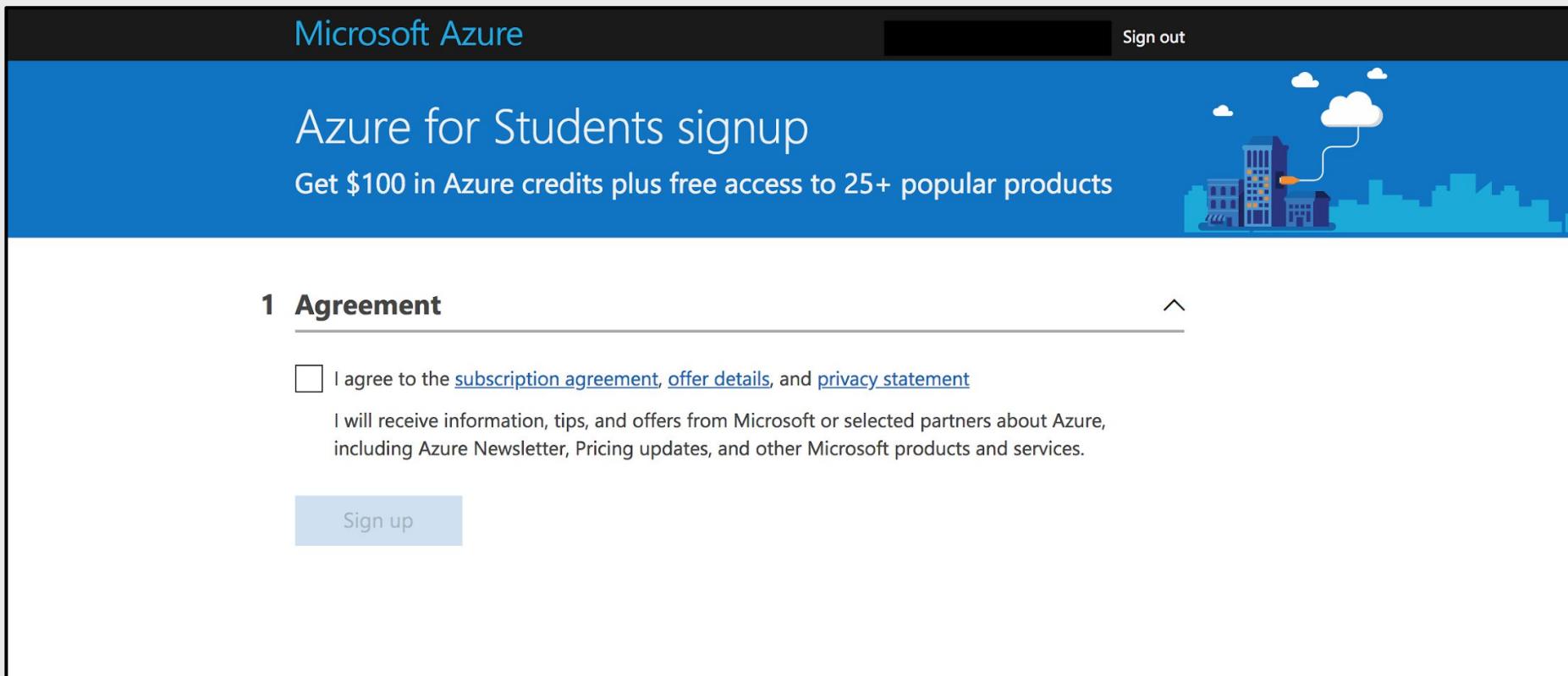
Re-enter school email address

Enter school email address

Verify and claim your offer

Create Microsoft Azure Account

Accept the terms of service and start using Microsoft Azure!



The screenshot shows the Microsoft Azure for Students signup page. At the top, there's a navigation bar with the Microsoft Azure logo and a "Sign out" link. The main header reads "Azure for Students signup" and "Get \$100 in Azure credits plus free access to 25+ popular products". To the right is a decorative illustration of a city skyline with clouds and a network connection line. Below the header, a section titled "1 Agreement" contains a checkbox followed by the text: "I agree to the [subscription agreement](#), [offer details](#), and [privacy statement](#)". A explanatory text block follows: "I will receive information, tips, and offers from Microsoft or selected partners about Azure, including Azure Newsletter, Pricing updates, and other Microsoft products and services." At the bottom left is a blue "Sign up" button.

Microsoft Azure

Sign out

Azure for Students signup

Get \$100 in Azure credits plus free access to 25+ popular products

1 Agreement

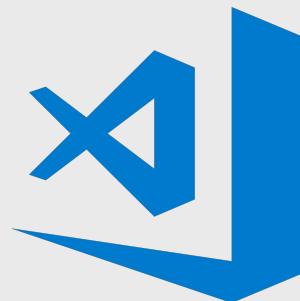
I agree to the [subscription agreement](#), [offer details](#), and [privacy statement](#)

I will receive information, tips, and offers from Microsoft or selected partners about Azure, including Azure Newsletter, Pricing updates, and other Microsoft products and services.

Sign up

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



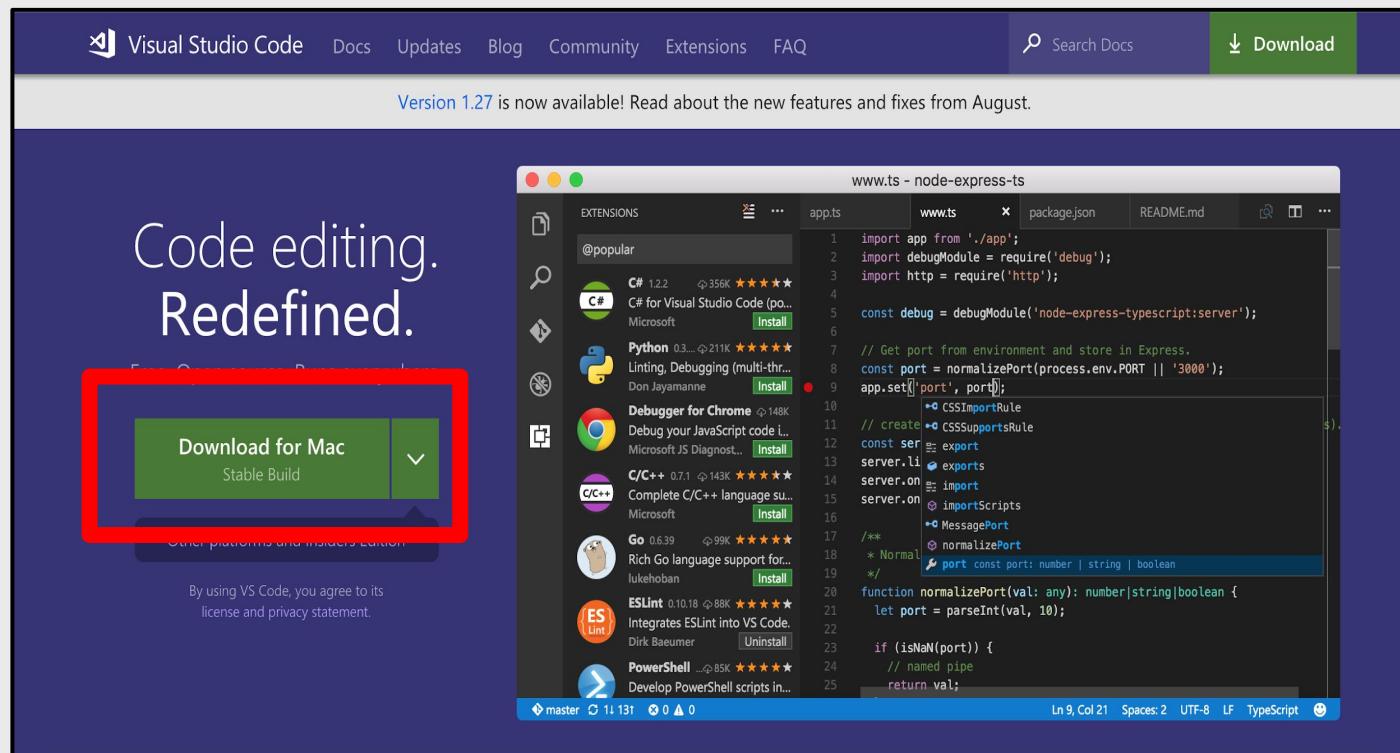
Visual Studio Code is a simple, code-centric integrated development environment (IDE).

- 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

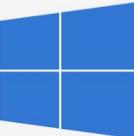
<http://mlhlocal.host/vscode>

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)



 Windows
Windows 7, 8, 10

User Installer 64 bit 32 bit
System Installer 64 bit 32 bit
.zip 64 bit 32 bit



 .deb
Debian, Ubuntu

 .rpm
Red Hat, Fedora, SUSE

.deb 64 bit 32 bit
.rpm 64 bit 32 bit
.tar.gz 64 bit 32 bit



 Mac
macOS 10.9+

Want new features sooner?
Get the [Insiders build](#) instead.

For PC

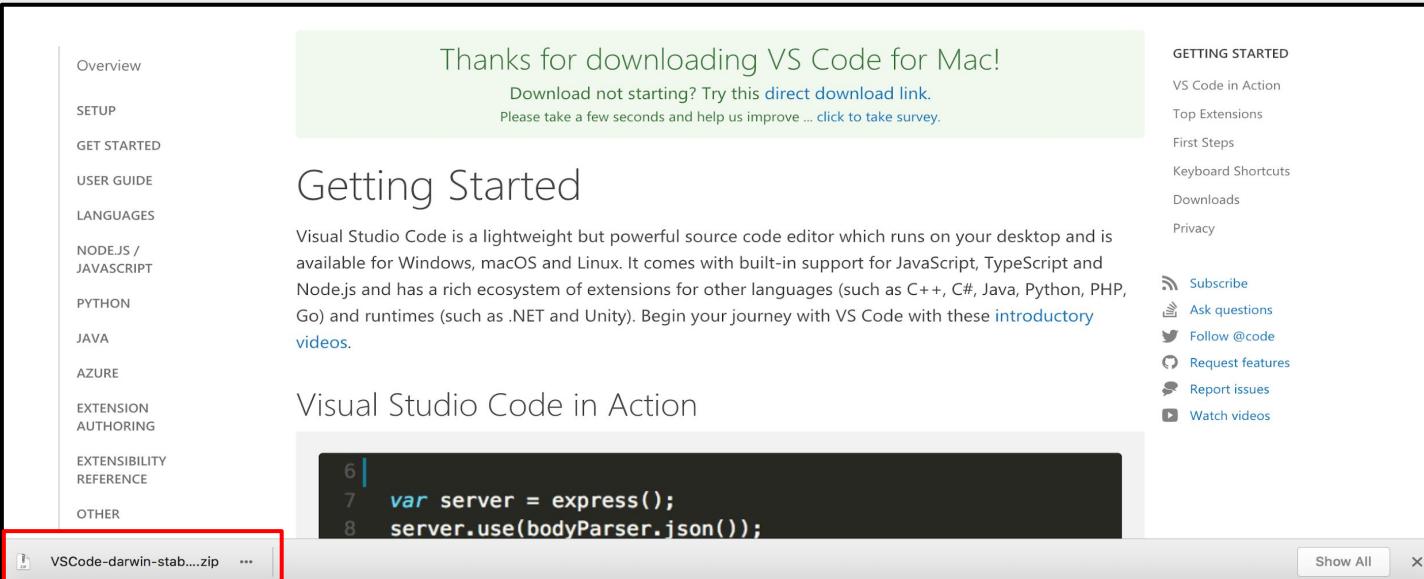
For Mac

LICENSE AND PRIVACY TERMS

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...



The screenshot shows the Visual Studio Code download page for Mac. The main content area features a green banner with the text "Thanks for downloading VS Code for Mac!", a direct download link, and a survey invitation. Below the banner, the "Getting Started" section introduces VS Code as a lightweight editor for Windows, macOS, and Linux, supporting various languages and extensions. The "Visual Studio Code in Action" section shows a code editor window with some JavaScript code. On the left, a sidebar lists categories like Overview, SETUP, GET STARTED, USER GUIDE, LANGUAGES, NODEJS / JAVASCRIPT, PYTHON, JAVA, AZURE, EXTENSION AUTHORIZING, EXTENSIBILITY REFERENCE, and OTHER. On the right, there's a "GETTING STARTED" sidebar with links to VS Code in Action, Top Extensions, First Steps, Keyboard Shortcuts, Downloads, and Privacy. At the bottom, social media links for subscribing, asking questions, following @code, requesting features, reporting issues, and watching videos are provided. A blue arrow points to the download link at the bottom of the page, which is highlighted with a red box.

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

 VSCode-darwin-stable.zip	Today at 1:34 PM	70.7 MB	ZIP archive
--	------------------	---------	-------------

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

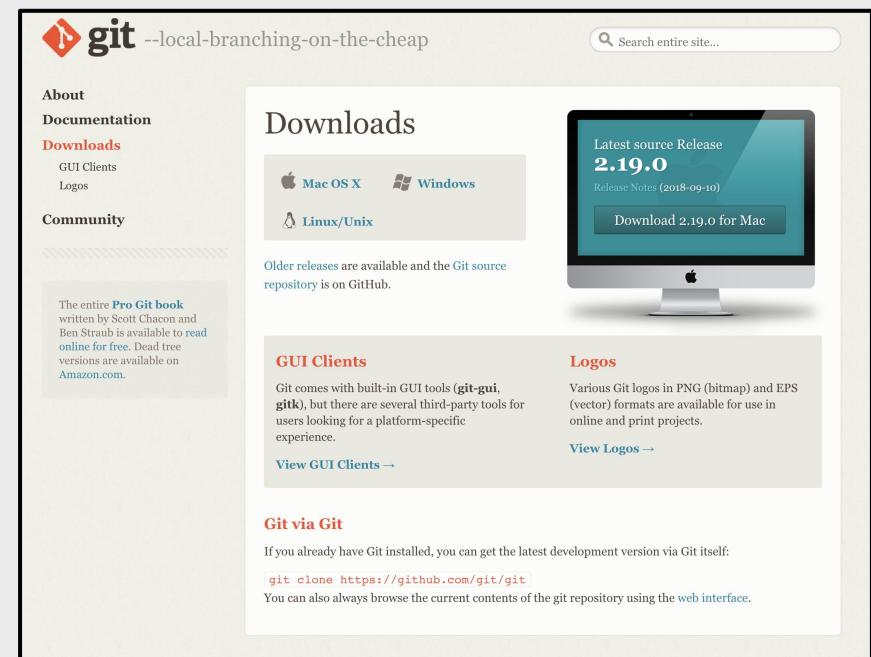
Let's get set up with Git!

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>

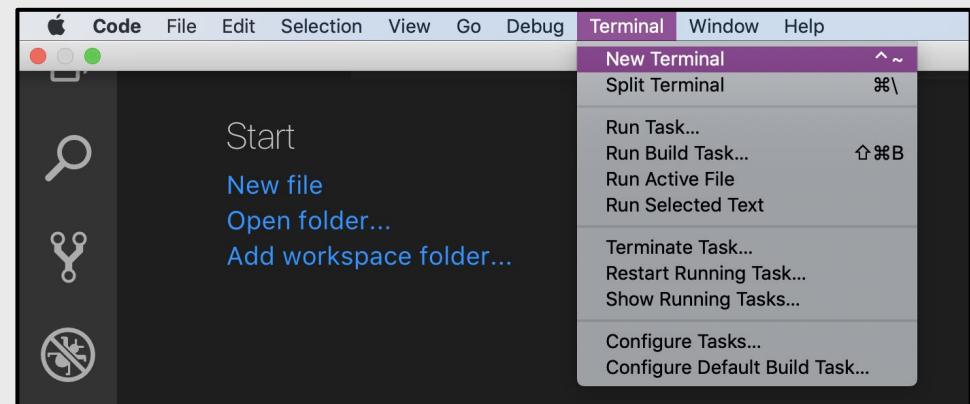
1. Visit the URL above.
2. Select your operating system.
3. Follow the prompts to install Git!
4. If Git asks you which default editor to use, we recommend VSCode!



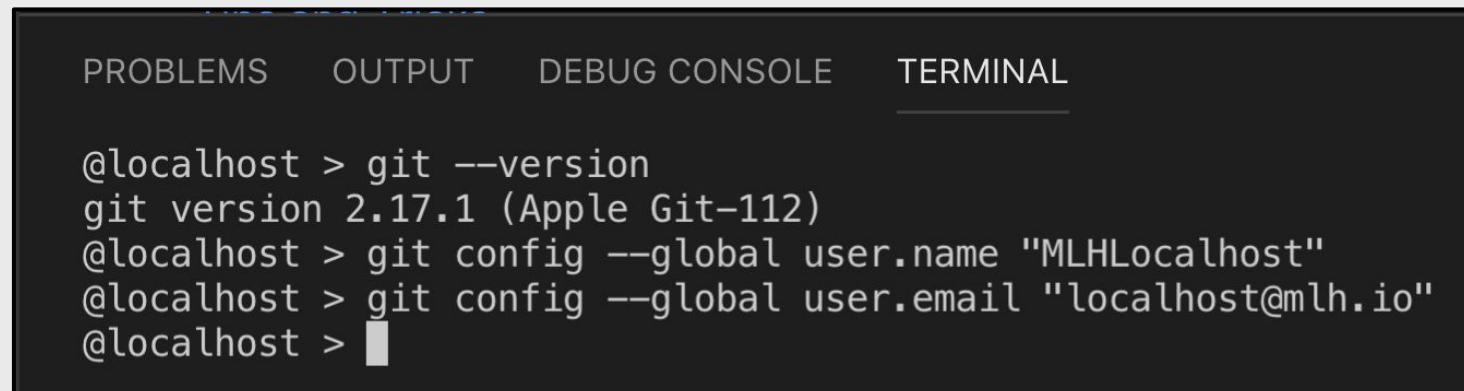
Configure Git Locally

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 :

A screenshot of the Visual Studio Code terminal window. The tab bar at the top shows 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL' (which is underlined). The terminal itself contains the following text:

```
@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 > █
```

The text is white on a dark background.

**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

- 1. Introduction to Web Development**
- 2. Microsoft Azure and Visual Studio**
- 3. Node.js and Express**
- 4. Deploy Your Web App!**
- 5. Review & Quiz**
- 6. Next Steps**

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

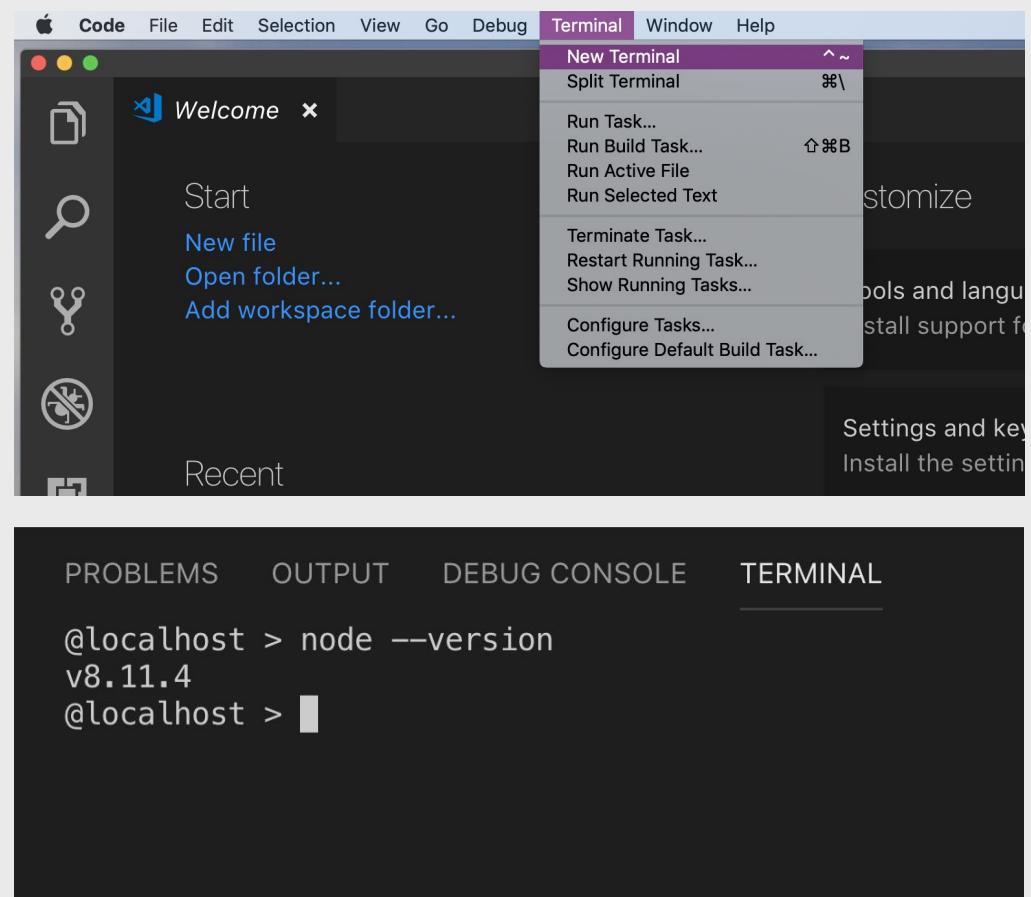
<http://mlhlocal.host/node-linux>

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

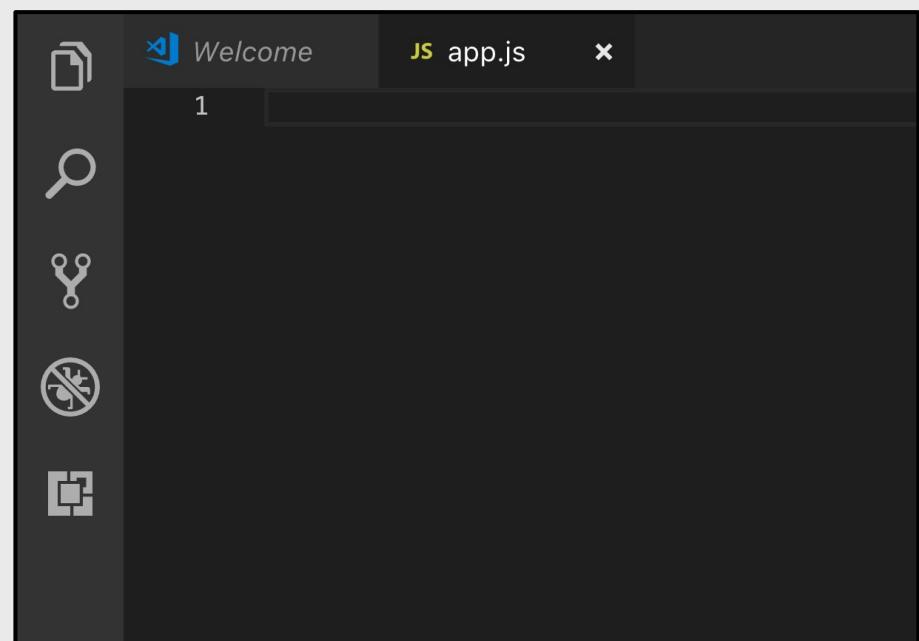
Express is a very simple web framework for Node.js.

- 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:

```
JS 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!

```
1
2
3
4
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}!`))
```

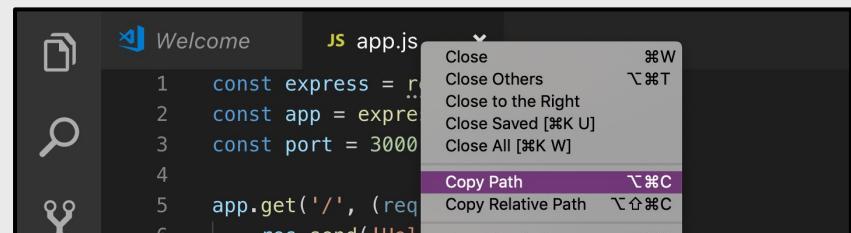
What does this code mean?

- **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.

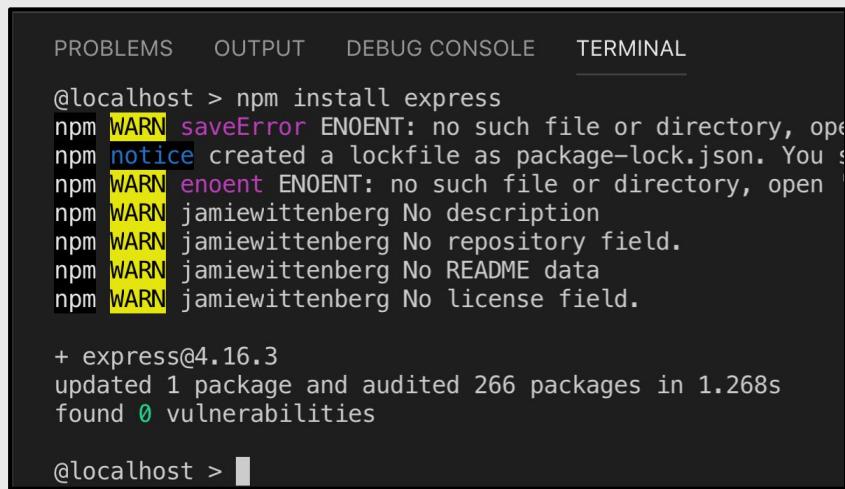
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
@localhost > cd /Users/localhost/Desktop/app.js
```

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

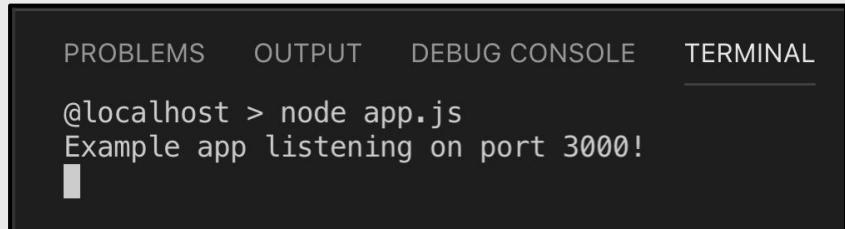
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
@localhost > cd /Users/localhost/Desktop
```

Hello, World!

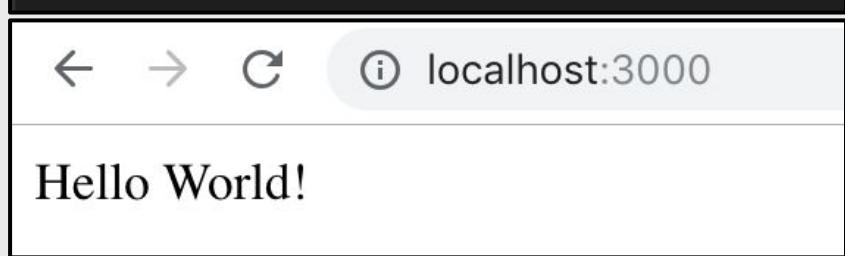
10. Type `npm install express` so that we can use Express on our computer.
11. Type `node app.js` to run our file.
12. Visit **localhost:3000** in your browser.
13. Go back to your terminal. What's changed?
14. Enter [CTRL C] to kill the server.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL  
@localhost > npm install express  
npm WARN saveError ENOENT: no such file or directory, open  
npm notice created a lockfile as package-lock.json. You  
npm WARN enoent ENOENT: no such file or directory, open  
npm WARN jamiewittenberg No description  
npm WARN jamiewittenberg No repository field.  
npm WARN jamiewittenberg No README data  
npm WARN jamiewittenberg No license field.  
+ express@4.16.3  
updated 1 package and audited 266 packages in 1.268s  
found 0 vulnerabilities  
@localhost >
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL  
@localhost > node app.js  
Example app listening on port 3000!  
|
```



← → ⌂ ⓘ localhost:3000
Hello World!

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!

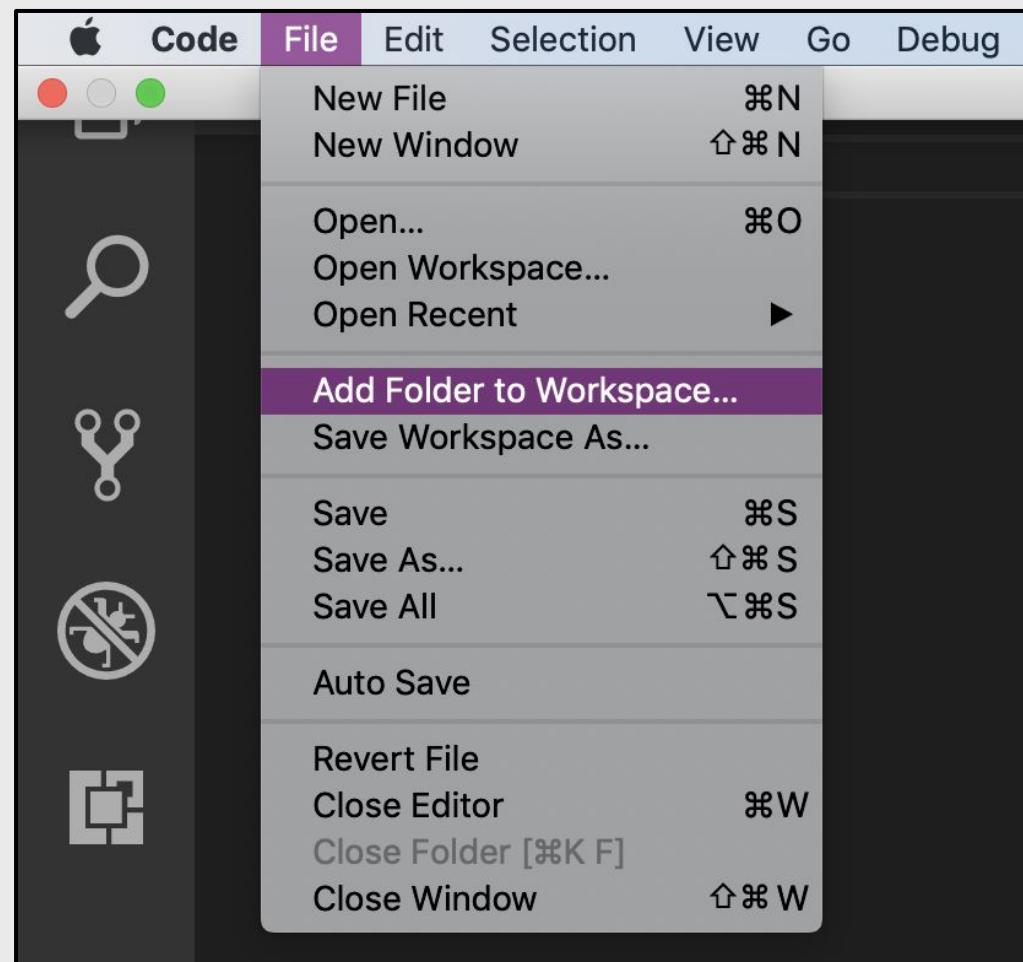
Download the Hacker Log Starter Code

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

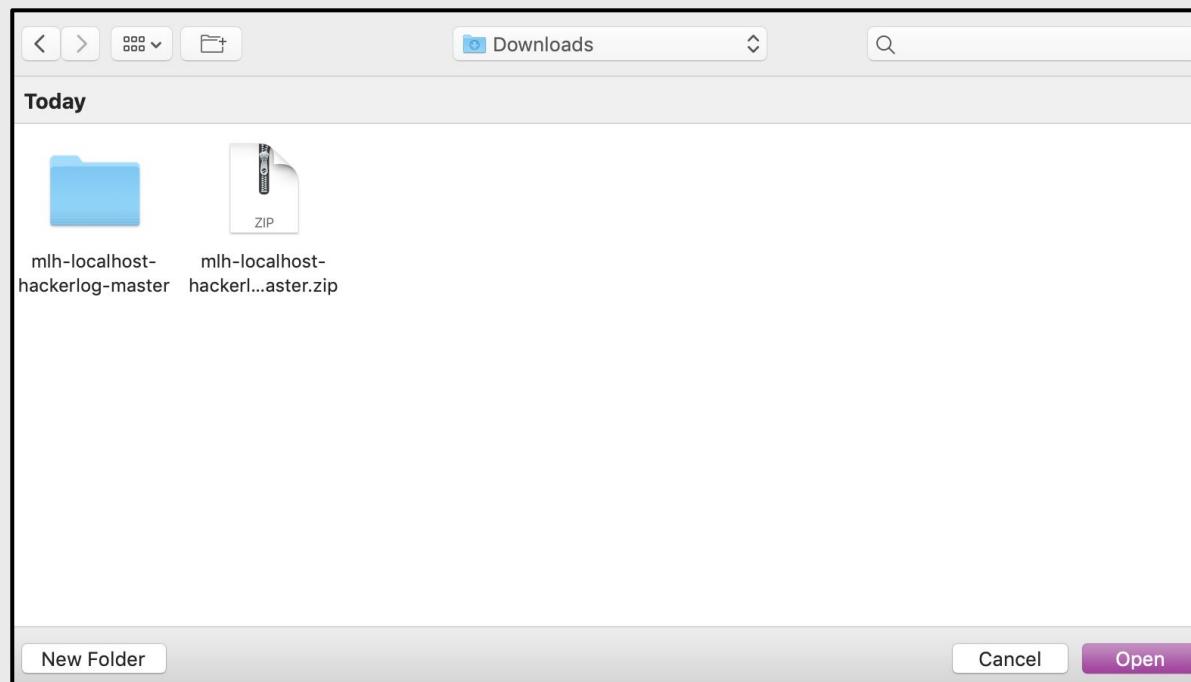
1. Navigate to the URL above.
2. A zip file of the code will be downloaded.
3. We need to unzip the file and open it in Visual Studio Code. These steps are different for MacOS and Windows – please follow the next steps **very carefully** depending on your operating system!

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.**

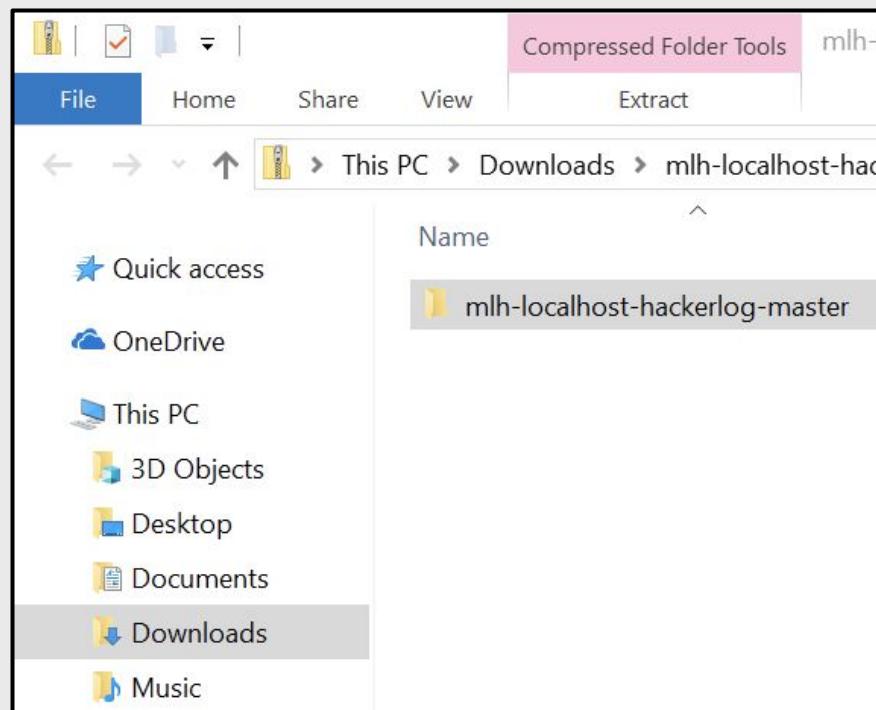


Add Project to Visual Studio Code: MacOS



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



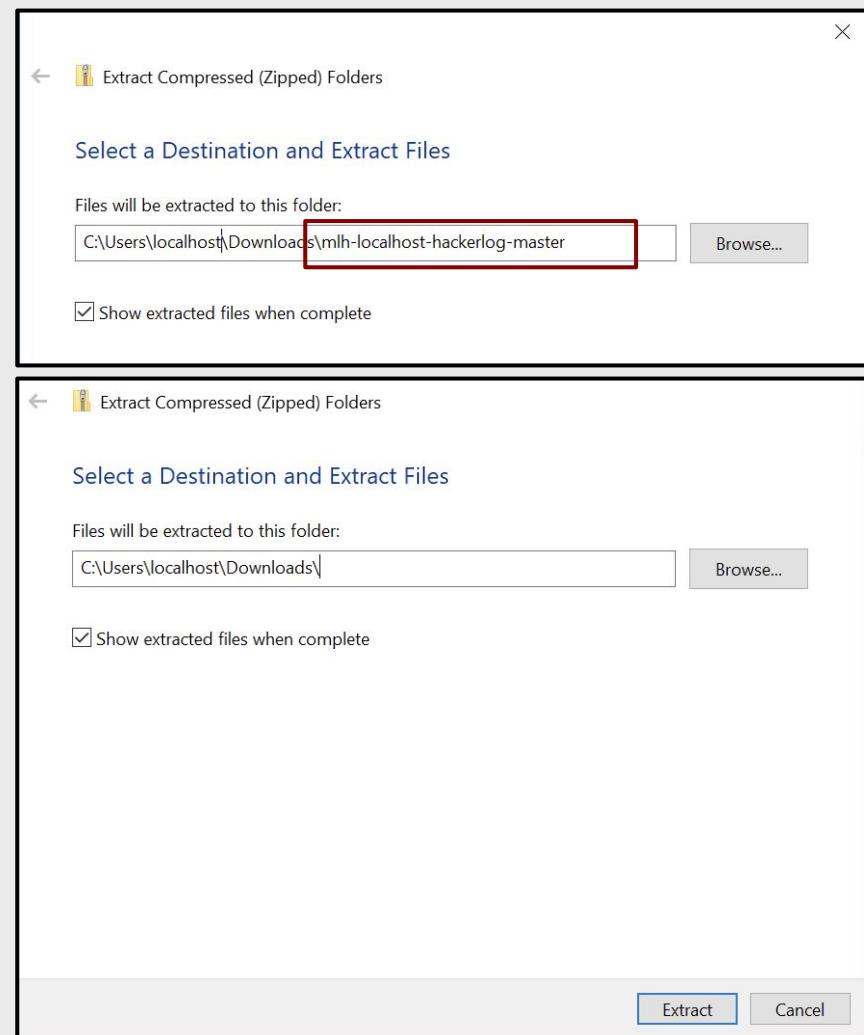
1. On a Windows computer, open the Downloads folder.
2. Click **mlh-localhost-hackerlog-master**.
3. Click **Extract** at the top.
4. Click **Extract All**.

Add Project to Visual Studio Code: Windows

5. This screen should pop up. Delete **mlh-localhost-hackerlog-master**. Click **Extract**.

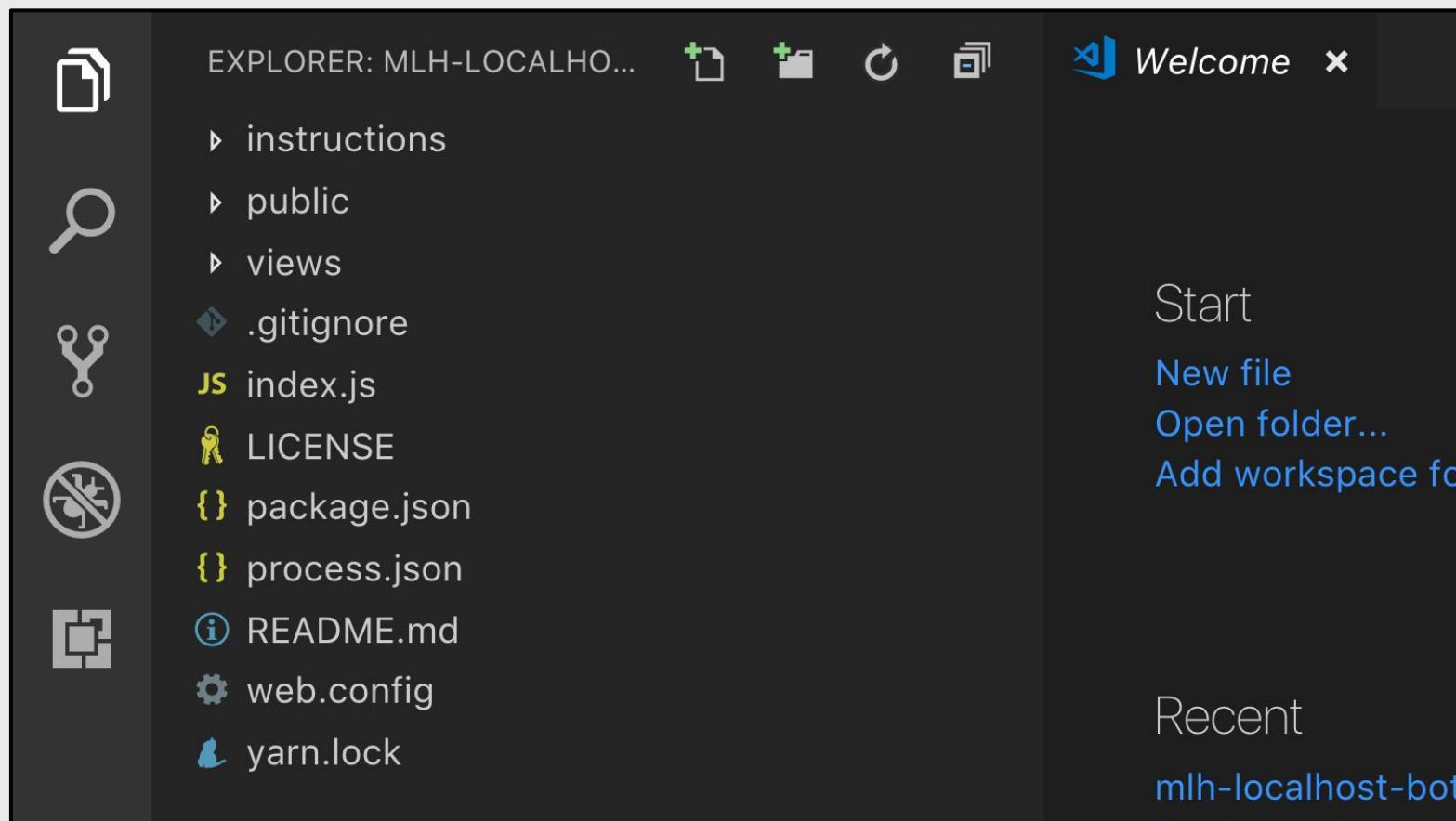
6. In Visual Studio Code, click **File** -> **Add Folder to Workspace**.

7. Select this folder from your Downloads.



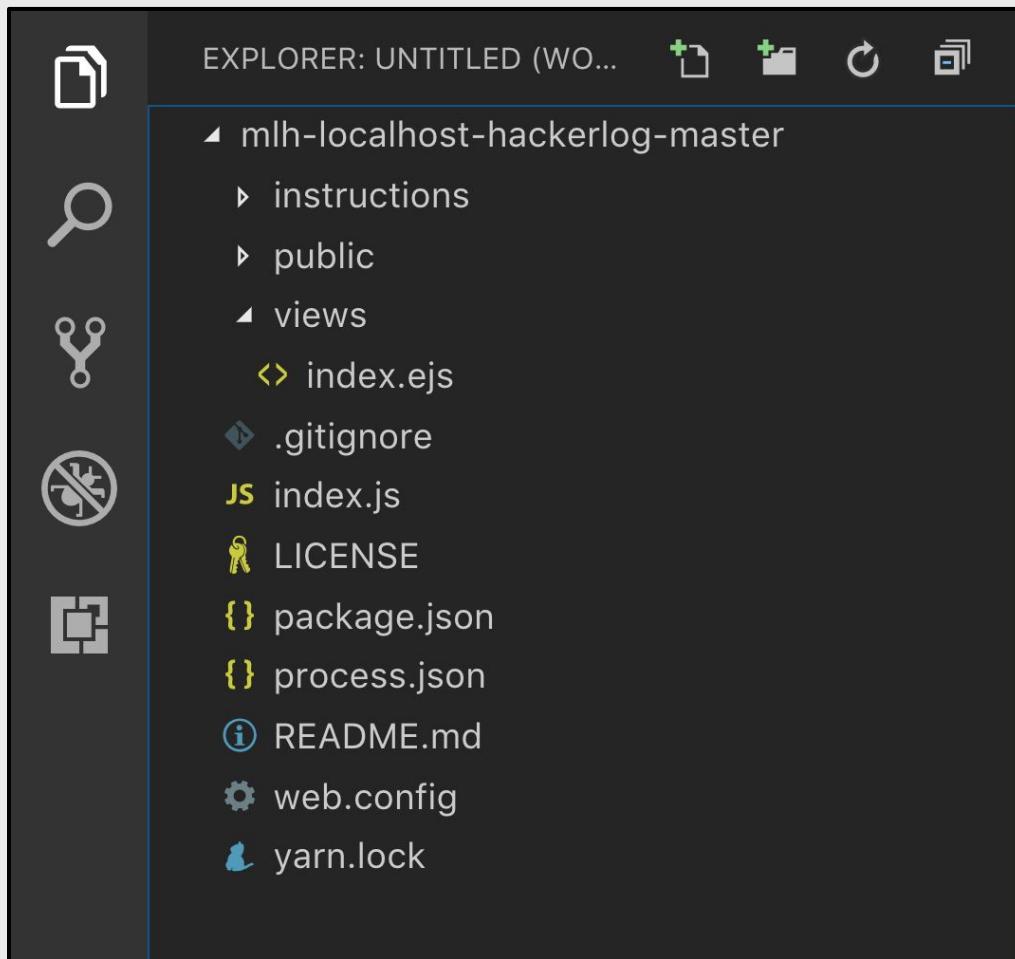
Add Project to Visual Studio Code

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



What's in this project?

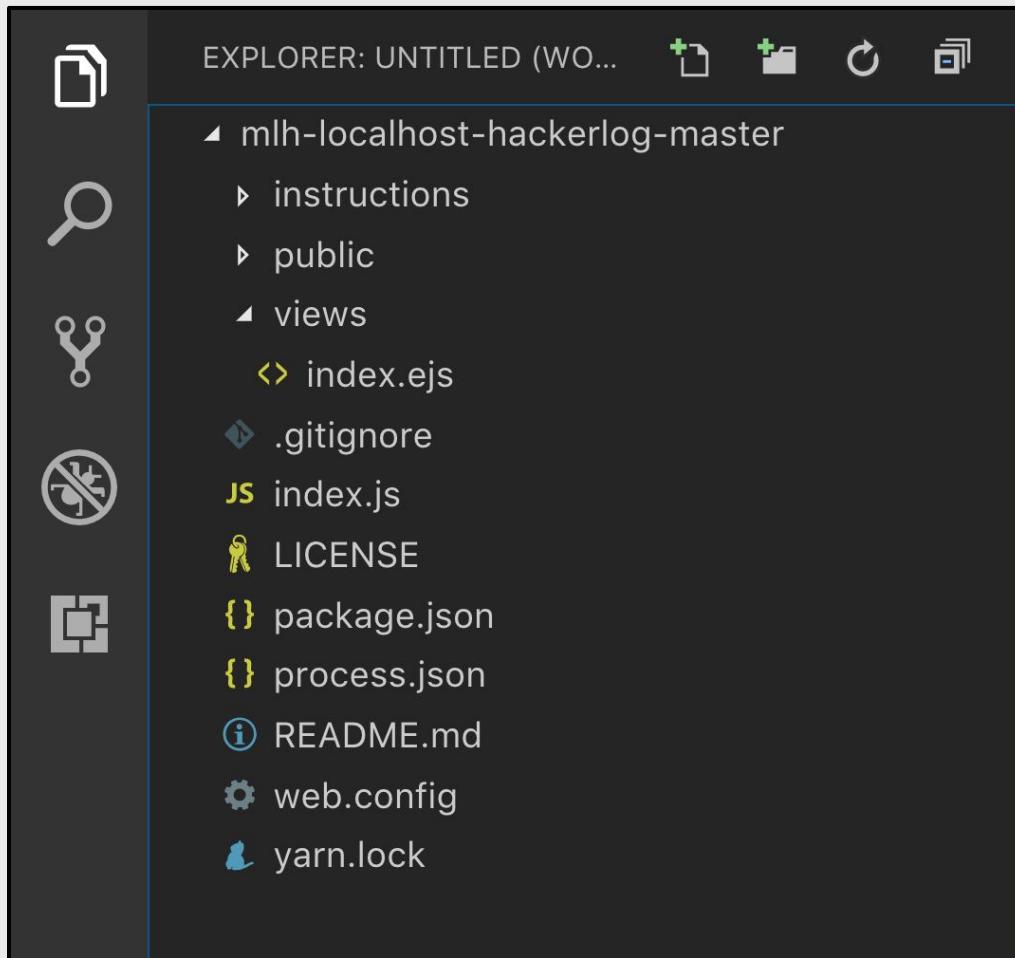
What are these files?



- **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!

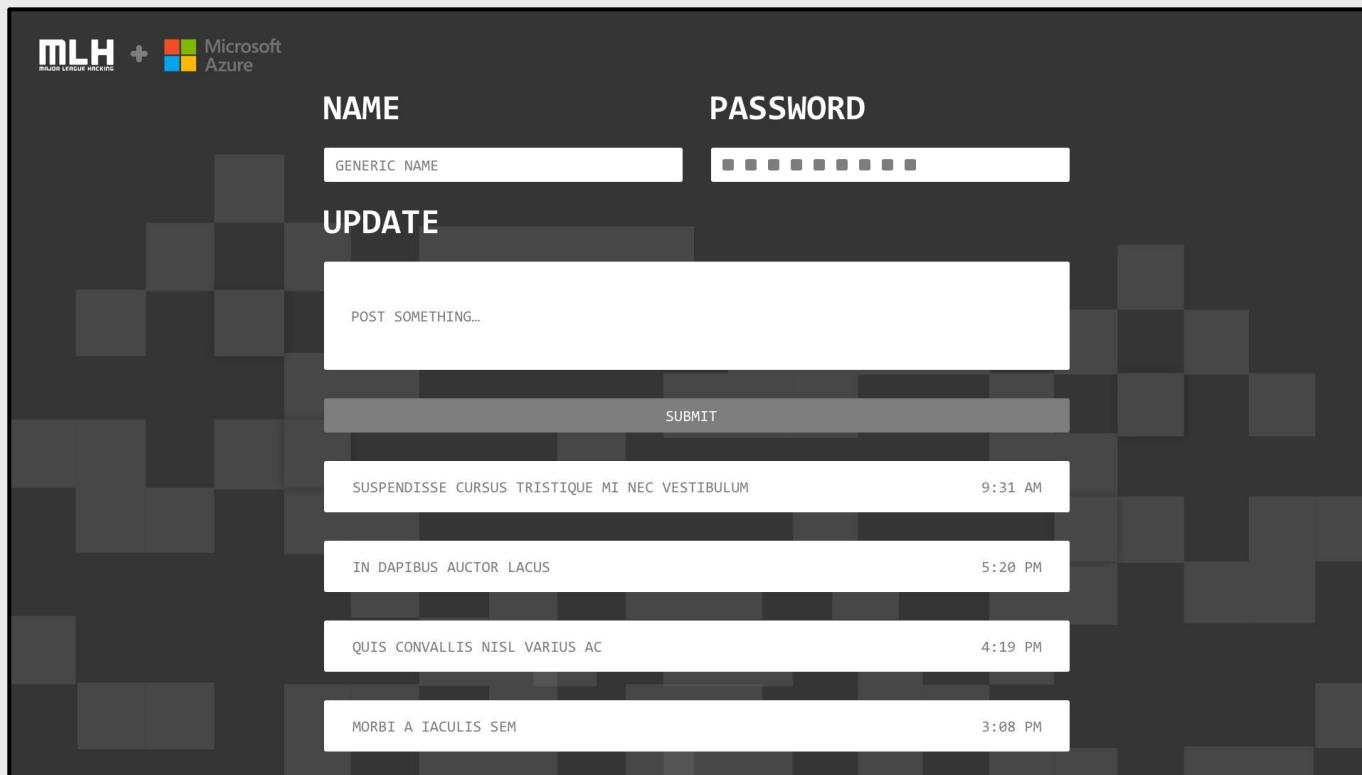
Now that our code is complete, let's try running our app again!

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!

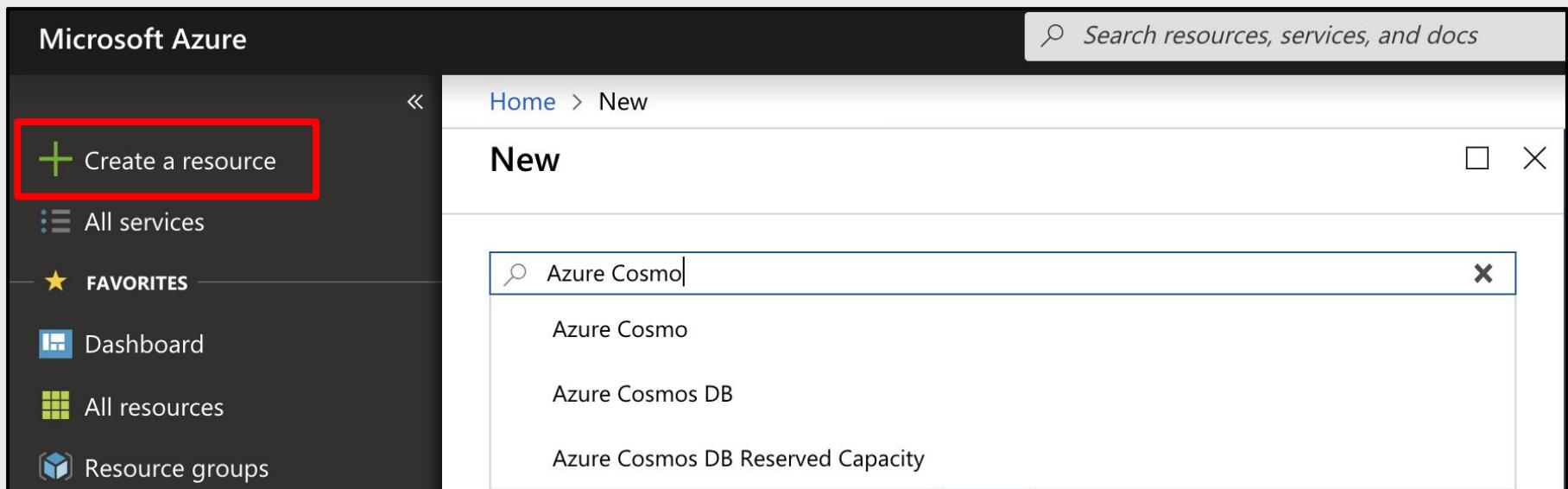
Table of Contents

- 1. Introduction to Web Development**
- 2. Microsoft Azure and Visual Studio**
- 3. Node.js and Express**
- 4. Deploy Your Web App!**
- 5. Review & Quiz**
- 6. Next Steps**

Create a Cosmos Resource

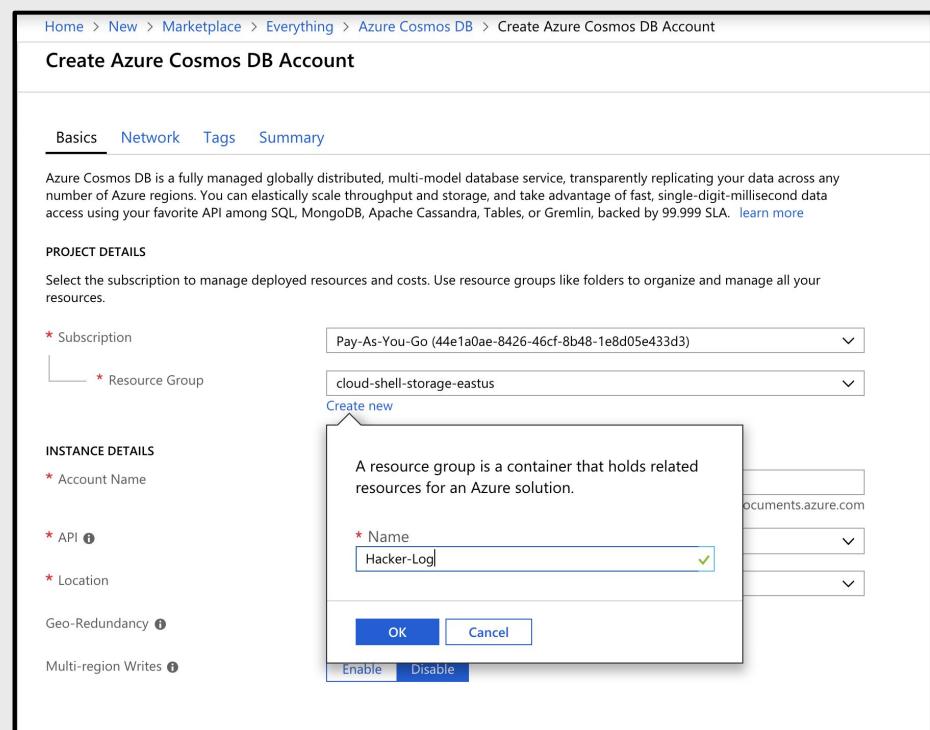
First, we'll create our database on Azure.

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 Azure Cosmos DB Account

Basics Network Tags Summary

Azure Cosmos DB is a fully managed globally distributed, multi-model database service, transparently replicating your data across any number of Azure regions. You can elastically scale throughput and storage, and take advantage of fast, single-digit-millisecond data access using your favorite API among SQL, MongoDB, Apache Cassandra, Tables, or Gremlin, backed by 99.999 SLA. [learn more](#)

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription: Pay-As-You-Go (44e1a0ae-8426-46cf-8b48-1e8d05e433d3)

* Resource Group: (New) Hacker-Log

Create new

INSTANCE DETAILS

* Account Name: mlhlocalhost

documents.azure.com

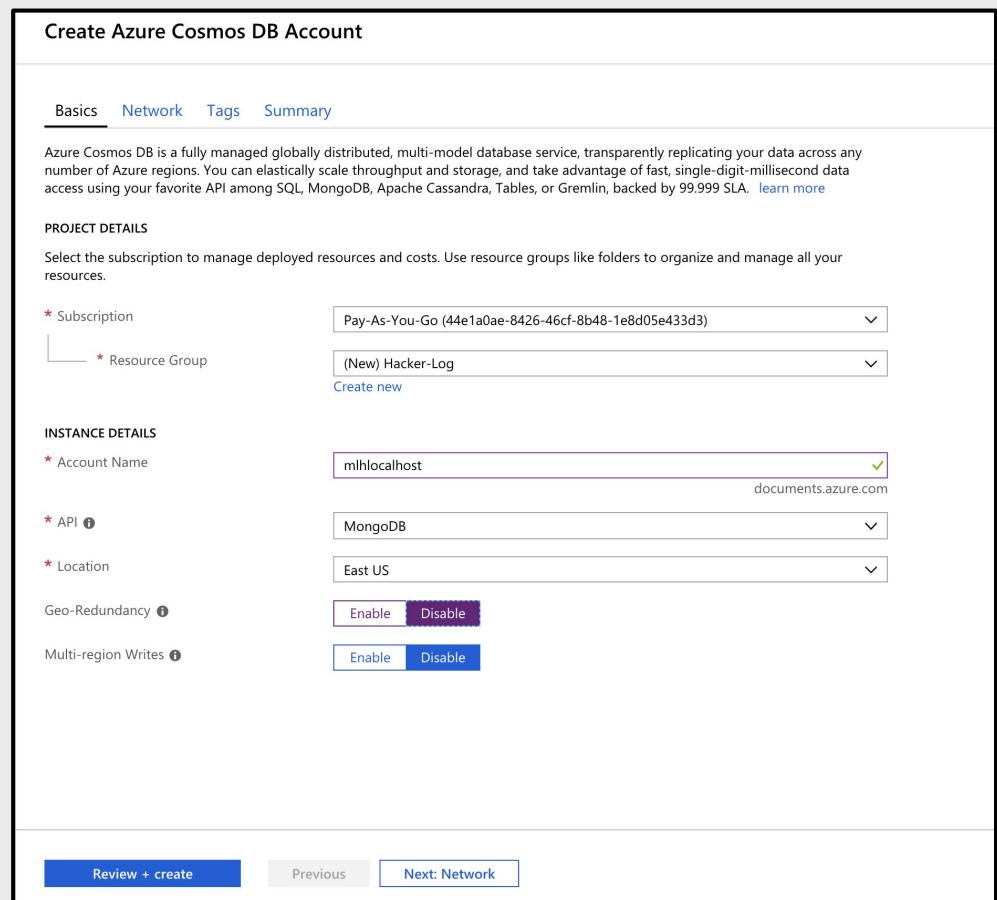
* API: MongoDB

* Location: East US

Geo-Redundancy: Enable Disable

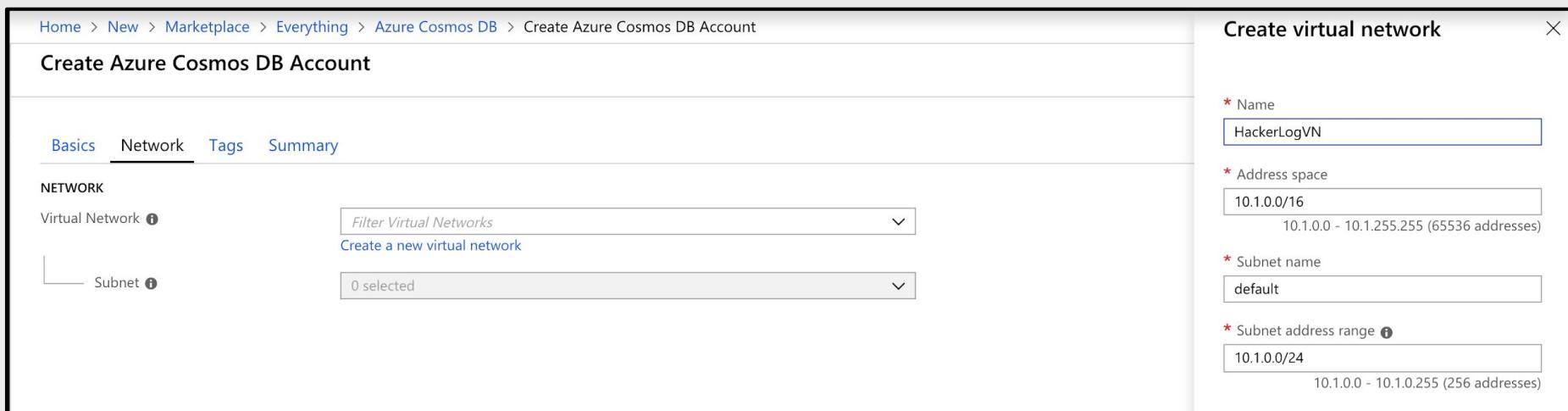
Multi-region Writes: Enable Disable

Review + create Previous Next: Network



Create a Cosmos Resource

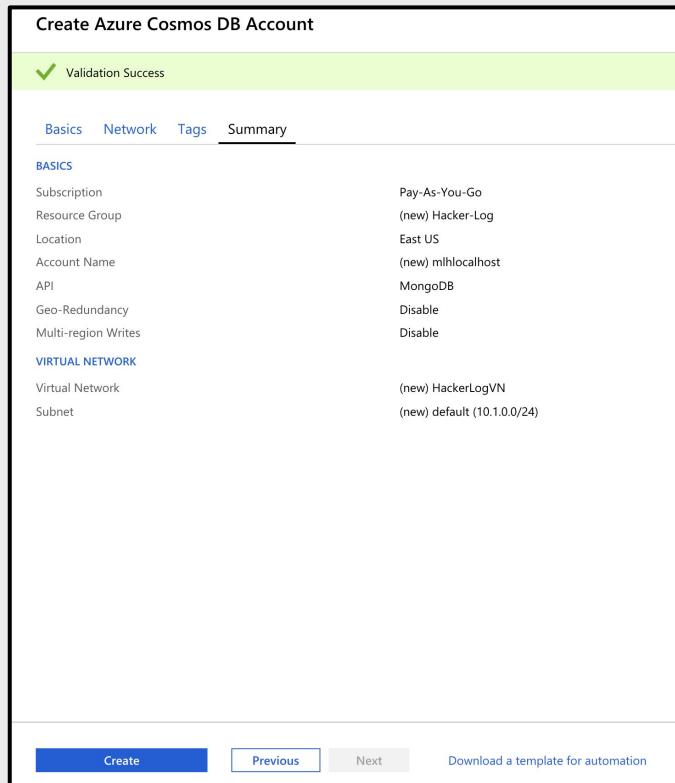
10. Click **Create a new virtual network**.
11. Name it HackerLogVN.
12. **Click OK**.
13. Click **Review + create**



The screenshot shows the Azure portal interface for creating an Azure Cosmos DB account. The main page title is "Create Azure Cosmos DB Account". Below it, there are tabs for "Basics", "Network", "Tags", and "Summary", with "Network" currently selected. Under the "NETWORK" section, there is a dropdown menu for "Virtual Network" with the placeholder "Filter Virtual Networks" and a link to "Create a new virtual network". Below this, there is a dropdown menu for "Subnet" with the placeholder "0 selected". To the right, a modal window titled "Create virtual network" is open, containing fields for "Name" (set to "HackerLogVN"), "Address space" (set to "10.1.0.0/16" with a note "10.1.0.0 - 10.1.255.255 (65536 addresses)"), "Subnet name" (set to "default"), and "Subnet address range" (set to "10.1.0.0/24" with a note "10.1.0.0 - 10.1.0.255 (256 addresses)").

Create a Cosmos Resource

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

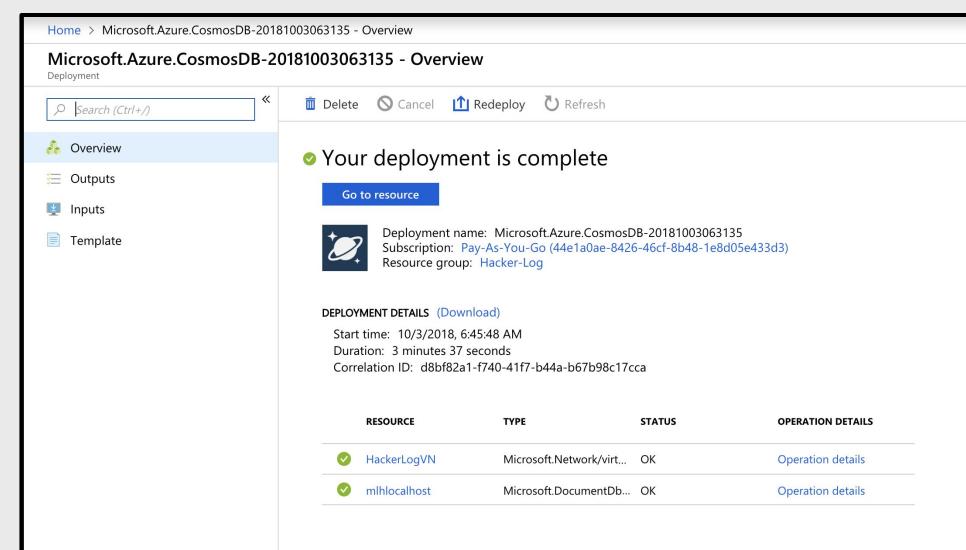


The screenshot shows the 'Create Azure Cosmos DB Account' wizard. The 'Validation Success' step has been completed. The 'Summary' tab is selected. The 'Basics' section contains the following configuration:

Setting	Value
Subscription	Pay-As-You-Go
Resource Group	(new) Hacker-Log
Location	East US
Account Name	(new) mihlocalhost
API	MongoDB
Geo-Redundancy	Disable
Multi-region Writes	Disable
VIRTUAL NETWORK	(new) HackerLogVN
Virtual Network	(new) default (10.1.0.0/24)

At the bottom, there are 'Create', 'Previous', and 'Next' buttons, along with a link to download a template for automation.

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



The screenshot shows the 'Microsoft.Azure.CosmosDB-20181003063135 - Overview' page. The deployment status is shown as 'Your deployment is complete'. Deployment details include:

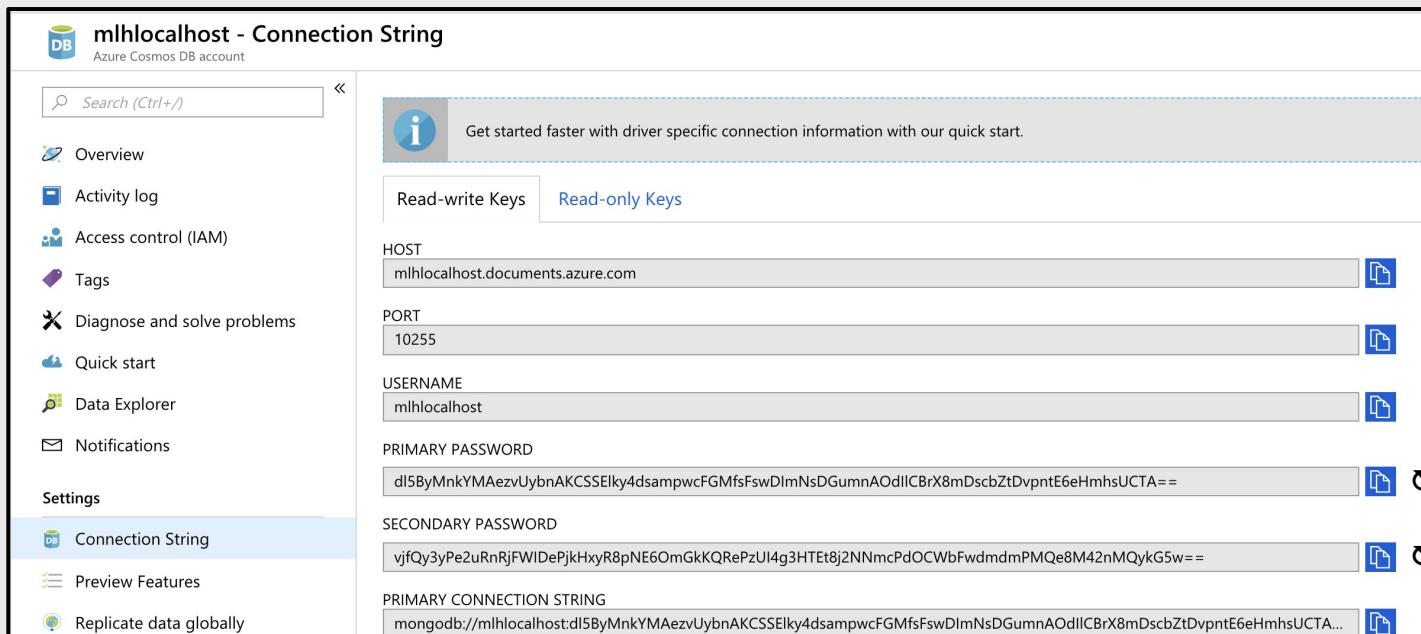
- Deployment name: Microsoft.Azure.CosmosDB-20181003063135
- Subscription: Pay-As-You-Go (44e1a0ae-8426-46cf-8b48-1e8d05e433d3)
- Resource group: Hacker-Log

DEPLOYMENT DETAILS (Download):
Start time: 10/3/2018, 6:45:48 AM
Duration: 3 minutes 37 seconds
Correlation ID: d8bf82a1-f740-41f7-b44a-b67b98c17cca

RESOURCE	TYPE	STATUS	OPERATION DETAILS
HackerLogVN	Microsoft.Network/virt...	OK	Operation details
mihlocalhost	Microsoft.DocumentDb...	OK	Operation details

Create a Cosmos Resource

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

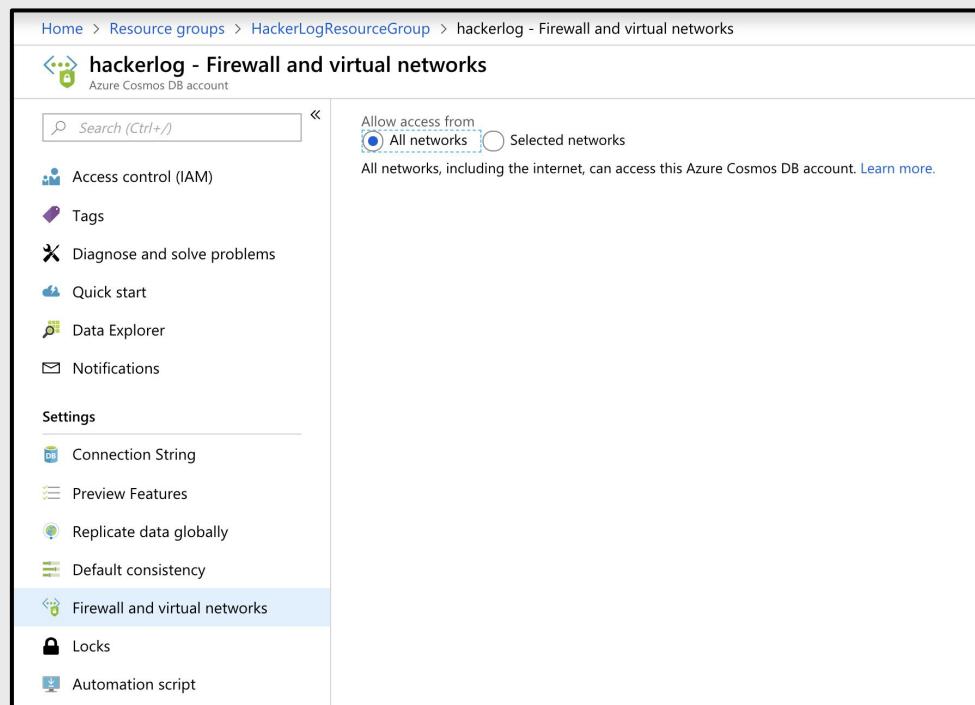


The screenshot shows the Azure portal interface for managing an Azure Cosmos DB account named "mlhlocalhost". The left sidebar has a tree view with "Connection String" selected under the "Settings" category. The main content area is titled "mlhlocalhost - Connection String" and "Azure Cosmos DB account". It displays connection information for a "Read-only Keys" endpoint:

Setting	Value	Action
HOST	mlhlocalhost.documents.azure.com	
PORT	10255	
USERNAME	mlhlocalhost	
PRIMARY PASSWORD	dl5ByMnkYMAezvUybnAKCSSElk4dsampwcFGMfsFswDlmNsDGumnAOdIICBrX8mDscbZtDvpntE6eHmhhsUCTA==	
SECONDARY PASSWORD	vjfQy3yPe2uRnRjFWIDePjkHxyR8pNE6OmGkKQRcpzUl4g3HTEt8j2NNmcPdOCWbfwdmdmPMQe8M42nMQykG5w==	
PRIMARY CONNECTION STRING	mongodb://mlhlocalhost:dl5ByMnkYMAezvUybnAKCSSElk4dsampwcFGMfsFswDlmNsDGumnAOdIICBrX8mDscbZtDvpntE6eHmhhsUCTA...	

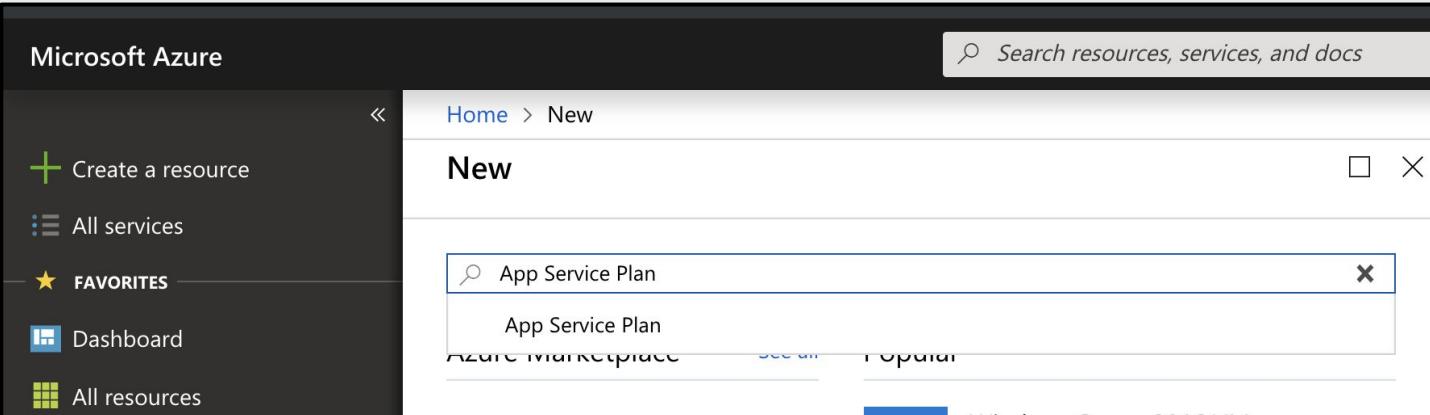
Create a Cosmos Resource

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!



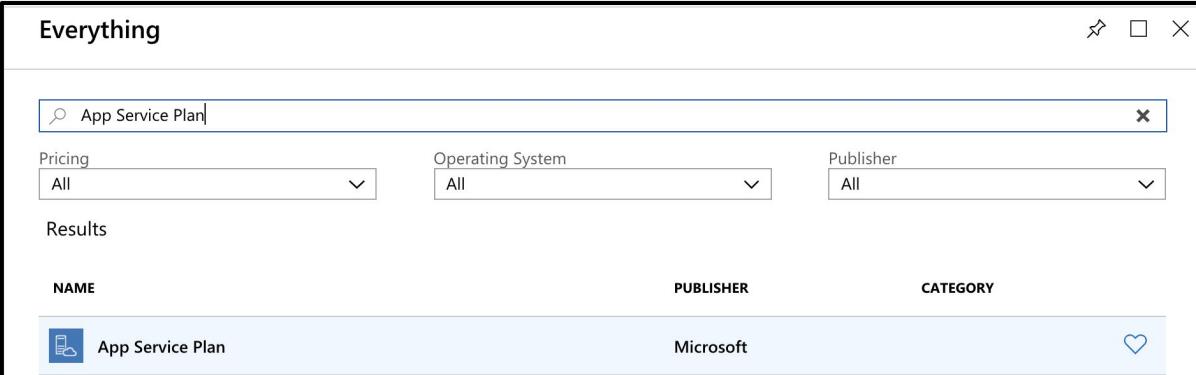
Create an App Service Plan

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



The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with options like 'Create a resource', 'All services', 'FAVORITES', 'Dashboard', and 'All resources'. The main area has a search bar at the top with the placeholder 'Search resources, services, and docs'. Below the search bar, the word 'New' is displayed. A search result for 'App Service Plan' is shown in a dropdown menu, with 'App Service Plan' being the selected item. The background shows other service offerings like 'Azure Marketplace'.

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



The screenshot shows the Azure Marketplace search results for 'App Service Plan'. The search bar at the top contains 'App Service Plan'. Below it, there are filters for 'Pricing' (set to 'All'), 'Operating System' (set to 'All'), and 'Publisher' (set to 'All'). The results section is titled 'Results' and shows one item: 'App Service Plan' by 'Microsoft'. This item has a small icon and a blue heart icon for favoriting.

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**.

New App Service Plan □ X

Create a plan for the web app

* App Service plan
HackerLogPlan ✓

* Subscription
Azure for Students ▼

* Resource Group i
 Create new Use existing
HackerLogResourceGroup ▼

* Operating System
Linux ▼

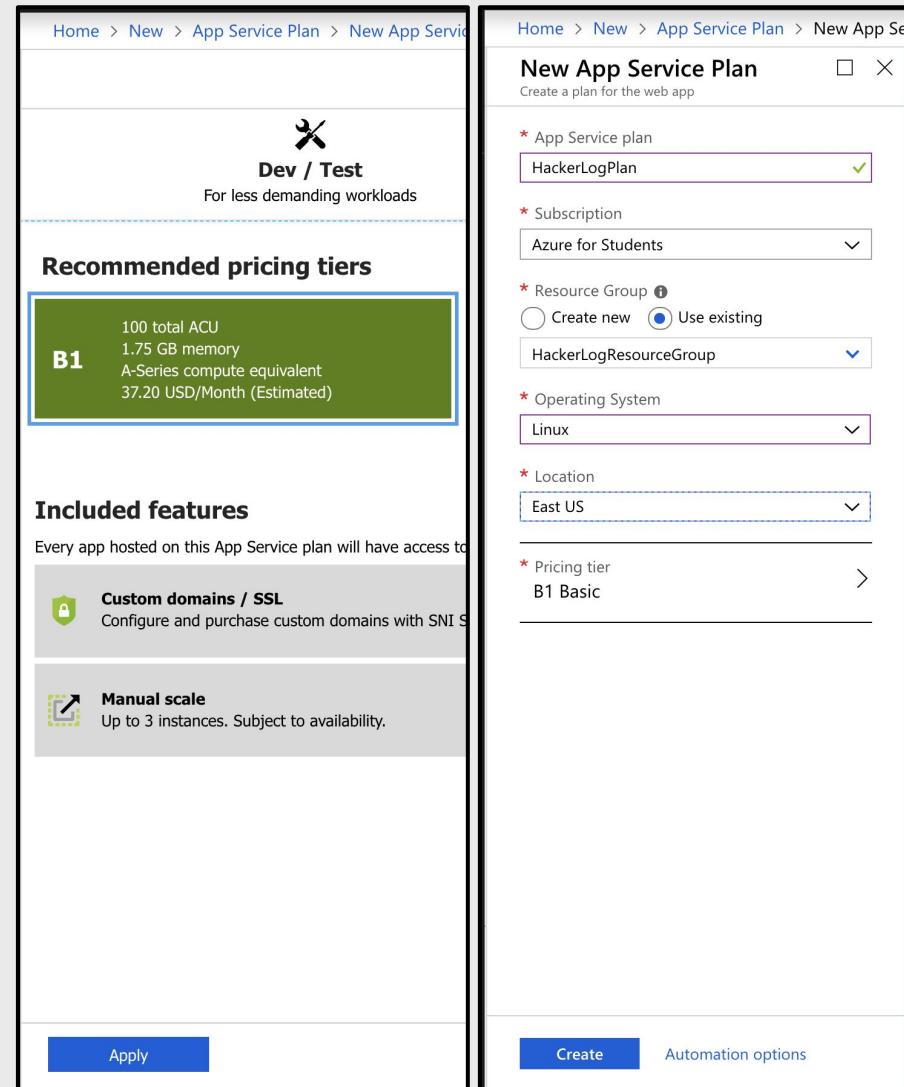
* Location
East US ▼

* Pricing tier
F1 Free >

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!

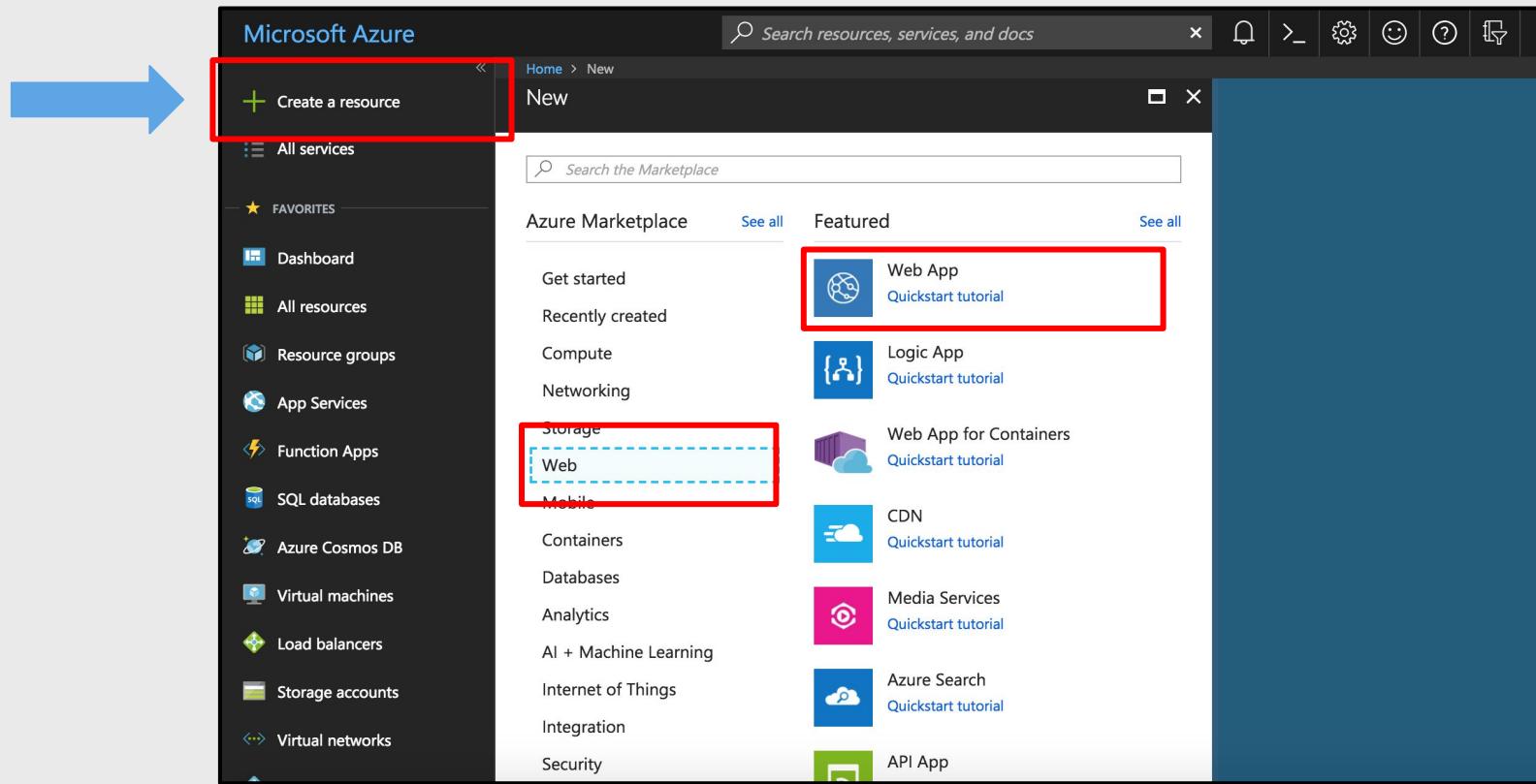


The screenshot shows two side-by-side Azure portal pages. The left page displays the 'New App Service Plan' configuration interface with fields for 'App Service plan' (set to 'HackerLogPlan'), 'Subscription' (set to 'Azure for Students'), 'Resource Group' (set to 'Create new' and 'HackerLogResourceGroup'), 'Operating System' (set to 'Linux'), and 'Location' (set to 'East US'). The 'Pricing tier' field is set to 'B1 Basic'. The right page shows the 'New App Service Plan' summary, indicating a 'Dev / Test' plan with 100 total ACU, 1.75 GB memory, A-Series compute equivalent, and a monthly cost of 37.20 USD. It also lists 'Included features' such as 'Custom domains / SSL' and 'Manual scale'. At the bottom of both pages are 'Apply' and 'Create' buttons.

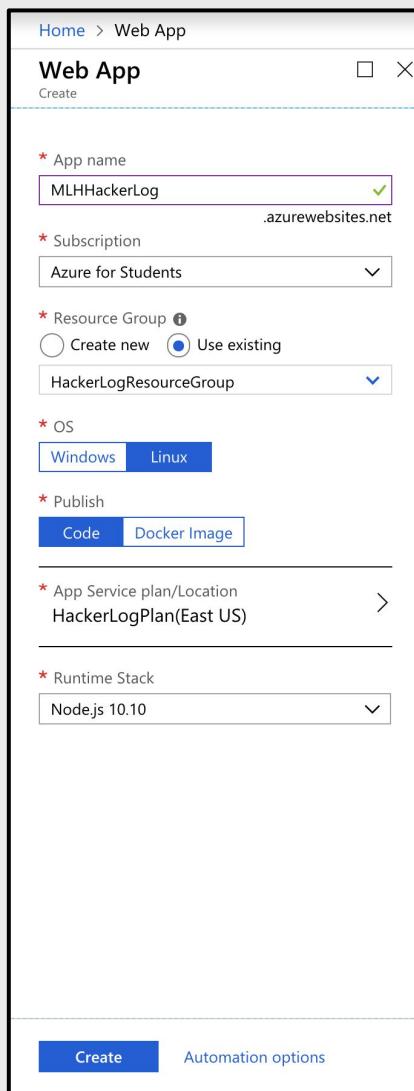
Deploy Your App to Microsoft Azure

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

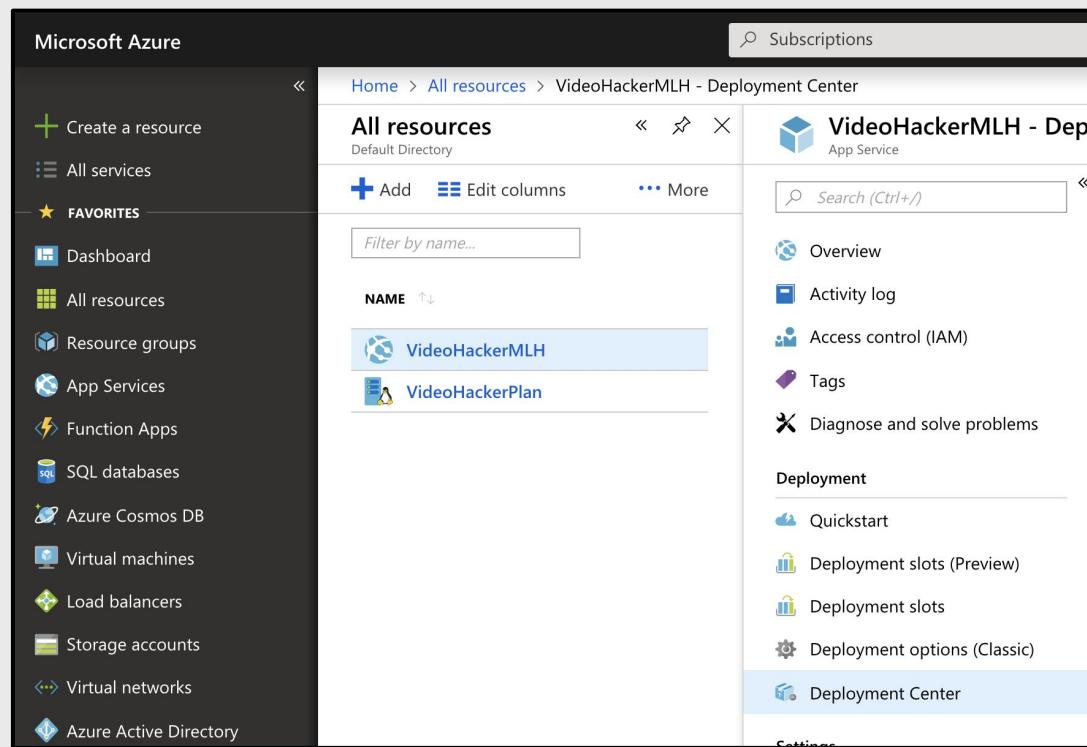


2. In the panel that opens, name your app something unique.
3. Select the Resource Group you made previously.
4. Select **Linux** as your operating system.
5. Select **Node.js 10.10** as your Runtime Stack.
6. Click **Create** at the bottom of the screen.

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

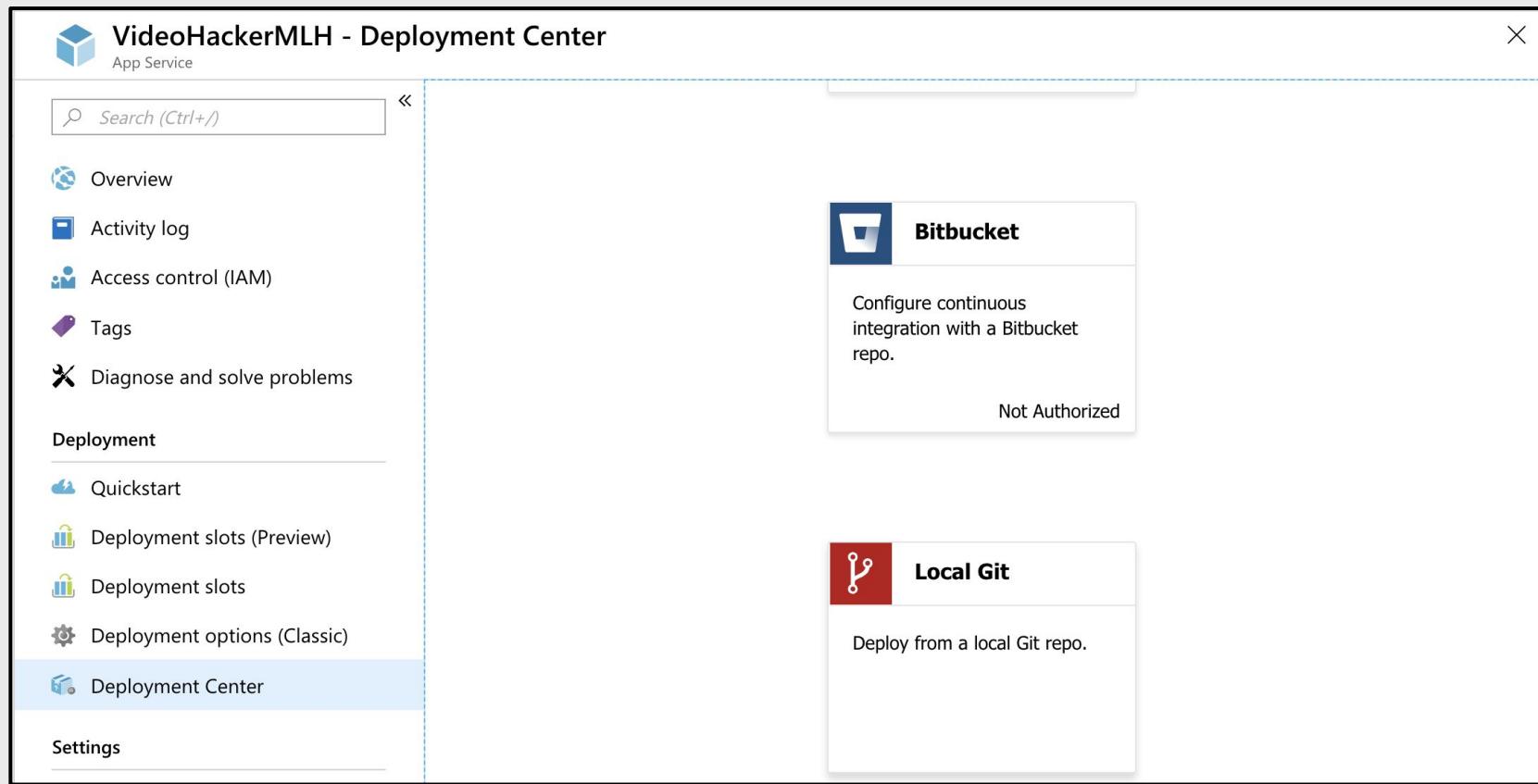
Deploy Your App to Microsoft Azure

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**.



The screenshot shows the Microsoft Azure Deployment Center interface for the "VideoHackerMLH - Deployment Center" app service. The left sidebar lists various management options like Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. Under the Deployment section, "Deployment Center" is selected and highlighted with a blue background. To the right, there are two main configuration boxes: "Bitbucket" and "Local Git". The Bitbucket box contains the text "Configure continuous integration with a Bitbucket repo." and a "Not Authorized" message. The Local Git box contains the text "Deploy from a local Git repo." Both boxes have their respective logos at the top.

Deploy Your App to Microsoft Azure

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

Deployment Center

App Service Deployment Center enables you to choose the location of your code as well as options for build and deployment to the cloud. [Learn more](#)

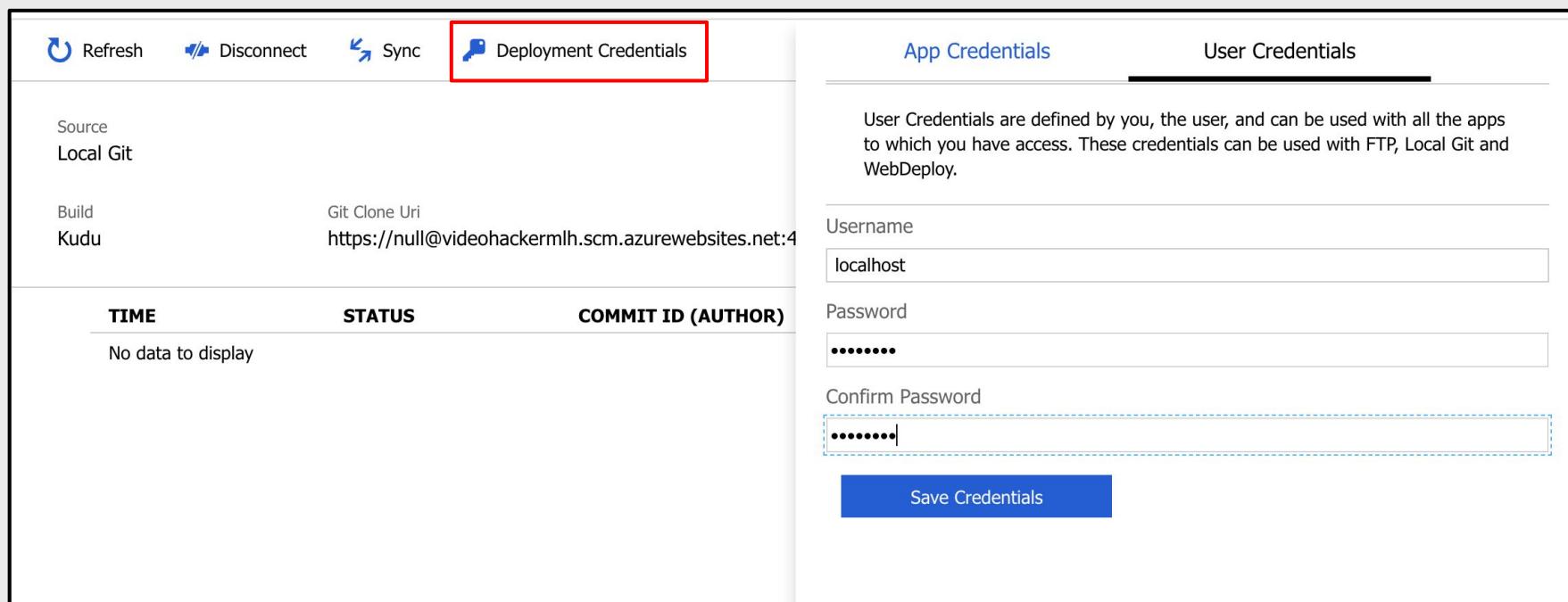
1 SOURCE CONTROL 2 BUILD PROVIDER 3 SUMMARY

 **App Service Kudu build server**
Use App Service as the build server. The App Service Kudu engine will automatically build your code during deployment when applicable with no additional configuration required.

 **Azure Pipelines (Preview)**
Configure a robust deployment pipeline for your application using Azure Pipelines, part of Azure DevOps Services (formerly known as VSTS). The pipeline builds, runs load tests and deploys to staging slot and then to production.

Deploy Your App to Microsoft Azure

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



The screenshot shows the Microsoft Azure Kudu interface for managing deployment credentials. The top navigation bar includes 'Refresh', 'Disconnect', 'Sync', and 'Deployment Credentials' (which is highlighted with a red box). Below this, there are sections for 'Source' (Local Git) and 'Build' (Kudu), with a 'Git Clone Uri' listed as <https://null@videohackermlh.scm.azurewebsites.net:443/>. A table at the bottom shows 'No data to display' for 'TIME', 'STATUS', and 'COMMIT ID (AUTHOR)'.

The main content area has two tabs: 'App Credentials' and 'User Credentials'. The 'User Credentials' tab is selected, displaying instructions: '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.' It includes fields for 'Username' (localhost) and 'Password' (represented by six dots). A 'Confirm Password' field also contains six dots. A large blue 'Save Credentials' button is at the bottom.

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>.

Refresh Disconnect Sync Deployment Credentials

Source
Local Git

Build
Kudu Git Clone Uri
<https://mlh-localhost@videohackermlh.scm.azurewebsites.net:443/videohackermlh.git>

TIME	STATUS	COMMIT ID (AUTHOR)	CHECKIN MESSAGE	LOGS
No data to display				

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>:<primary_password>@<hostname>:<port>/?ssl=true&replicaSet=globaldb
```

Application settings

Info Application Settings are encrypted at rest and transmitted over an encrypted channel. You can choose to display them in plain text in your browser by using the controls below.

[Hide Values](#) [Show Values](#)

APP SETTING NAME	VALUE	SLOT SETTING	DELETE
MONGODB_URI	Enter a value		

[+ Add new setting](#)

You just:

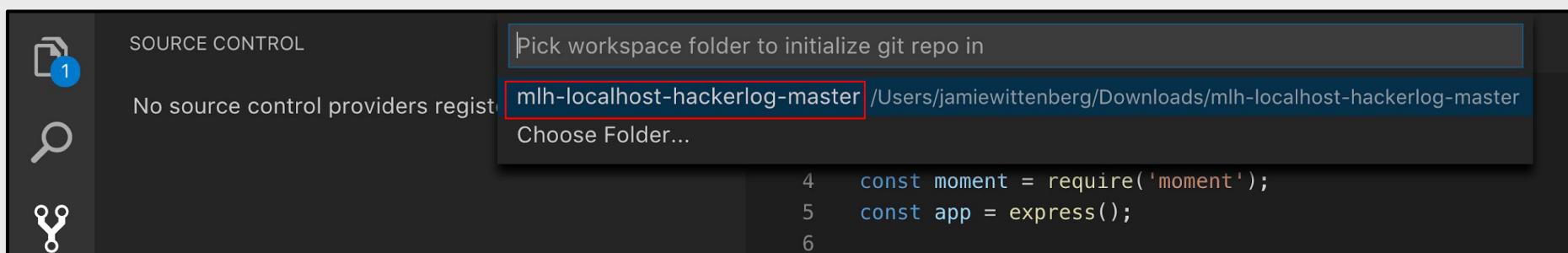
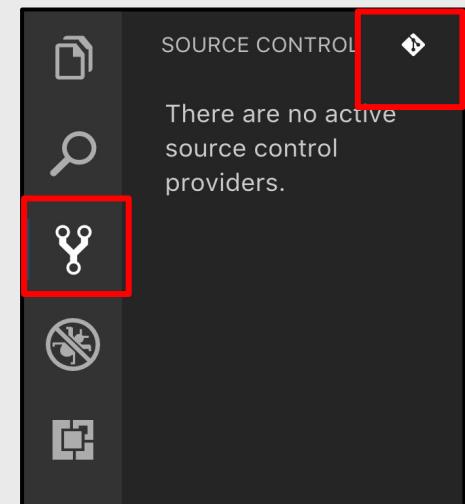
1. Created a database on Microsoft Azure.
2. Created an App Service Plan.
3. Created a Node app environment to deploy your app to.

Head back to Visual Studio Code for the rest!

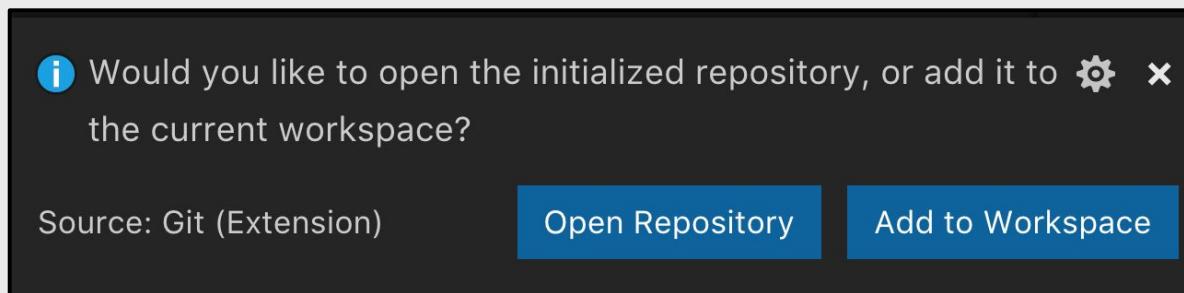
Deploy Your App to Microsoft Azure

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

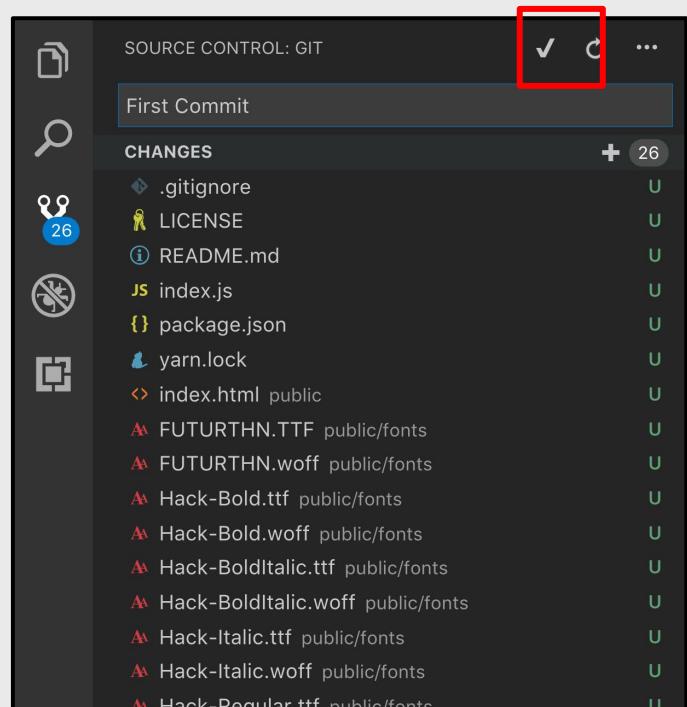
mlh-localhost-hackerlog-master and click it.



Deploy Your App to Microsoft Azure

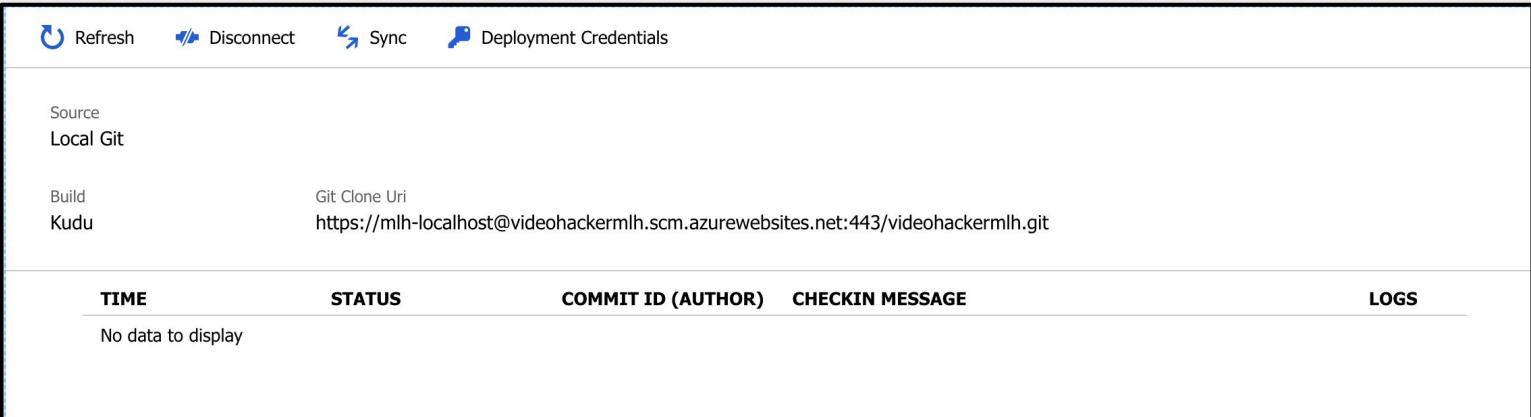


4. Select **Add to Workspace** if prompted.
5. Type "First Commit" above CHANGES.
6. Click the checkmark.
7. Click **Yes** if you get a modal asking about staging changes.



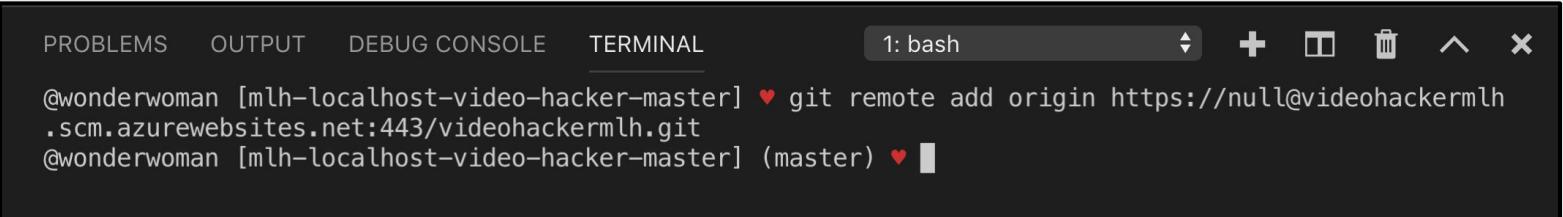
Deploy Your App to Microsoft Azure

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



The screenshot shows the Microsoft Azure Deployment Center interface. At the top, there are buttons for Refresh, Disconnect, Sync, and Deployment Credentials. Below that, under the Source section, it says "Local Git". Under the Build section, it says "Kudu" and shows the "Git Clone Uri" as <https://mlh-localhost@videohackermlh.scm.azurewebsites.net:443/videohackermlh.git>. At the bottom, there is a table with columns: TIME, STATUS, COMMIT ID (AUTHOR), CHECKIN MESSAGE, and LOGS. A message "No data to display" is shown in the first column.

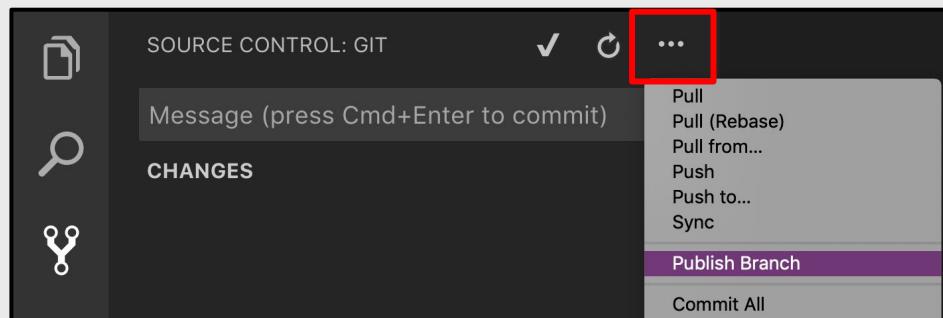
9. In your terminal, type `git remote add origin` and paste what you just copied.



The screenshot shows a terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is active, showing a bash shell. The command `@wonderwoman [mlh-localhost-video-hacker-master] ❤ git remote add origin https://mlh-localhost@videohackermlh.scm.azurewebsites.net:443/videohackermlh.git` has been entered and is being processed. The output shows the command being run and the current branch status: `@wonderwoman [mlh-localhost-video-hacker-master] (master) ❤`.

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.

>Password

Git: <https://mlh-localhost@videohackermlh.scm.azurewebsites.net:443> (Press 'Enter' to confirm or 'Escape' to cancel)

Deploy Your App to Microsoft Azure

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

Source				
Local Git				
Build				
Kudu	Git Clone Uri	https://mlh-localhost@videohackermlh.scm.azurewebsites.net:443/videohackermlh.git		
TIME	STATUS	COMMIT ID (AUTHOR)	CHECKIN MESSAGE	LOGS
Sunday, November 18, 2018				
7:52:16 PM GMT-5	Running deployment comm	ed2dd4f (JamieMLH)	First Commit	

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!

Home > MLHacks

 **MLHacks**
App Service

Search (Ctrl+/)

[Browse](#) [Stop](#) [Swap](#) [Restart](#) [Delete](#) [Get publish profile](#) [Reset publish profile](#)

Resource group (change) MLHacks	URL https://mlhacks.azurewebsites.net 
Status Running	App Service plan/pricing tier ServicePlan3867df8-9605 (PremiumV2: 1 Small)
Location Central US	GitHub Project https://github.com/MLH/mlh-localhost-faq-bot
Subscription (change) Pay-As-You-Go	
Subscription ID 44e1a0ae-8426-46cf-8b48-1e8d05e433d3	
Tags (change) Click here to add tags	

Table of Contents

1. Introduction to Web Development

2. Microsoft Azure and Visual Studio

3. Node.js and Express

4. Deploy Your Web App!

 **5.** Review & Quiz

6. Next Steps



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

- 1.** Introduction to Web Development
- 2.** Microsoft Azure and Visual Studio
- 3.** Node.js and Express
- 4.** Deploy Your Web App!
- 5.** Review & Quiz
-  **6.** Next Steps

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!

NAME

GENERIC NAME

UPDATE

PASSWORD

Workshop

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

SUSPENDISSE CURSUS TRISTIQUE MI NEC VESTIBULUM

9:31 AM

 localhost

IN DAPIBUS AUCTOR LACOS

5:20 PM

QUIS CONVALLIS NISL VARIUS AC

4:19 PM



MORBI A IACULIS SEM

3:08 PM