

002 - A Simple Node Plan

In this video, we are going to deploy a simple Node application to Azure utilizing the App Service extension for VS Code. We're going to start with a super simple app, and then build up to something more complicated so that we can see how Azure treats different kinds of applications.

Just as Rob did in episode 1, we're going to start with the the code from the Node "Getting Started" guide. Like I said, super simple.



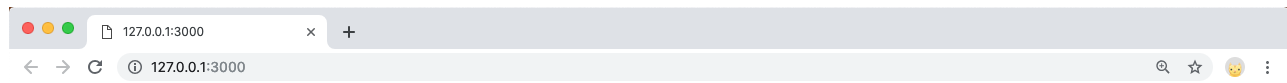
```
index.js
1  const http = require("http");
2
3  const hostname = "127.0.0.1";
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader("Content-Type", "text/plain");
9    res.end("Hello World\n");
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostname}:${port}/`);
14 });
15
```

The screenshot shows the Visual Studio Code interface. The top bar indicates the file path: ~/Dev/burkeholland/azure-casts/vs-code-app-service. The editor window displays the index.js file with the following code: 1 const http = require("http"); 2 3 const hostname = "127.0.0.1"; 4 const port = 3000; 5 6 const server = http.createServer((req, res) => { 7 res.statusCode = 200; 8 res.setHeader("Content-Type", "text/plain"); 9 res.end("Hello World\n"); 10 }); 11 12 server.listen(port, hostname, () => { 13 console.log(`Server running at http://\${hostname}:\${port}/`); 14 }); 15 The status bar at the bottom shows 0 errors, 0 warnings, and 0 info messages. It also includes links for Sign in..., Live Share, and various settings like Ln 15, Col 1, Spaces: 2, UTF-8, LF, JavaScript, ESLint, and Prettier.

To run this, We can press `cmd/ctrl + `` to open the terminal inside of VS Code. You can use your standard terminal or command line if you want, but the built-in terminal is super convenient. Then we just execute our file with node.



And let's test it in the browser just to ensure everything is on the up and up. You can click on links in the VS Code terminal by pressing `Cmd/Ctrl` when you hover over them.



Hello World

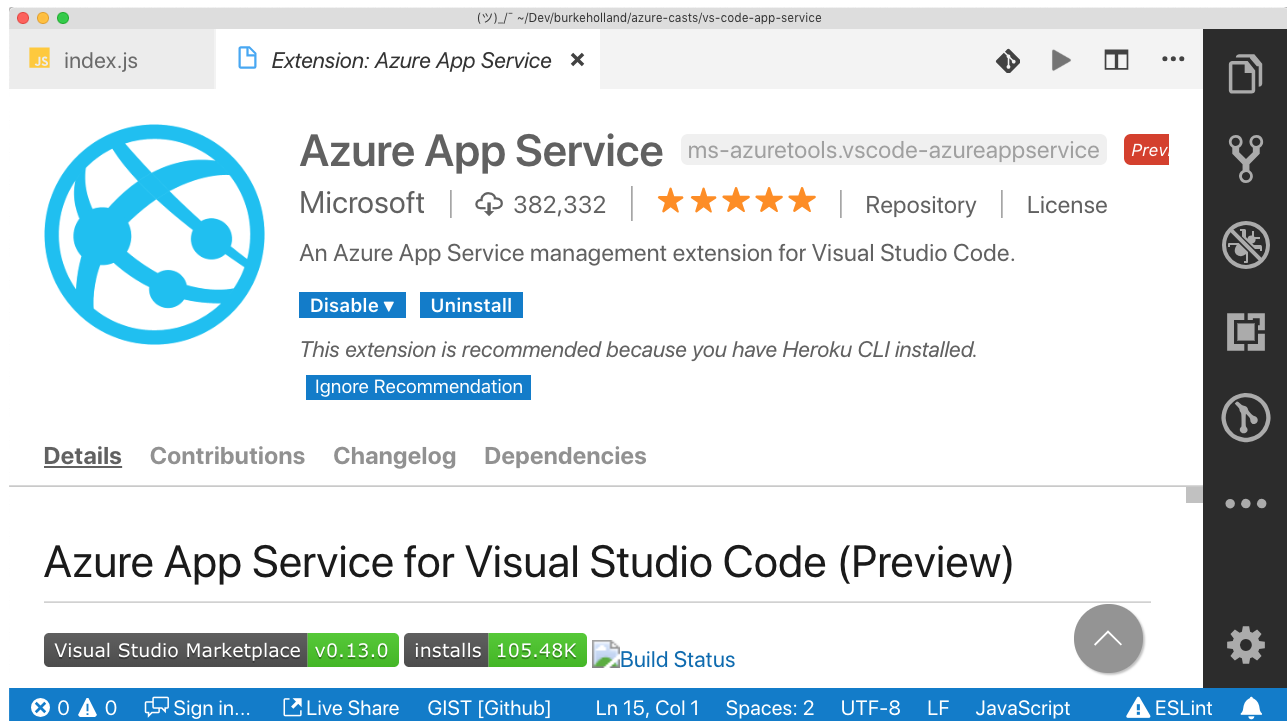
Super! Now that we have an app, let's deploy it to Azure.

There are several ways to deploy and host sites in Azure. You can do with with a virtual machine, you can do it with Docker, you can do it as just a static file, or you can do it with something called App Service.'

App Service is what is referred to as a "Platform as a Service". Often shorted to just "PaaS". Pronounced Pazzzz. I think.

The idea behind App Service is that you can just deploy your code, and Azure will provide the runtime. In our case that's Node. The important thing to note here is that you never actually have to know anything about the operating system or the server that is running your app. You don't care about that. That's Azure's problem. You just focus on deploying your app to Azure and someone else worries about the server.

Since we're in VS Code, we can use the App Service extension.



The screenshot shows the Visual Studio Code interface with the Azure App Service extension page. The extension is titled "Azure App Service" by Microsoft, with a version of v0.13.0 and 105.48K installs. It is marked as a "Preview" extension. The page includes buttons for "Disable", "Uninstall", and "Ignore Recommendation". The status bar at the bottom indicates the current file is "index.js" and the extension is active.

Azure App Service ms-azuretools.vscode-azureappservice Prev

Microsoft | 382,332 | ★★★★★ | Repository | License

An Azure App Service management extension for Visual Studio Code.

[Disable](#) [Uninstall](#)

This extension is recommended because you have Heroku CLI installed.

[Ignore Recommendation](#)

[Details](#) [Contributions](#) [Changelog](#) [Dependencies](#)

Azure App Service for Visual Studio Code (Preview)

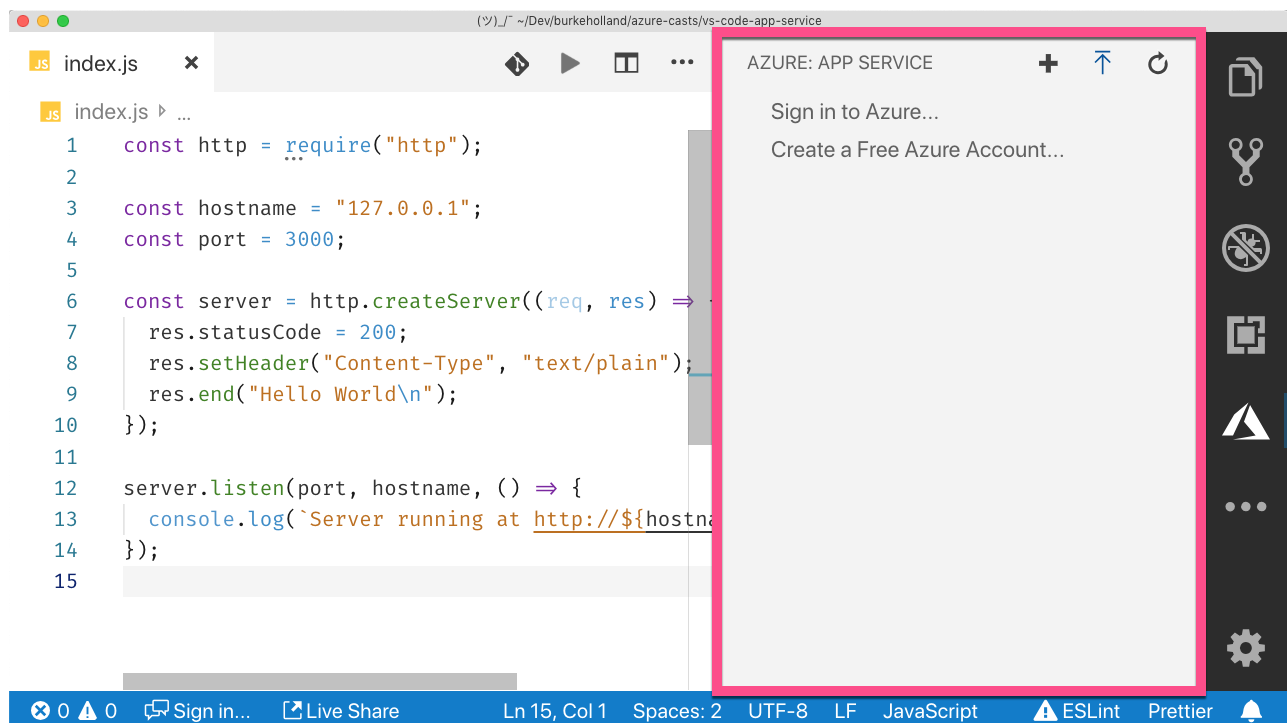
Visual Studio Marketplace v0.13.0 installs 105.48K [Build Status](#)

0 0 Sign in... Live Share GIST [Github] Ln 15, Col 1 Spaces: 2 UTF-8 LF JavaScript ESLint

This extension is installed from the VS Code extension marketplace. It's made by Microsoft, so that's how you know you've got the right one. After you install the extension, you'll see an Azure logo in the Action Bar inside of VS Code.

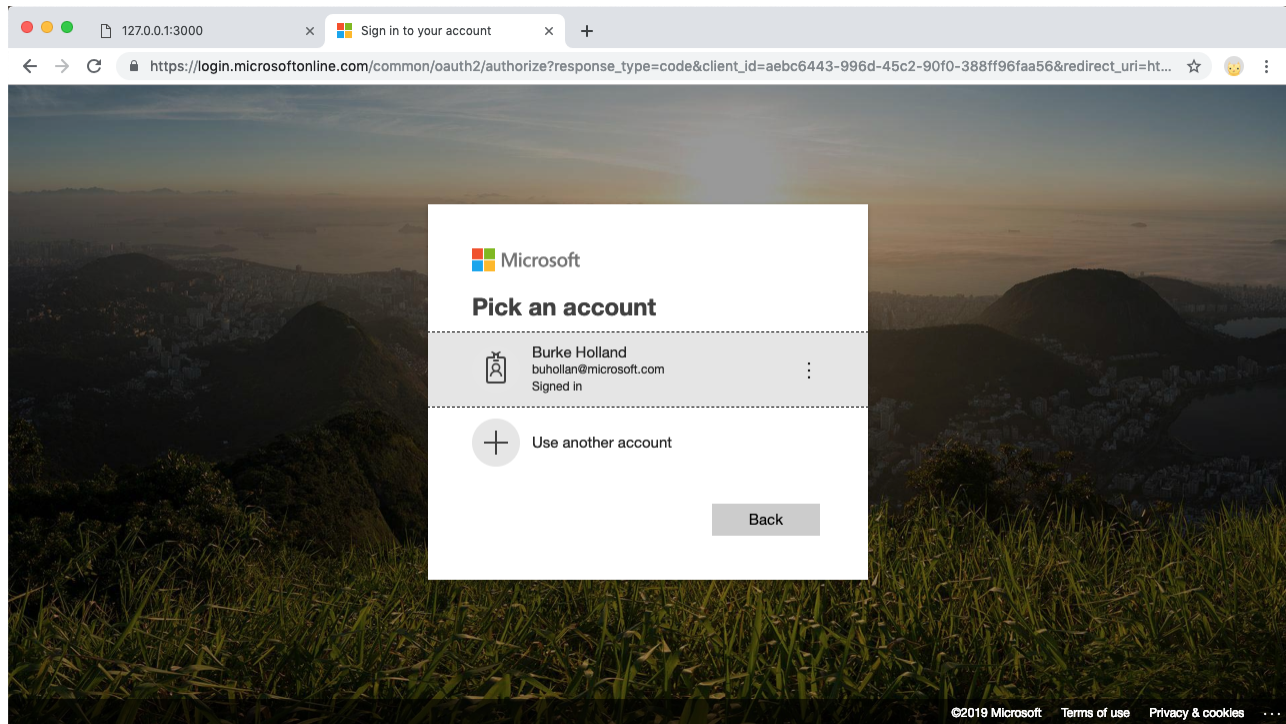


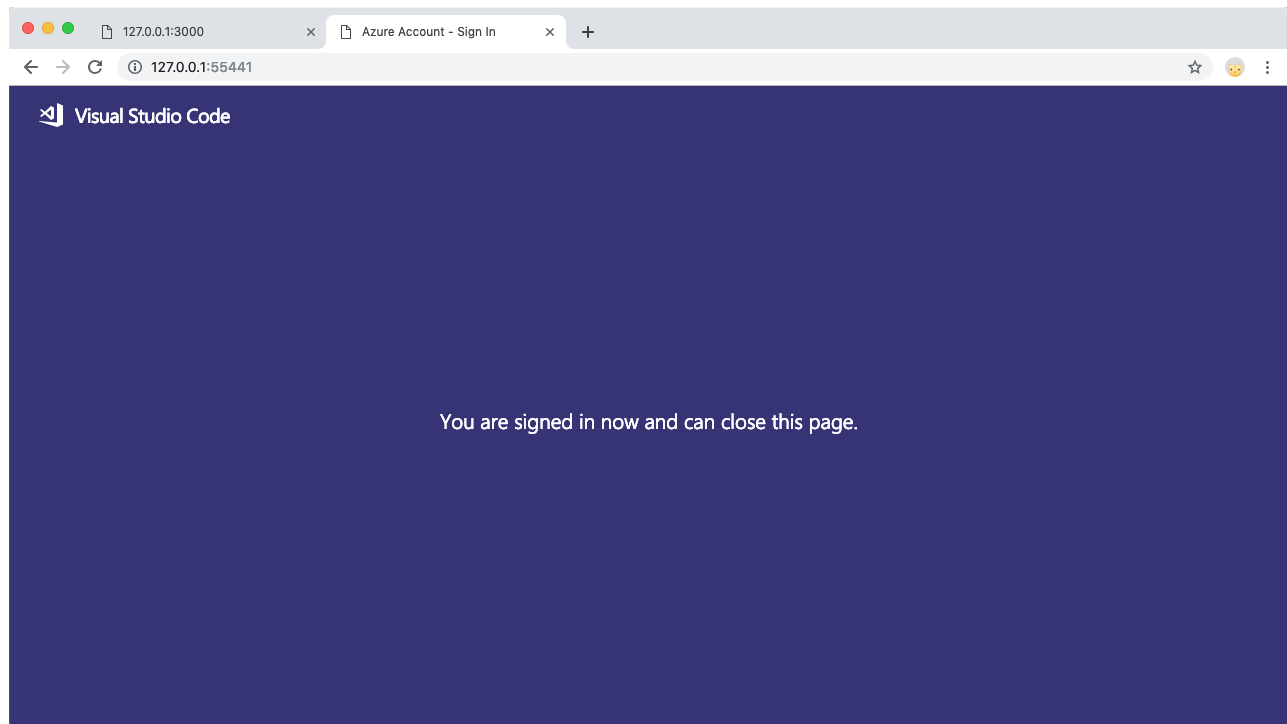
If you click on that icon, you'll get a new explorer view for Azure App Service.



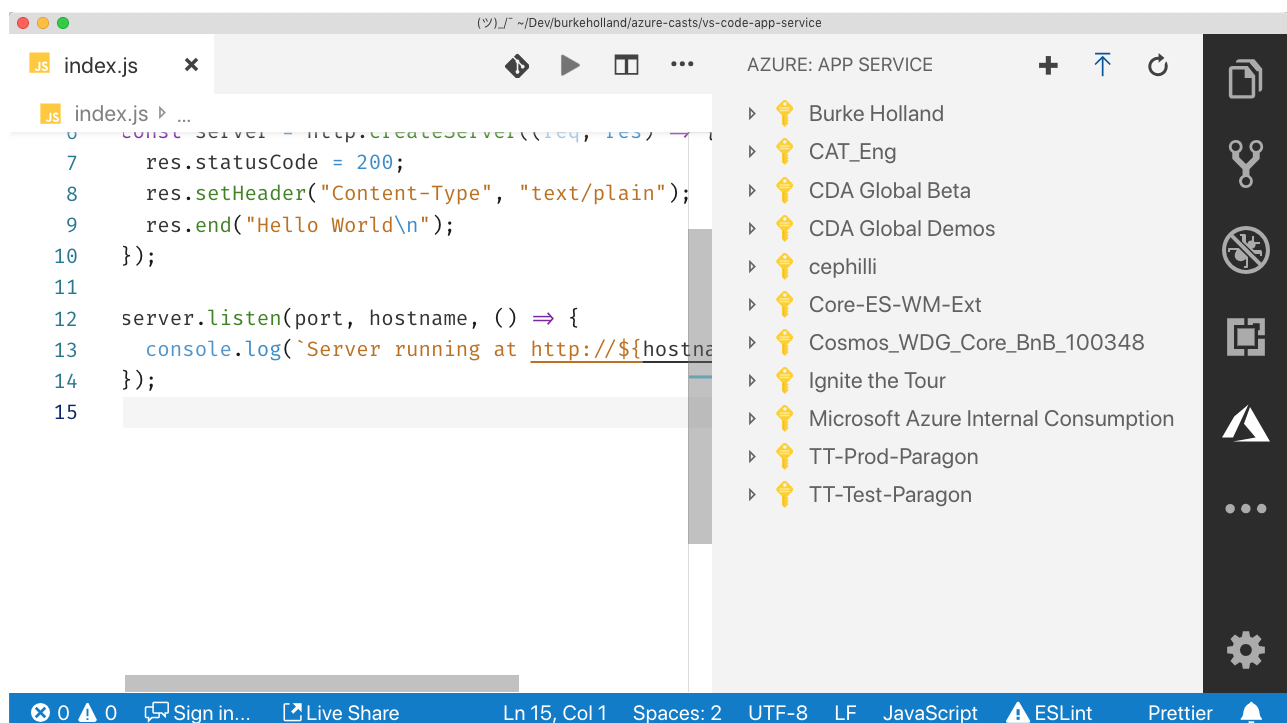
There are other Azure extensions that you can install, and they will all show up right here. Right where the Azure Icon is.

It wants me to sign in, so I'm going to go ahead and do that by clicking on the text that says "Sign in to Azure...". When I do, it's going to launch a browser window asking me to sign in.

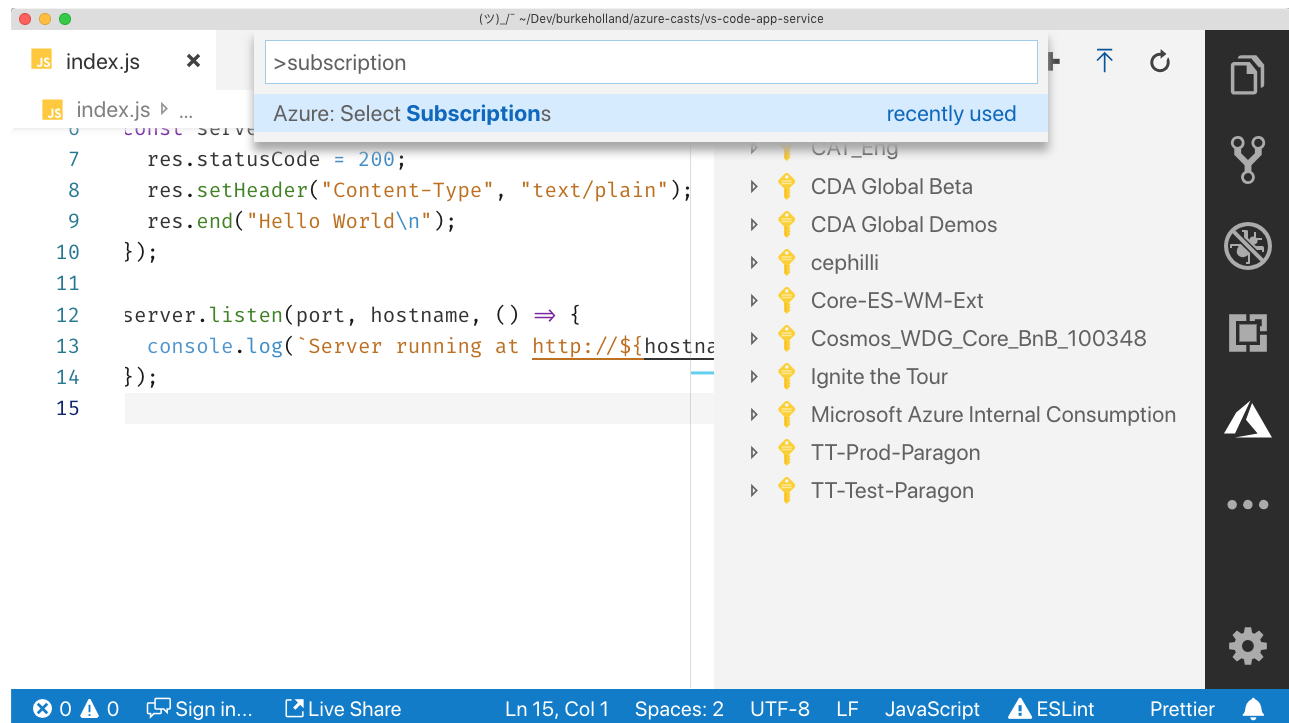




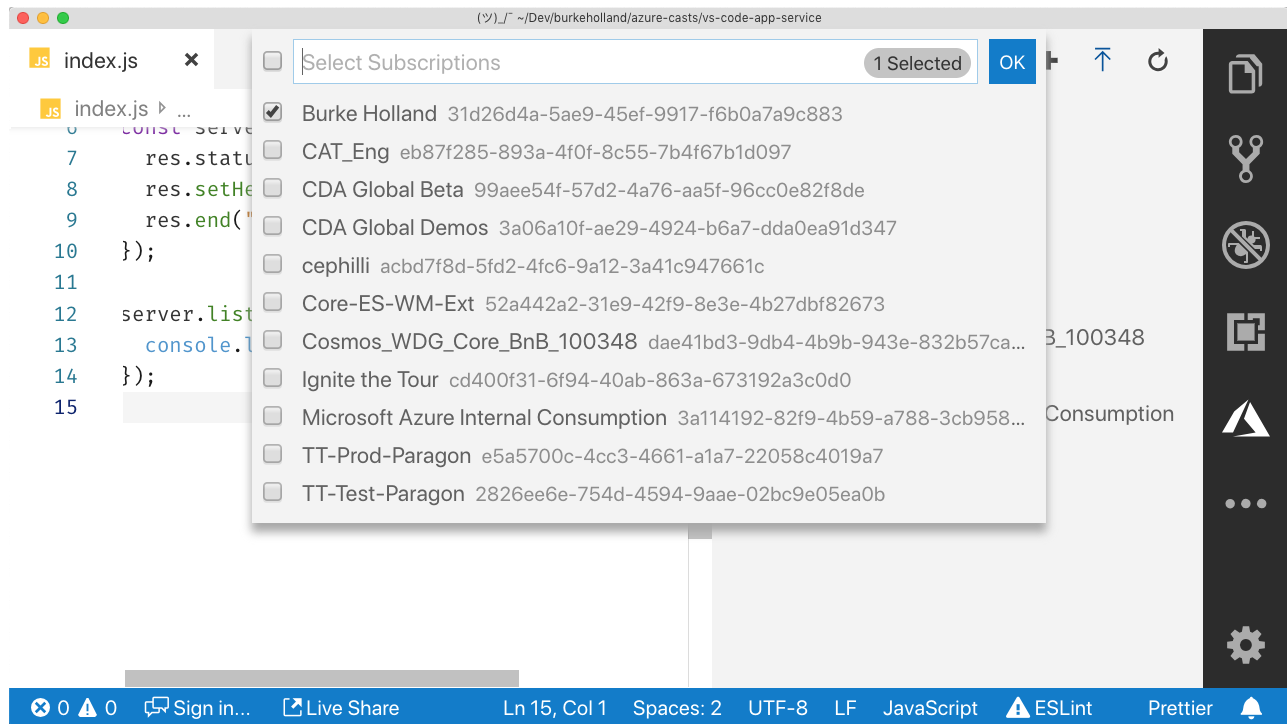
If I go back to VS Code. I now see all the Azure Subscriptions that I have access to displayed with that little key icon.



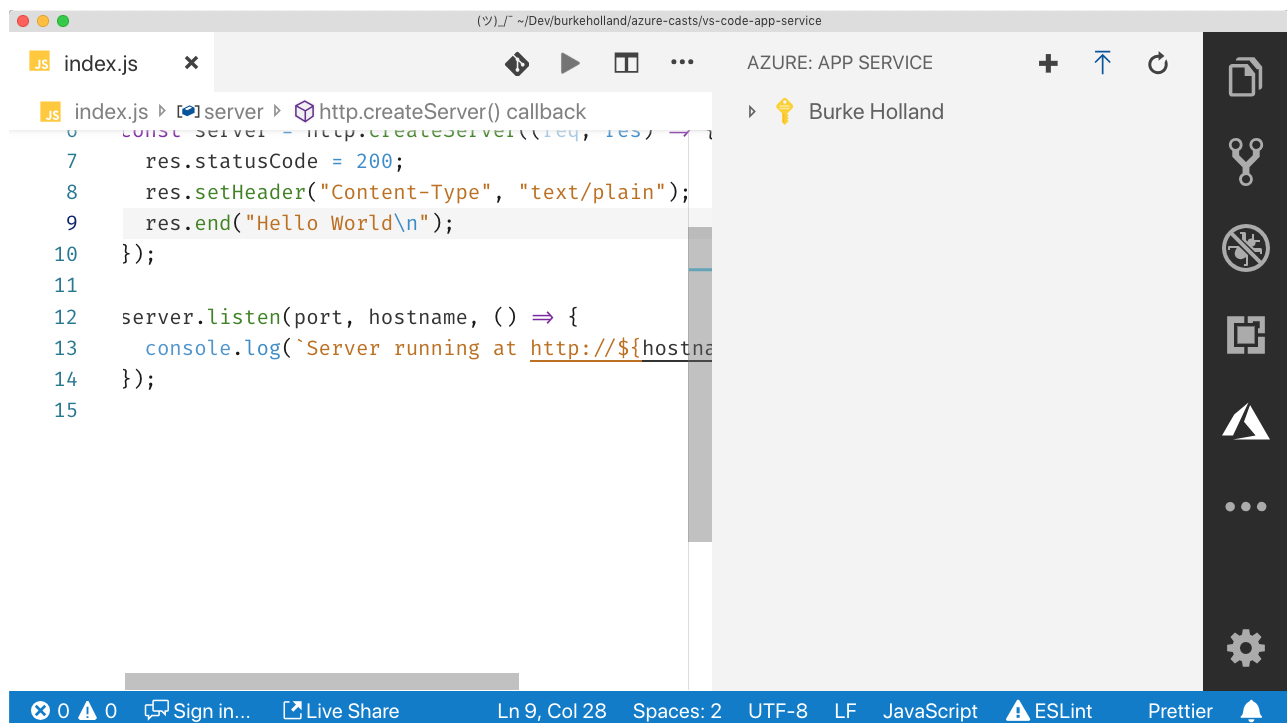
In my case, that's a lot! Too many in fact. It's looking quite cluttered over here. Since I only want to work with my own subscription - that first one there at the top, I'm going to open the VS Code command palette by pressing `Ctrl/Cmd + Shift + P` and I'm going to type "subscription". This will filter the list of options down to "Azure: Select Subscriptions".



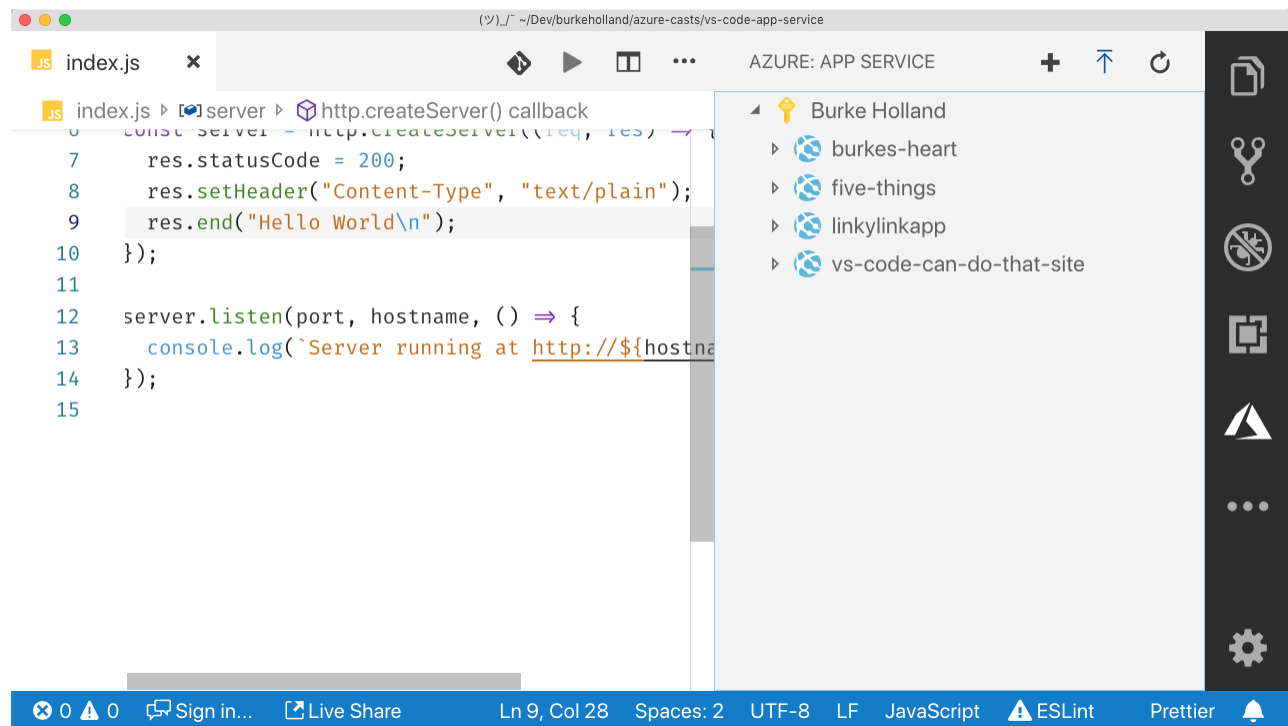
I can click on that and then I get a list of all my available subscriptions. I can deselect them all, and then just pick the one I want to work with, and click the "OK" button.



Now I see just the subscription I selected.

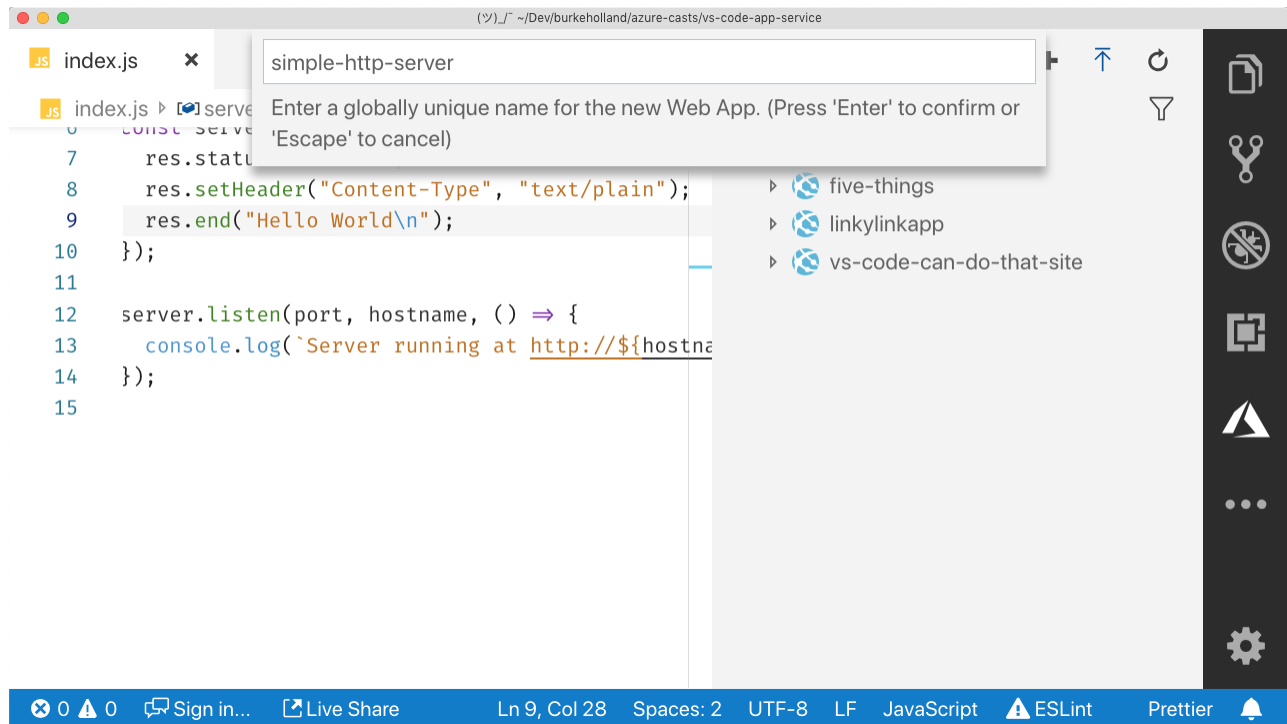


If I expand that subscription, I see all of the current apps I have for this subscription listed out.



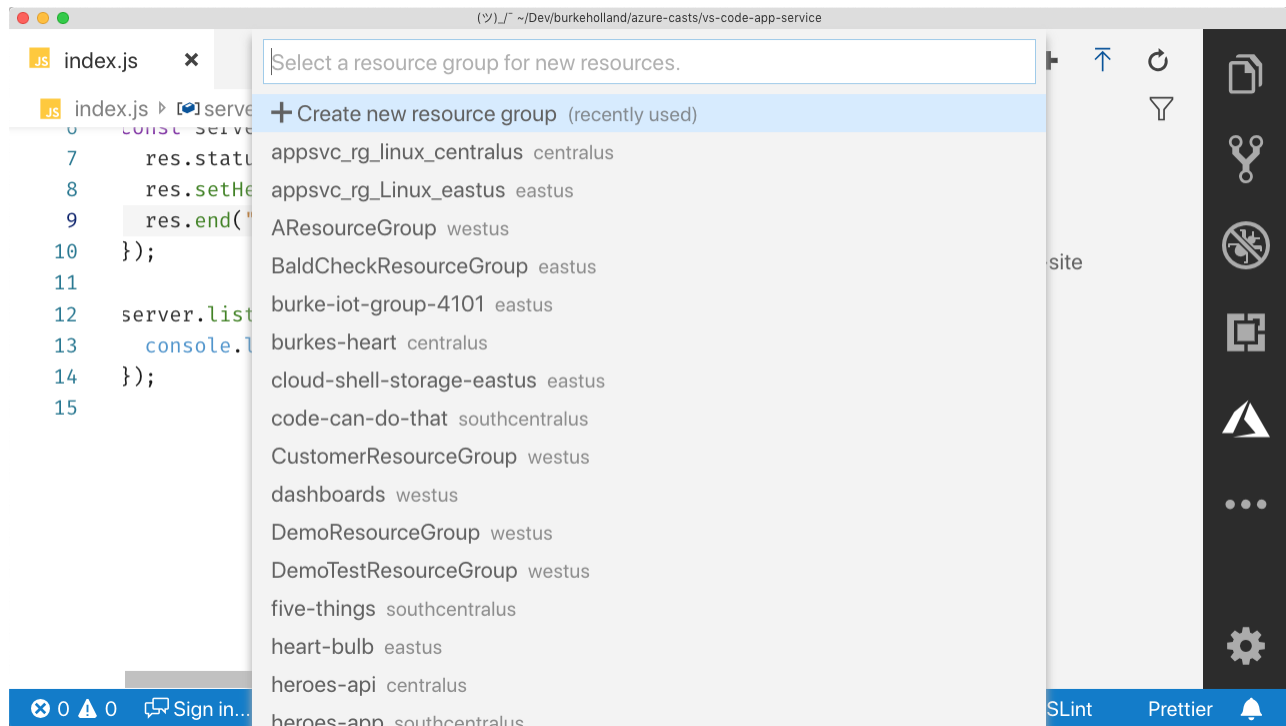
I'm going to create a new application here so that I can deploy this simple Node server. There are a few ways to do that. I can click the plus button up at the top of the Azure App Service extension explorer space. I could also right-click on the subscription and select "Create new web app". Another way would be to open the command palette by pressing `Cmd/Ctrl + shift + p` and search for "create" which will give me the option "Azure App Service: Create New Web App". I could also go back to the File Explorer where my index.js file is and right click it and select "Deploy to Web App" and then select "Create new web app". So many ways to do the same thing. You can pick the one you like the most. I'm going to just click this little plus sign up here.

First we have to pick a name.



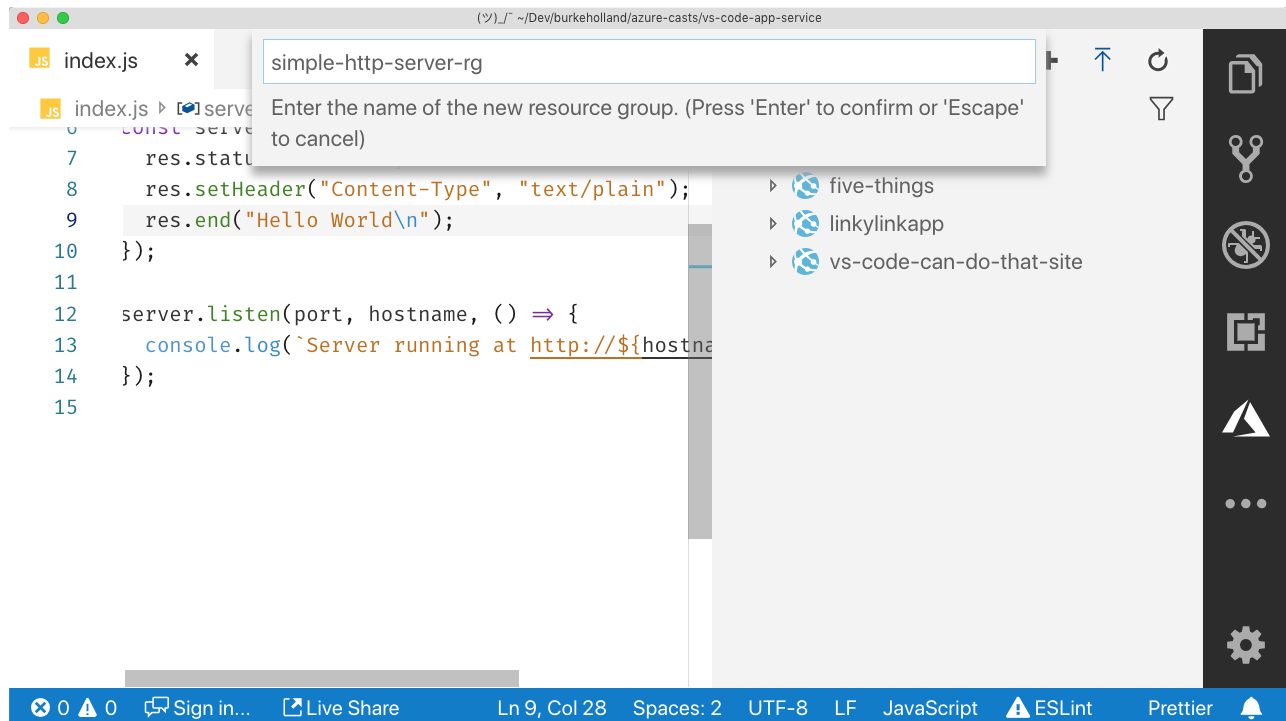
Note that the name you pick has to be unique across all of Azure. That's because after you create this site, you are going to get a URL for it in the format <site name>.azurewebsites.net. And URL's gotta be unique. So your name has to be unique.

The next thing it asks us for is a Resource Group.

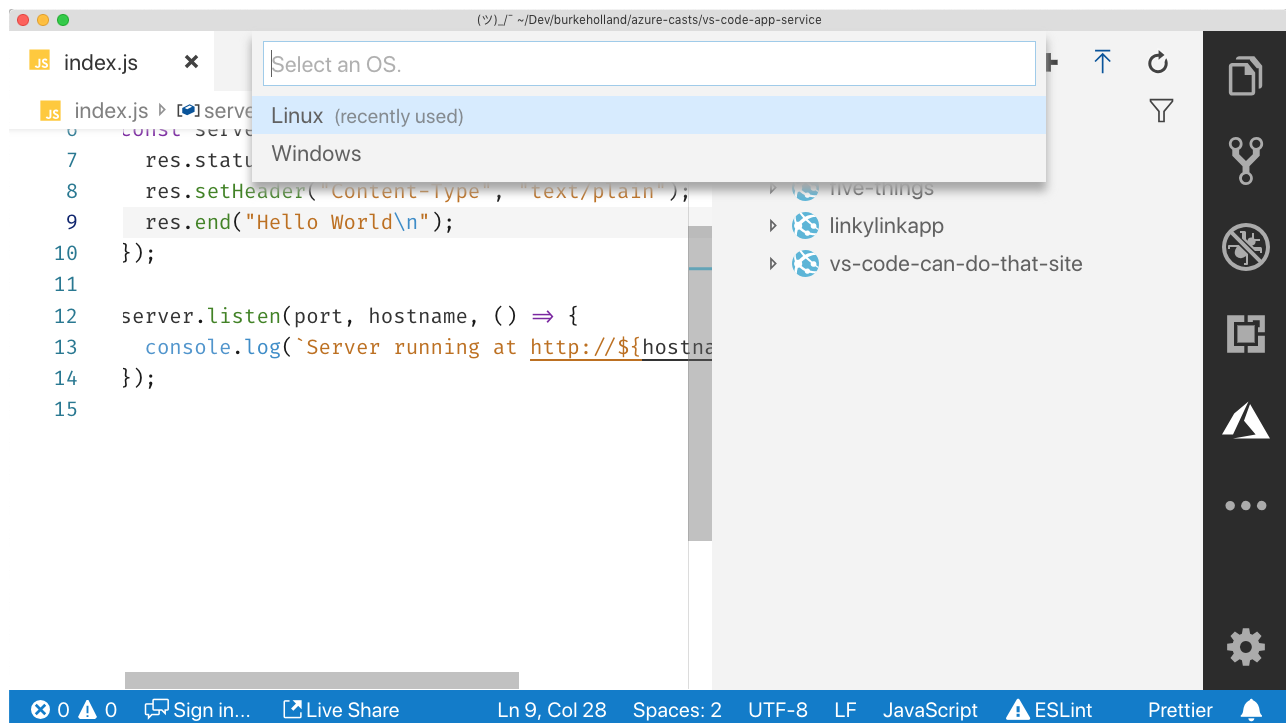


What's a resource group? Resource Groups are a concept that was introduced in Azure a while back. They are simply a logical grouping for your resources. For instance, our app could have a database in Azure, maybe some Serverless Functions in Azure and we would probably want to group all of that stuff together in a new Resource Group. You can always move things between Resource Groups. The other nice thing about Resource Groups is that when you delete the group, everything inside of it gets deleted. You'll find that you end up with a lot of resources the most you use Azure and Resource Groups help you keep it all nice and tidy.

I'm going to create a new Resource Group for my application. We can just give it the same name as our app, plus an "rg" designator on the end. I like to do that just to make them stand out in the Azure Portal as Resource Groups.



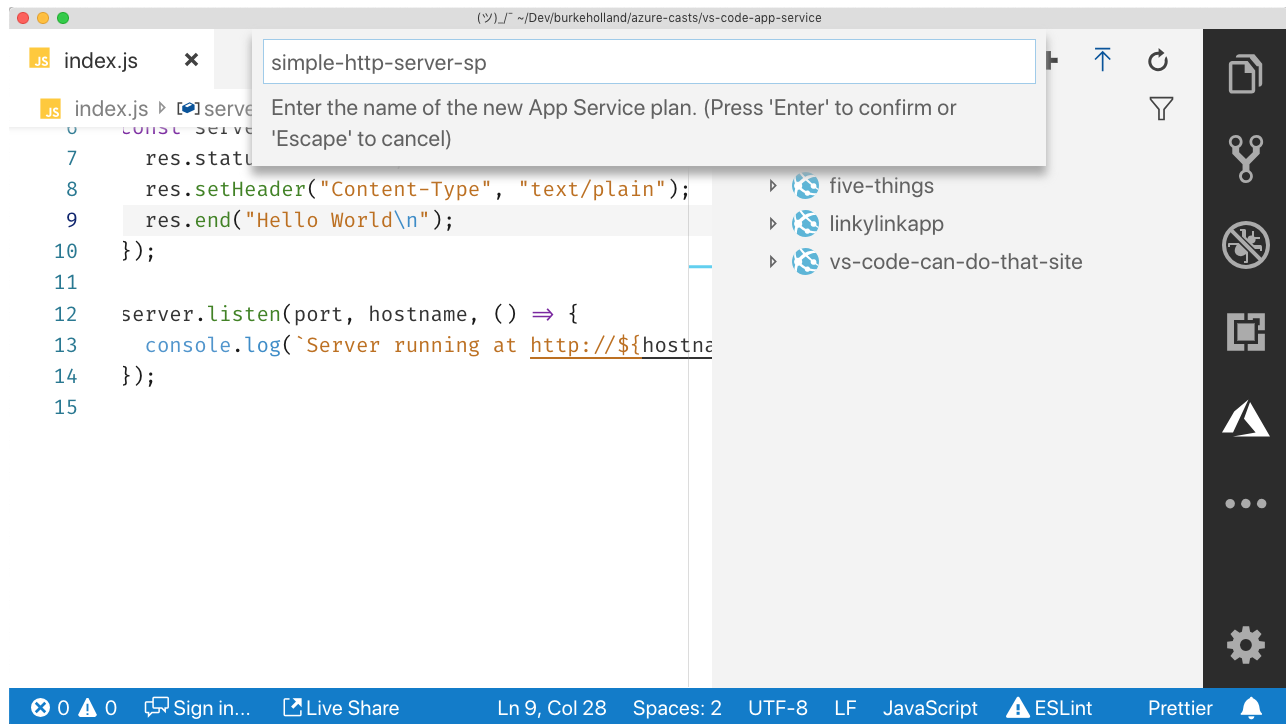
Now it will ask us if we want Windows or Linux. This is exactly what you would think - what sort of operating system do you want your application to run on. I said you don't have to worry about the server - and that's still true. But the way Windows and Linux app service works is different. Therefore this option is important. For this video, we're going to focus on Linux hosting. So I'll select Linux.



Now it wants to know what runtime we want. If you scroll through the list you'll see Node, ASP.NET, Ruby, PHP and Tomcat (Java). We're running a Node app, so we want a Node version. You'll notice that it recommends that we use version 10.10 because that's LTS (Long term support). That's actually not accurate. The current LTS version at the time of this writing is 10.15.3. That means that the highest LTS version available in Azure (as I'm typing this) is 10.14. So that's the one we'll select.

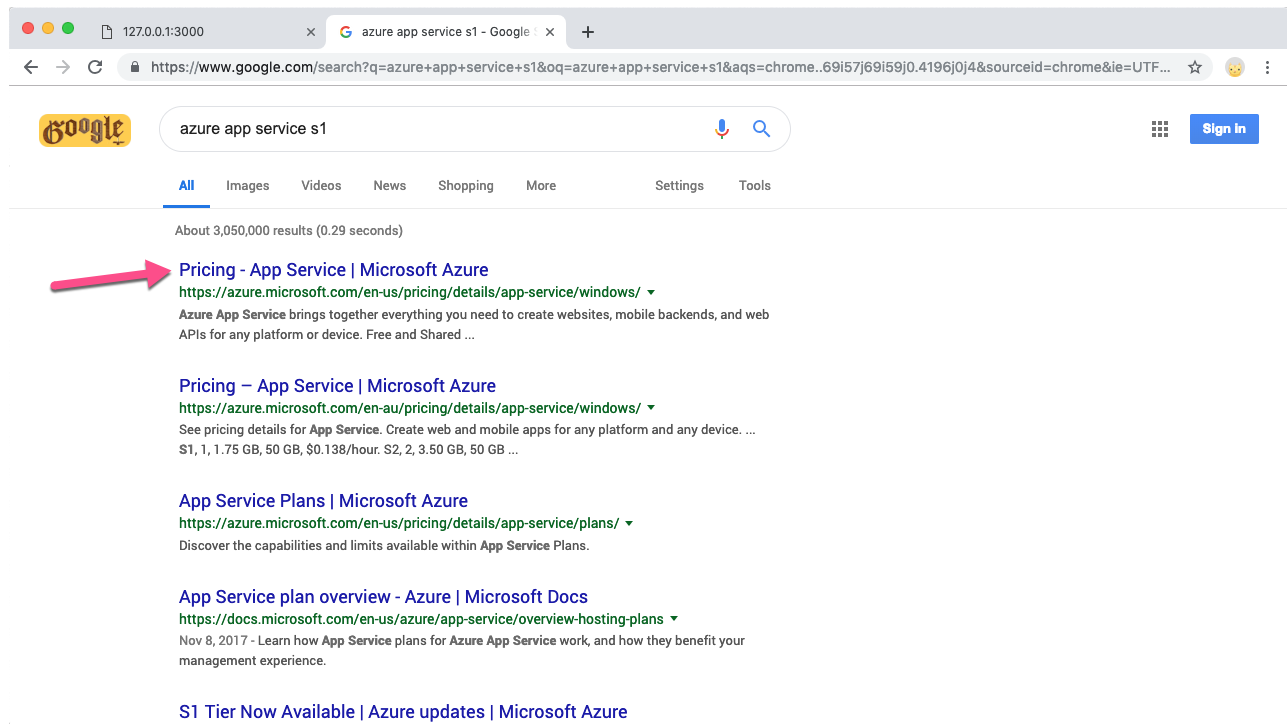
The next prompt will ask you to create an App Service Plan. This is basically you choosing how much "compute" you want. Or rather, how much power. How much octane. Do you want lots of memory and processing power, or do you not need that much? Remember, you pay for compute, so this is where you save or spend money.

I'm going to create a new App Service Plan



Selecting a “tier” is where you select how much power you want. You see here that we have B1, B2, B3 and S1, S2, S3. What do those numbers mean?

If we google “azure app service s1”, the first result is “pricing”.



That link will take us to another page, which looks a bit complex. If we change the operating system to “Linux” and scroll down a bit, things will start to make sense.

Down in the section title “Basic Service Plan”, you’ll see those B1, B2, and B3 identifiers. These are levels of the “Basic” service plan. On each one of these you can see how many CPU cores you get, how much memory, how much storage and how much all of that costs.

127.0.0.1:3000


Pricing - App Service | Microsoft

+

← → ↻

https://azure.microsoft.com/en-us/pricing/details/app-service/linux/

☆ 🧑

 Microsoft Azure

Contact Sales: 1-800-867-1389

Search

My account

Portal

Sign in

Overview

Solutions

Products

Documentation

Pricing

Training

Marketplace

Partners

Support

Blog

More

Free account

OS/Software

Linux

Region:

Central US

Currency:

US Dollar (\$)

Display pricing by:

Hour

Basic Service Plan

The Basic service plan is designed for apps that have lower traffic requirements, and don't need advanced auto scale and traffic management features. Pricing is based on the size and number of instances you run. Built-in network load balancing support automatically distributes traffic across instances. The Basic service plan with Linux runtime environments supports [Web App for Containers](#).

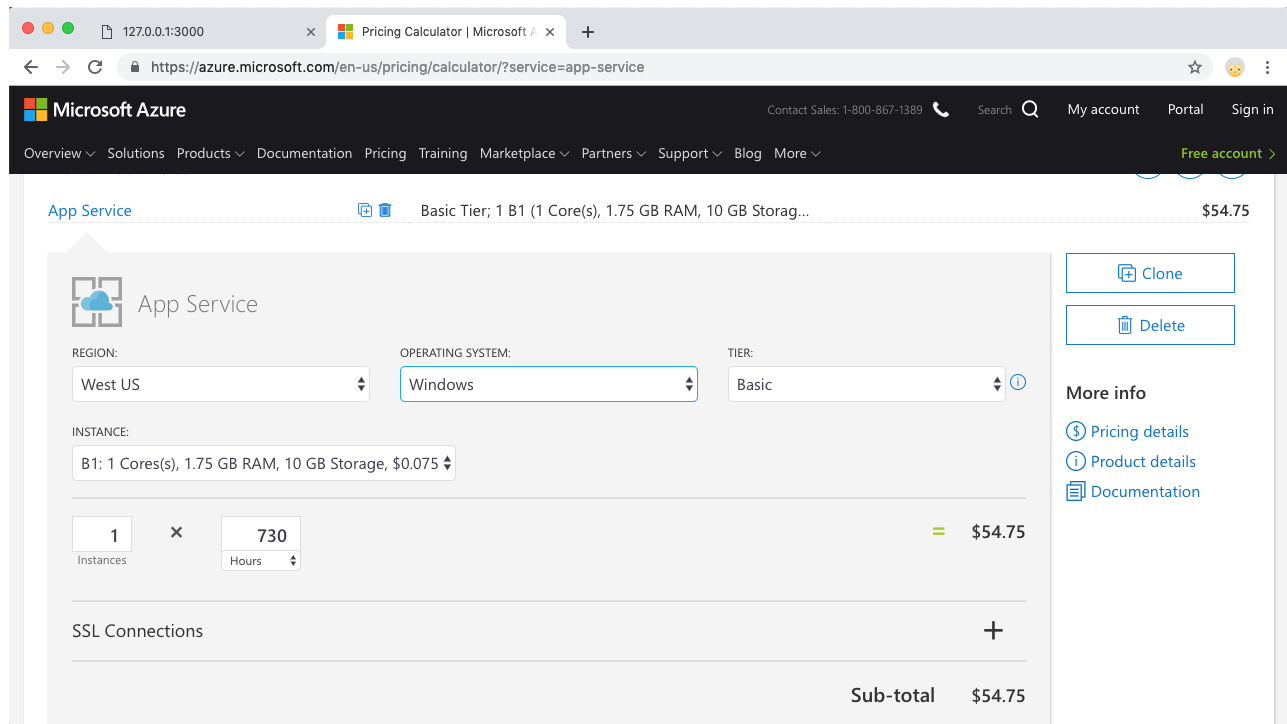
INSTANCE	CORES	RAM	STORAGE	PRICES
B1 ¹	1	1.75 GB	10 GB	\$0.052/hour
B2	2	3.50 GB	10 GB	\$0.104/hour
B3	4	7 GB	10 GB	\$0.208/hour

¹The Linux Free Grant only applies to the first 722 hours of usage for the first B1 core in a new subscription. It does not apply to other core types or additional B1 cores used. The grant expires after the first month.

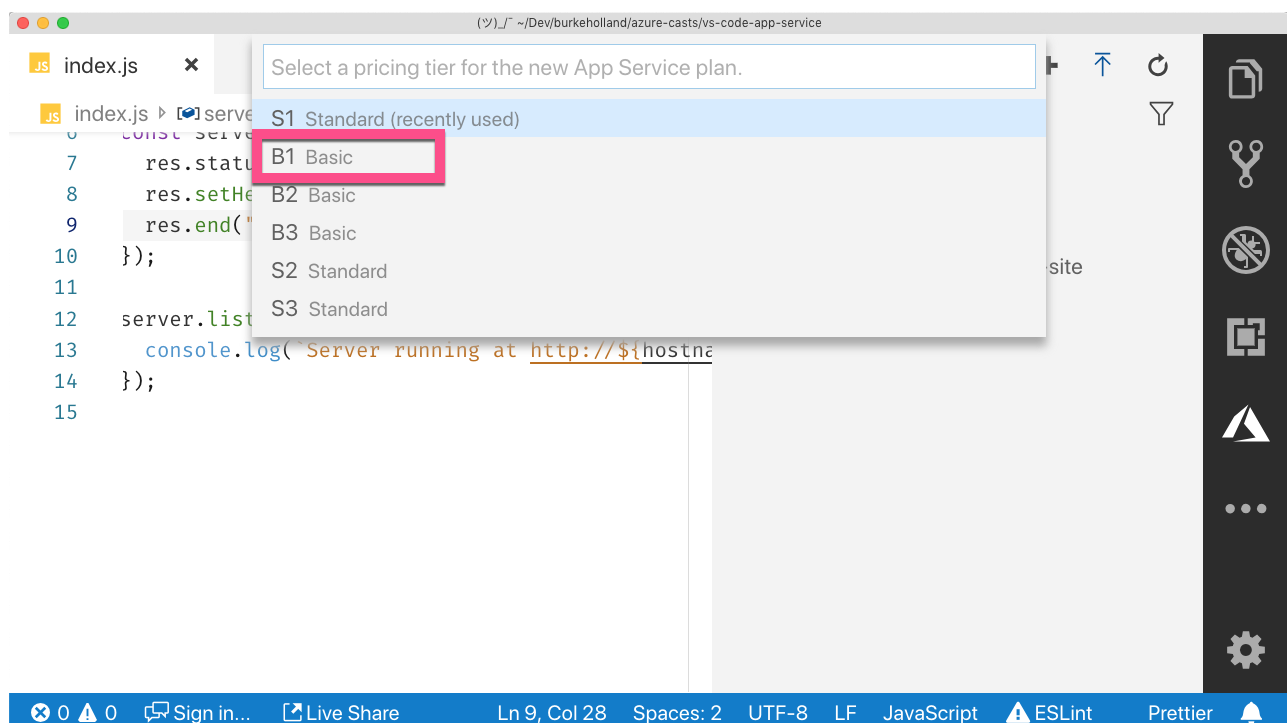
Standard Service Plan

Chat live with an agent

For the B1 plan, we're basically looking at approximately 38\$ per month. There's another way to figure this out besides doing math here. If you scroll back up and go to Calculator in the nav menu, you get a screen where you can select the product you want to price and then select the operating system and tier. Notice that for App Service, Windows is actually about \$16 bucks more than Linux. Why is that? I don't know, but cheaper is better and we're going with Linux anyway.

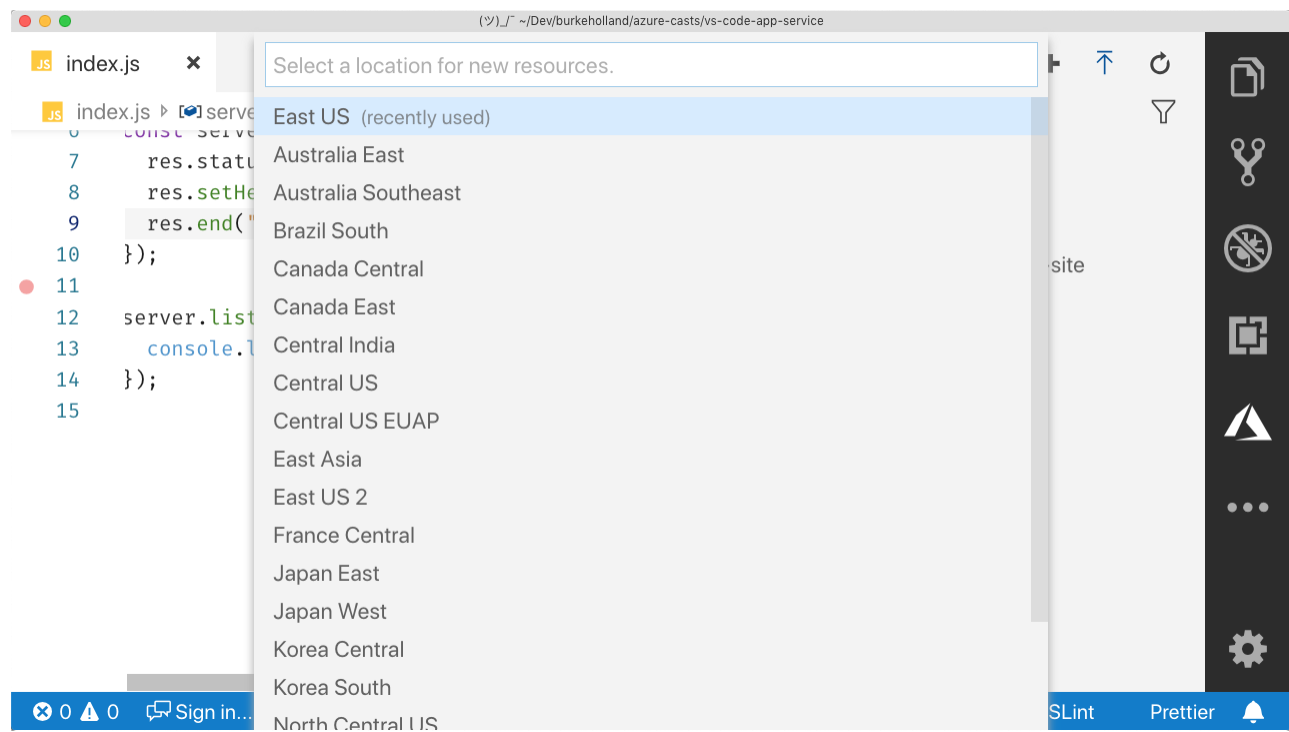


Back in VS Code, we've got a simple HTTP Server, so I think B1 is all the beef we need for this hamburger.

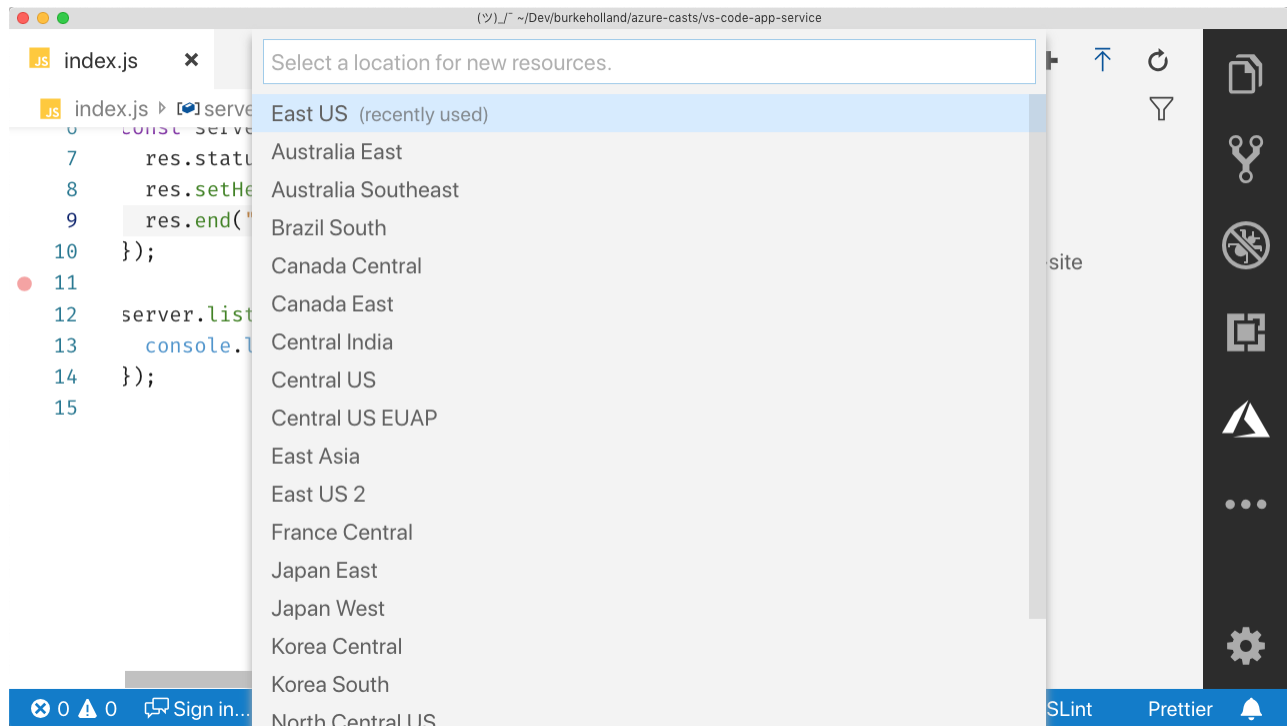


Do note that the size of power of your server will affect your npm install. We're not doing any npm installs here, so we don't care about that. But if we had a big project with a big npm install, it could take quite some time on something like the B1 tier.

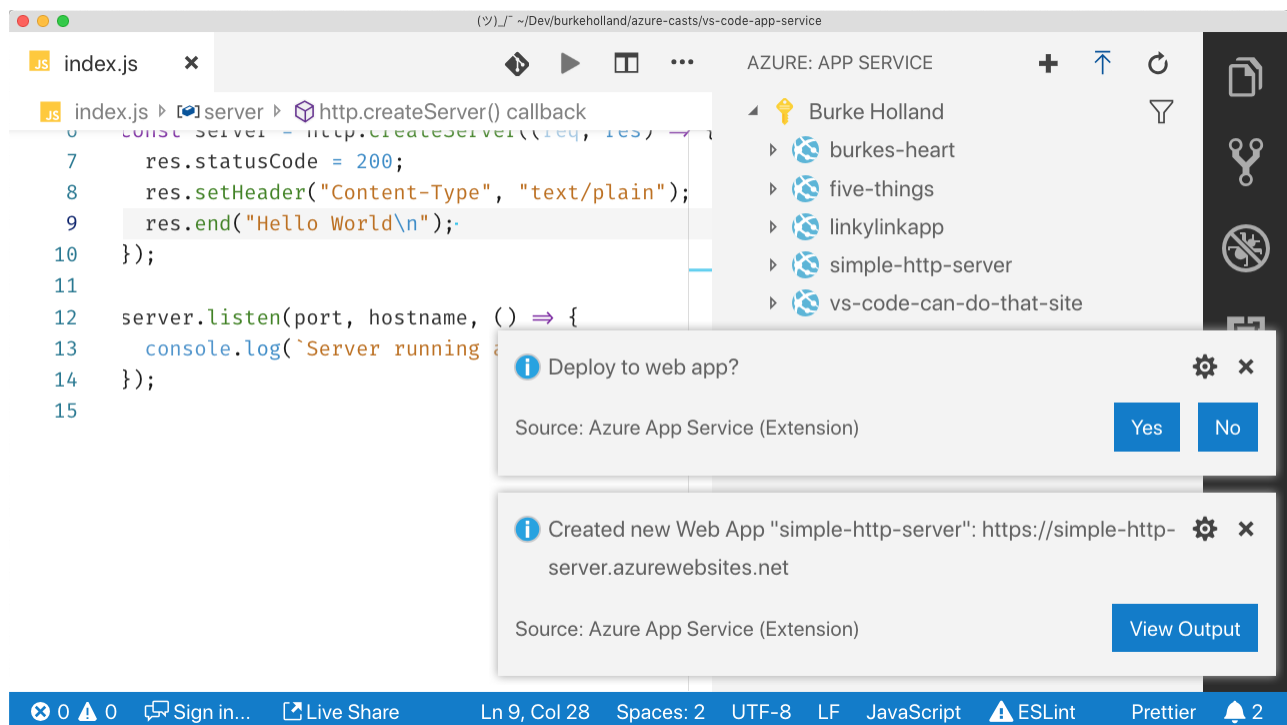
Now we need to select our data center. I'm going to pick the one that's closest to me. You do you.



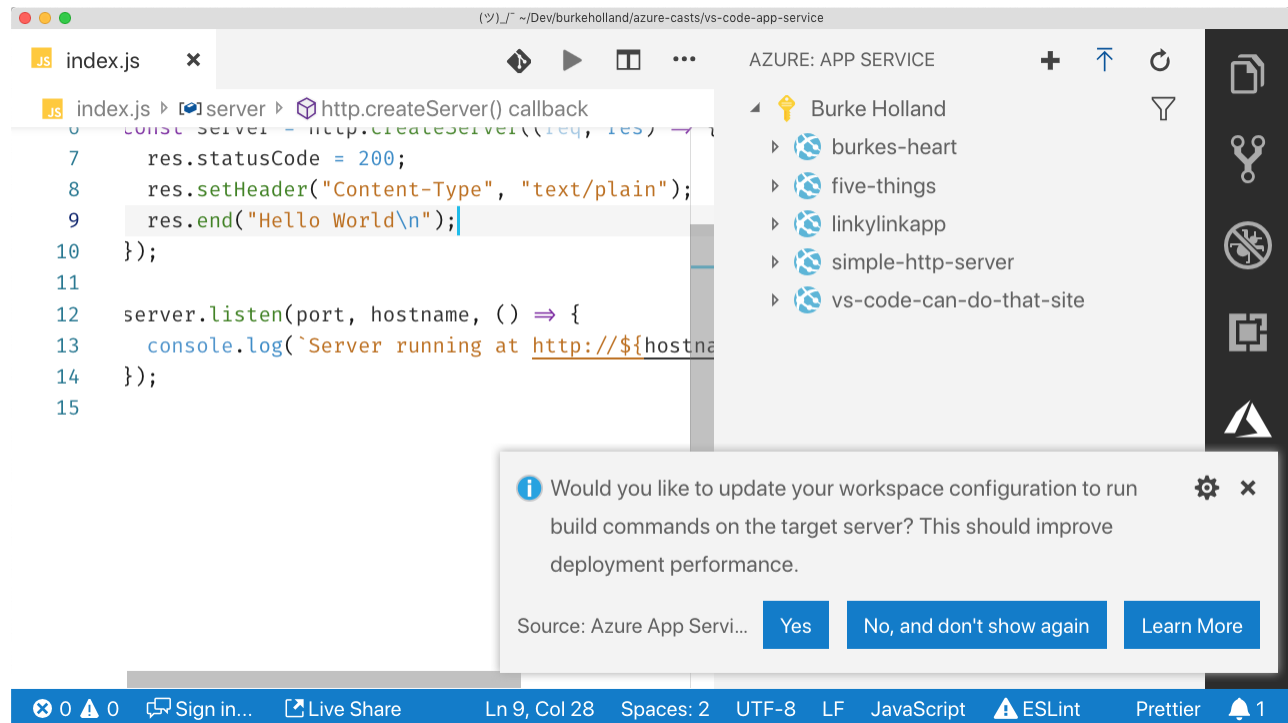
And now VS Code will start to create your new App Service site...



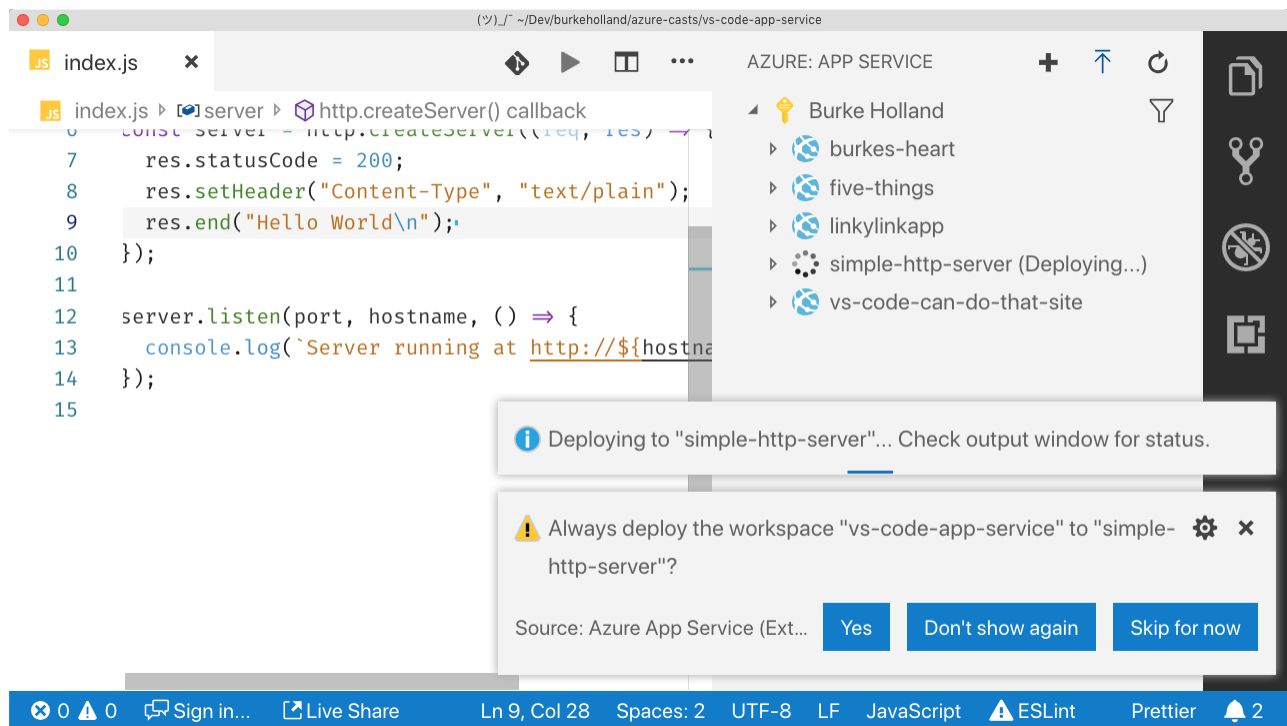
And when it's done, it's going to ask you if you want to deploy it....



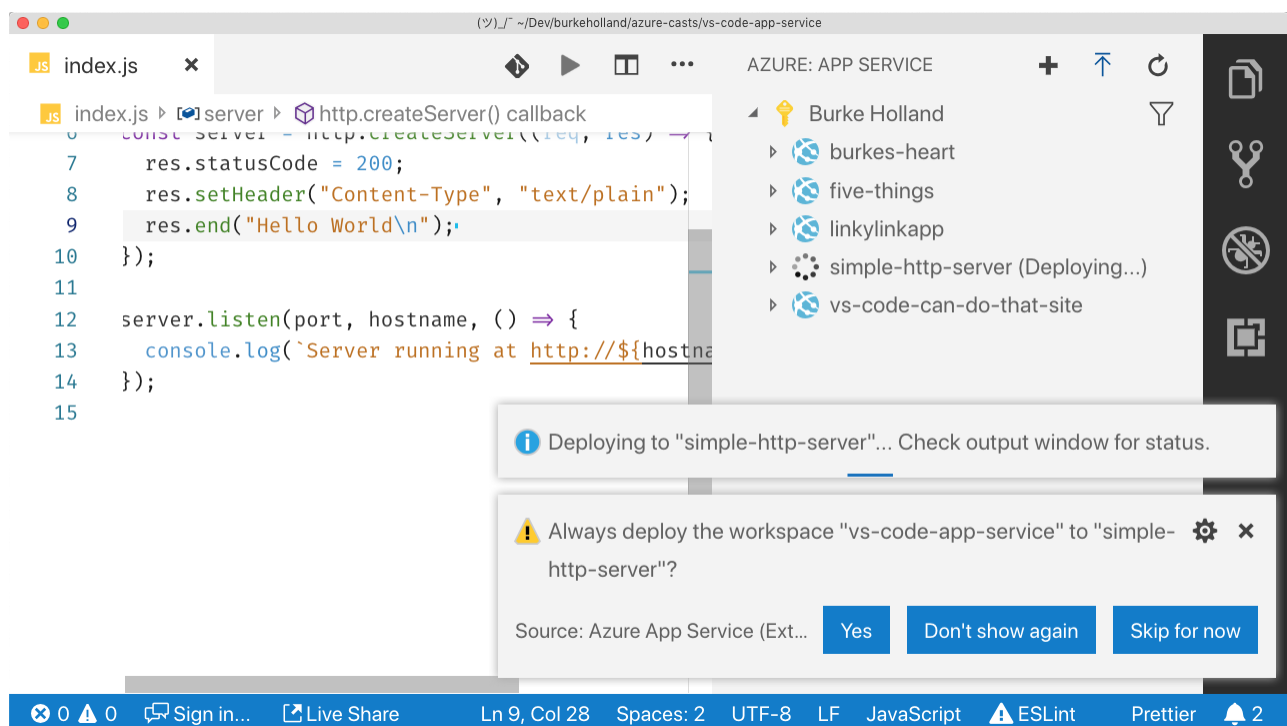
Go ahead and say “Yes”. Then it asks which folder you want to deploy. Just select the current folder from the menu. After you do this once, it’s going to ask if you want to run npm install on the target server. We’re going to say “yes”, even though we don’t have any npm install to run. We’ll get to that in a minute.



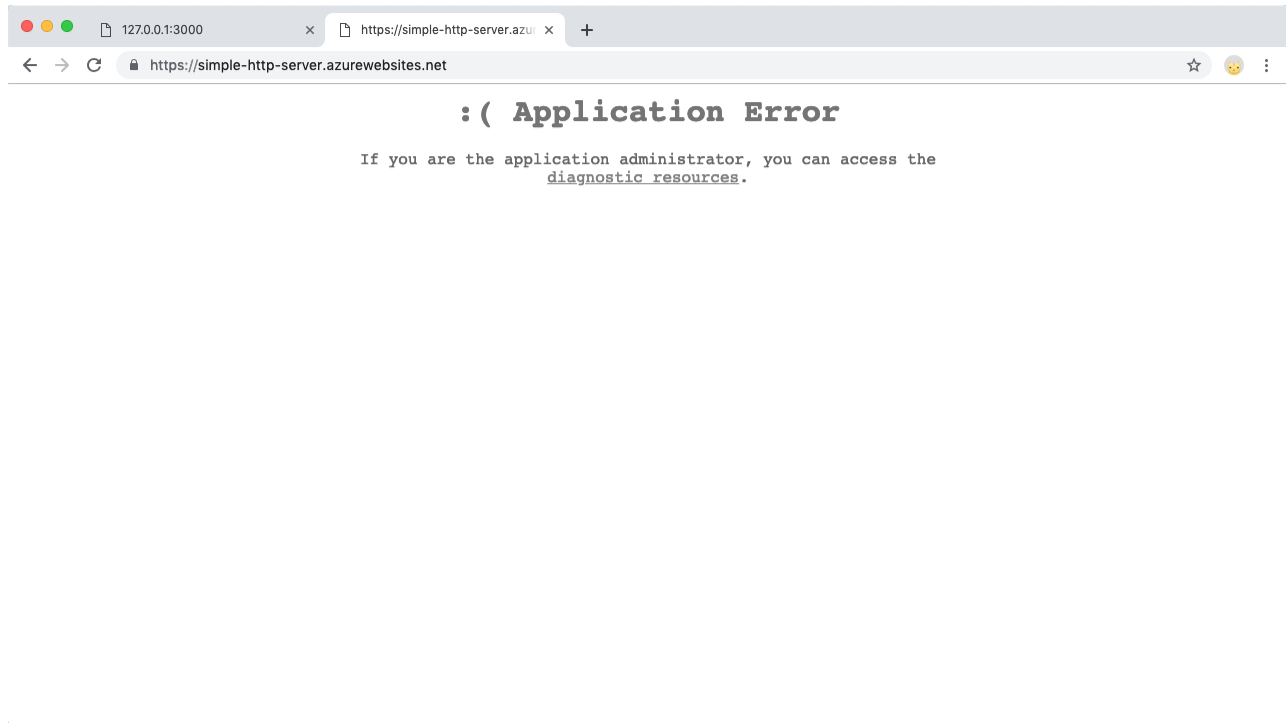
Then it’s going to ask you if you want to always deploy the current directory when doing a deployment. Say yes to that as well so it doesn’t ask us every single time we do a deploy.



And after a minute or so, your site will be deployed, and you can select “Browse Website”.



A browser window will load and...it takes a while to load....quite a long while actually....still loading. But eventually, we get this...



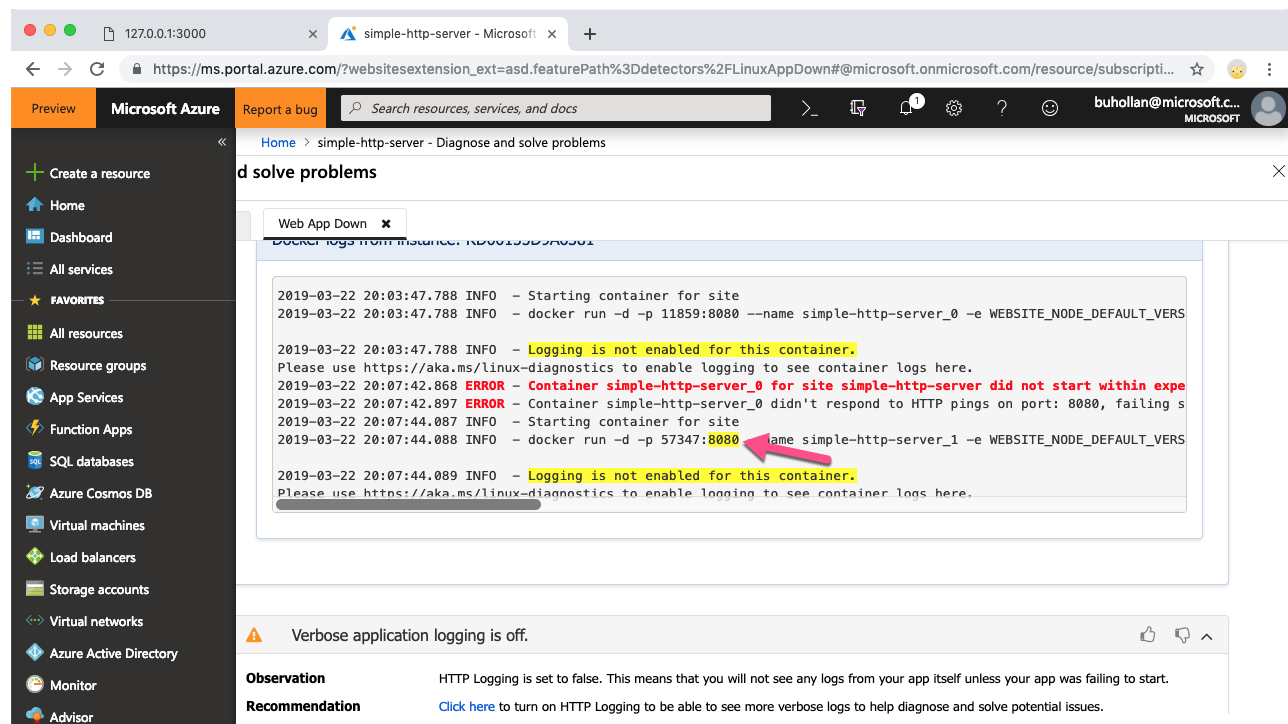
What happened?!? Why is our site not loading? It works locally so why doesn't it work in Azure? Go ahead and click on that "diagnostic resources" link. This will open the Azure portal and go to the "diagnose and solve problems" section for this site.

Scroll down until you see an orange triangle with an exclamation mark. That probably means trouble. Those are our logs. If we look through this section - there are two logs - an application log and a docker log. A DOCKER LOG? What is going on here?

Underneath the hood, Azure App Service is loading your application into a Docker container. You may not know anything about Docker, and that's ok. Just know that we have two log files here. The first one reports on our application, and the second reports on the docker container that our application is running in. If we look at the second log there is an error....

```
2019-03-22 20:07:42.868 ERROR - Container simple-http-server_0
for site simple-http-server did not start within expected time
limit. Elapsed time = 230.6462151 sec
```

It looks like what has happened is that the container that App Service is using to host my Node app didn't start. And the reason for that is highlighted in yellow. Azure is trying to help us here.



The screenshot shows the Azure portal interface. The left sidebar contains navigation links for 'Create a resource', 'Home', 'Dashboard', 'All services', 'FAVORITES', 'All resources', 'Resource groups', 'App Services', 'Function Apps', 'SQL databases', 'Azure Cosmos DB', 'Virtual machines', 'Load balancers', 'Storage accounts', 'Virtual networks', 'Azure Active Directory', 'Monitor', and 'Advisor'. The main content area is titled 'Diagnose and solve problems' and shows a 'Web App Down' alert. Below the alert, a log viewer displays the following logs:

```
2019-03-22 20:03:47.788 INFO - Starting container for site
2019-03-22 20:03:47.788 INFO - docker run -d -p 11859:8080 --name simple-http-server_0 -e WEBSITE_NODE_DEFAULT_VERSION=10.15.0 simple-http-server
2019-03-22 20:03:47.788 INFO - Logging is not enabled for this container.
Please use https://aka.ms/linux-diagnostics to enable logging to see container logs here.
2019-03-22 20:07:42.868 ERROR - Container simple-http-server_0 for site simple-http-server did not start within expected time limit. Elapsed time = 230.6462151 sec
2019-03-22 20:07:42.897 ERROR - Container simple-http-server_0 didn't respond to HTTP pings on port: 8080, failing start
2019-03-22 20:07:44.087 INFO - Starting container for site
2019-03-22 20:07:44.088 INFO - docker run -d -p 57347:8080 --name simple-http-server_1 -e WEBSITE_NODE_DEFAULT_VERSION=10.15.0 simple-http-server
2019-03-22 20:07:44.089 INFO - Logging is not enabled for this container.
Please use https://aka.ms/linux-diagnostics to enable logging to see container logs here.
```

A pink arrow points to the port mapping '57347:8080' in the second log entry. Below the logs, a message states: 'Verbose application logging is off.' Under the 'Observation' section, it says: 'HTTP Logging is set to false. This means that you will not see any logs from your app itself unless your app was failing to start.' Under the 'Recommendation' section, it says: 'Click here to turn on HTTP Logging to be able to see more verbose logs to help diagnose and solve potential issues.'

See that port? There's actually two of them, separated by a colon. This is called port publishing in Docker. What it means is that the port 57347 inside the container is published to port 8080 outside the container. When our application is accessed in Azure, we actually hit port 8080 on the container, which is mapped to port 57347 inside the container. That means that Azure expects our app to run on port 57347. But what port is our app running on? Do you remember?

If we scroll up and look at the log section above, you can see what hostname and port App Service is trying to start our application on.

simple-http-server - Microsoft

https://ms.portal.azure.com/#@microsoft.onmicrosoft.com/resource/subscriptions/31d26d4a-5ae9-45ef-9917-f6b0a7a9c883/resourceGroups/simple-h...

Preview Microsoft Azure Report a bug Search resources, services, and docs

Home > simple-http-server - Diagnose and solve problems

Diagnose and solve problems

Web App Down

```
2019-03-25T15:48:09.041194063Z pm2 launched in no-daemon mode (you can add DEBUG="*" env variable to get more messages)
2019-03-25T15:48:10.017541950Z 2019-03-25T15:48:10: PM2 log: Launching in no daemon mode
2019-03-25T15:48:10.177622657Z 2019-03-25T15:48:10: PM2 log: [PM2] Starting /home/site/wwwroot/index.js in fork_mode
2019-03-25T15:48:10.186727538Z 2019-03-25T15:48:10: PM2 log: App [index:0] starting in -fork mode-
2019-03-25T15:48:10.219959330Z 2019-03-25T15:48:10: PM2 log: App [index:0] online
2019-03-25T15:48:10.238737595Z 2019-03-25T15:48:10: PM2 log: [PM2] Done.
2019-03-25T15:48:10.319108802Z 2019-03-25T15:48:10: PM2 log: 
```

App name	id	version	mode	pid	status	restart	uptime	cpu	mem
index	0	N/A	fork	36	online	0	0s	0%	28.4 MB


```
2019-03-25T15:48:10.319124802Z
2019-03-25T15:48:10.319129602Z
2019-03-25T15:48:10.319134002Z
2019-03-25T15:48:10.319138002Z
2019-03-25T15:48:10.319557806Z 2019-03-25T15:48:10: PM2 log: Use `pm2 show` to get more details about an app
2019-03-25T15:48:10.355701224Z 2019-03-25T15:48:10: PM2 log: [--no-daemon] Continue to stream logs pid=1
2019-03-25T15:48:10.782731879Z 15:48:10 0|index | Server running at http://127.0.0.1:3000/
```

Docker logs from instance: RD00155D9A6381

```
2019-03-25 11:51:13.198 INFO - Starting container for site
2019-03-25 11:51:13.198 INFO - docker run -d -p 47486:8080 --name simple-http-server_0 -e WEBSITE_NODE_DEFAULT_VERSION=10.16.1 --rm simple-http-server
2019-03-25 11:51:13.198 INFO - Logging is not enabled for this container.
```

We're trying to bind to local host and run on port 3000. But Azure is trying to start our application on port 8080. And we have a second and sort of common Docker problem. We can't bind to the host 127.0.0.1 because we're in a Docker container. So let's fix both of these issues.

First, instead of binding to 127, we can bind to 0.0.0.0.



```
1  const http = require("http");
2
3  const hostname = "0.0.0.0";
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader("Content-Type", "text/plain");
9    res.end("Hello World\n");
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostname}:${port}/`);
14 });
15 ..
```

Now you may be thinking - does this still run locally? YES. It does. Those 4 zeros basically mean “no ip address”, so the computer goes ahead and claims it as it’s own. Because it doesn’t exist. This is all part of the IP4 specification, which is great reading if you’re having trouble getting to sleep.

Now let’s also fix out port problem. One way would be to set our port to 8080. But a better way is to set the value of the port to the PORT environment variable. This value is set automatically by Azure, so we can just bind to that. And since we still want it to run locally on port 3000, we can fall back to that in case the environment variable isn’t found.



The screenshot shows a Visual Studio Code editor window with a file named `index.js`. The code is as follows:

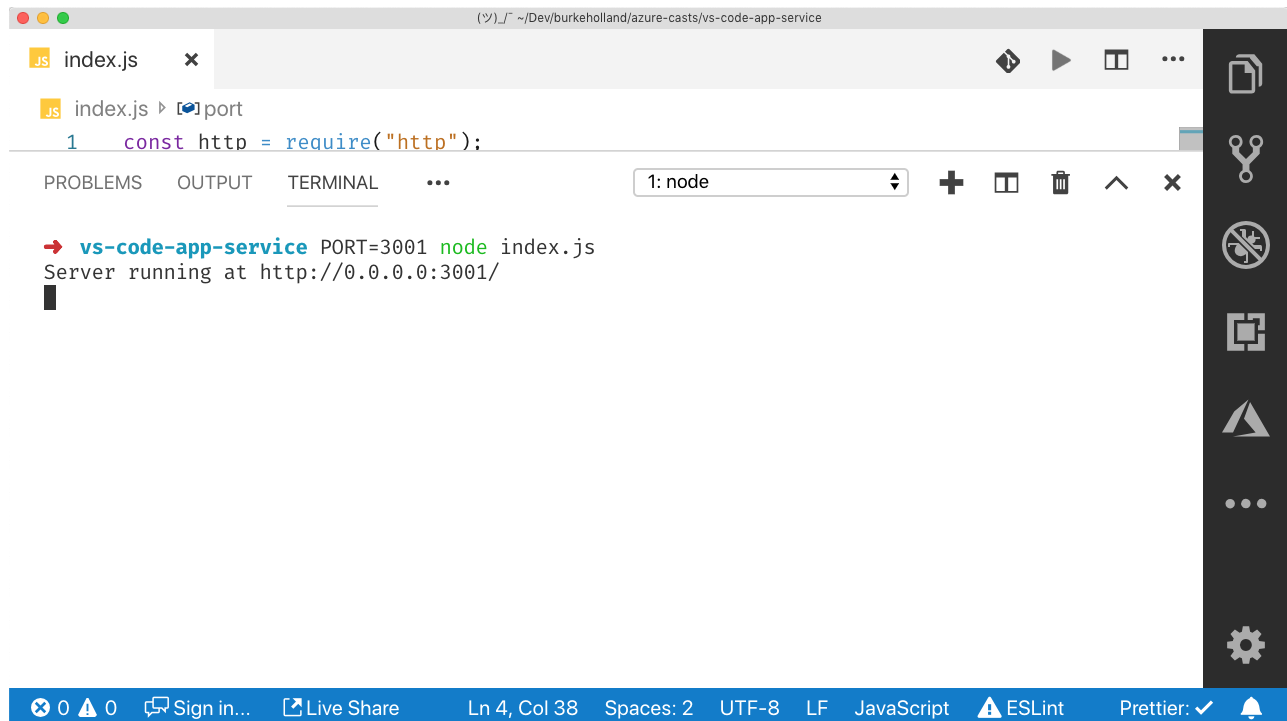
```
1  const http = require("http");
2
3  const hostname = "0.0.0.0";
4  const port = process.env.PORT || 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader("Content-Type", "text/plain");
9    res.end("Hello World\n");
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostname}:${port}/`);
14 });
15
```

The line `const port = process.env.PORT || 3000;` is highlighted with a pink rectangular box. The status bar at the bottom indicates the file is `index.js`, line 4, column 38, with 2 spaces, UTF-8 encoding, LF line endings, JavaScript language, ESLint warnings, and Prettier formatting enabled.

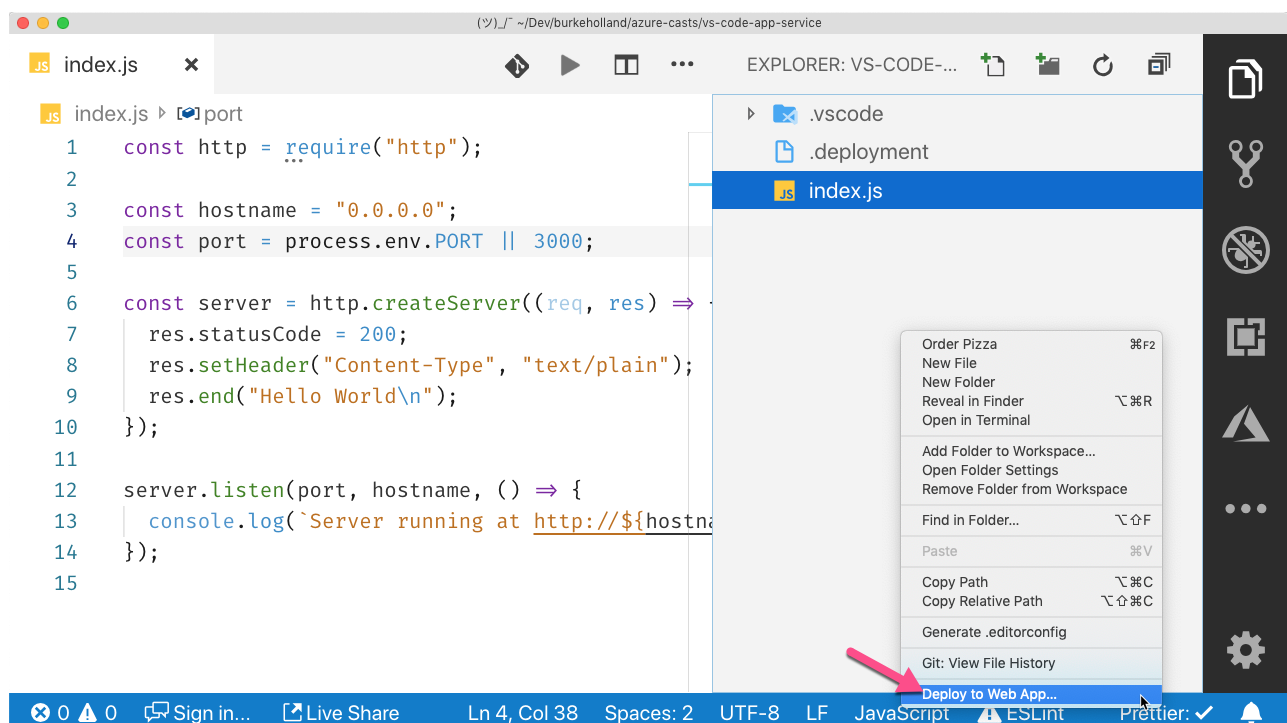
If we wanted to run this locally and pass in that variable, you can run the following from the terminal

```
$> PORT=3001 node index.js
```

And our app will be running on port 3001.



So let's deploy again. We can just right-click anywhere in the empty space on the sidebar and choose "Deploy to Web App".



Now try and browse your site after the deployment finishes. VICTORY!

I hope you enjoyed this video. Make sure to check out the other Azure Casts, and I'll see you again soon.