Deploy Deploy Deploy ….

**So... you've made an app!**

Getting your app working on your machine is the first step in getting it working EVERYWHERE ON THE INTERNET! Today we look at deploying your app so other people can use it.

Today we will be working with the following things:

1. IP addresses and DNS (a.k.a. the internet)
2. package.json scripts
3. environment variables

With a sneak peek of security sprinkled in.

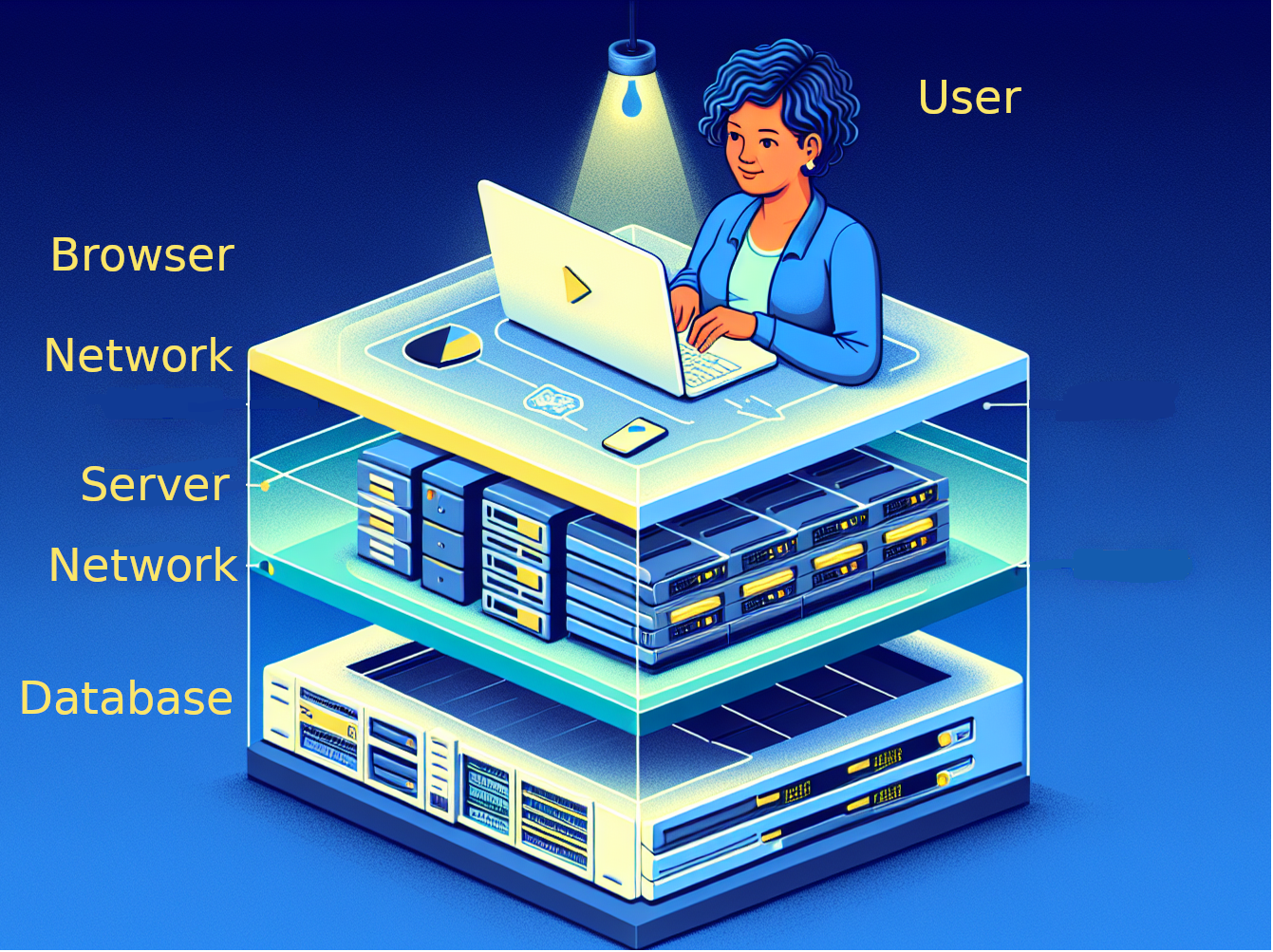
**Hosting options**

Since the rise of the post dial-up internet (about 2005) these options have appeared

* Hook one of your computers to the internet and leave it turned on
* Rent a server from your ISP
* Rent a server from a datacenter provider like Amazon/Microsoft (Azure)/Google Cloud
* Use a deploy service

The final two options are most prevalent today for new services so we will focus on specifics of those methods.

**But first a bit of theory**



Our stack, complete with torso-less user

Now that we will be running our server and database out on the internet, the network is going to play a larger role in our thinking. Up to now we have been using "localhost" to connect our layers together. Once we deploy that changes and we benefit from knowing a bit about how internet addresses work.

### Internet Protocol Addresses and the Domain Name System

Every computer attached to the internet that can accept incoming connections needs to have an IP address. An IP address is like a phone number - it is represented by groups of digits. The old IP address standard is 4 numbers from 0-255 separated by '.' (example 192.168.0.1), the new style IP addresses are 8 numbers from 0-65535 separated by colons (for example [2001:0db8:85a3:8d3:1319:8a2e:370:7348]).

In fact we use IP addresses even on local networks, whenever you join a wifi (or ethernet) network your computer asks for an address to use from the network hub or wifi router. You can find your address in the advanced network settings of your computer.

Fun Fact: localhost is actually the name for 127.0.0.1

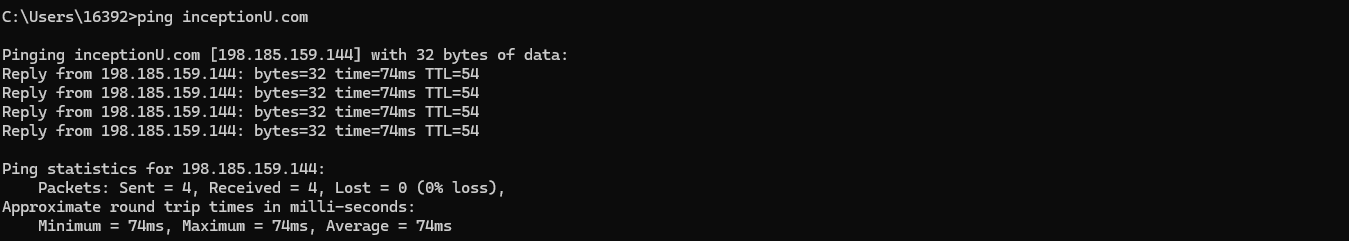
### Look me up!

With so many possible IP addresses the real question is how do you know which one to connect to? Enter the Domain Name System (DNS).

DNS is a standard protocol and a set of servers that our computers use to look up the IP addresses for names like "inceptionu.com" so that our browsers can make connections to the right computer on the internet.

Use the "ping" tool to look up an address on the internet and get the IP. Note that some security locked-down servers may not respond to "ping"

**ping inceptionU.com**



**ping smartcalgary.ca**



From this point forward we will be using hostnames in our connection strings.

## Preparing your app for deployment

When deploying a node/express app, most providers will require your app to conform to a few basic rules - use the following guidelines to ensure your app deploys smoothly in multiple environments. Firstly your app should cleanly assemble itself and start with the following steps:

1. git clone
2. npm install
3. npm run build
4. npm start

This means you may have to adjust your "scripts" in your package.json file to make sure the right things are happening.

Most importantly:

**Your http server should listen on the port specified in the environment variable "PORT"**

Fun Fact: most deployment providers use "bash" to run your commands. Make sure your scripts work in that shell for an easy road on deployment servers. This is why I always use bash.

### Speaking of the environment...

Every deployment provider will allow you to set the "environment variables" that your app will run with. In node we can access the values of environment variables from the process.env global object. You have already seen us using environment variables to get the port configuration in all our server.js files in the line:

const port = process.env.PORT || 3000

Unless you are deploying your database directly on the computer that is doing the hosting (which isn't usually the case unless you are provisioning your own machine), you will probably also have an environment variable that contains the database connection string. You can find ours in db.js

const mongo\_uri = process.env.MONGO\_URI || 'mongodb://localhost:27017/c13-public-washrooms'

Other things that you might end up with in your environment settings are API keys for services like Google Maps, because you would use one set of keys for development and a different set of keys for production.

Really anything that you want to customize at deploy time will probably end up in an environment variable, they are broadly supported.

Environment variables are strings. If you need them to be a number or other type in your code you will need to parse them

Often our build tools will use a package to help us manage our environment variables called "dotenv"

## Making the Public Washrooms project... public.

Armed with this knowledge lets deploy the public washrooms project.

### Tony's no-fail deploy process

Well, no-fail is a bit of an oversell... the point is really just to spread the failures out so that you are only introducing one new variable at a time.

**Step 1:** Make sure it works locally

git clone  
npm install  
npm run build  
npm start

Point your browser at localhost:3000 and make sure the app "works" with data already loaded in your local mongo.

In order to make this step work I needed to do two things:

1. mess around with package.json to make sure npm install and npm run build switched into the client and server folders to do their work.
2. add a "static" handler to serve the output of the vite-based build instead of the old html/css/js that we started project 2 with.

app.use(express.static('../client/dist'))

**Step 2:** Make sure it works with the REMOTE database

Set up an atlas, load your data into it and get your local server pointed at it.

export MONGO\_URI="some atlas url"  
npm start

In order to make this step work I noticed that I was missing the 2dsphere index in my "washroomData.js" file.

**Step 3:** Make sure it works with a different PORT

export PORT=4000  
npm start

**Step 4:** Set up your deployment project

Go to your service provider and sign-up.

Create a project that deploys a node server from github.

Set the environment variables to what you currently have from Step 3.

**Try your first deploy!**

Look at the logs to see if your checkout/build/start went smoothly.

**Step 5:** Check it out on the internet

Your provider will assign you a DNS name, so you can try that instead of "localhost:3000" to get online.

Any database connection issues may become apparent at this step.

You will need to add your application server to the whitelist for your database cluster.

**Step 6:** Triumph!

Deploy again. Just to make sure.

### Other considerations

Now that you have an app on the internet a few directions to improve your deployment are open to you.

1. You probably want a "real" domain name for your app. That involves buying a name from a registrar like godaddy, setting up your name records to point at your deployed site, and telling your hosting provider to respond to that name in an HTTP request. All of these steps are specific to your hosting provider and your name provider.
2. We enabled open access to the internet for our mongodb. Superior approaches include:
   1. Deploy with your mongo "behind the firewall" with your app
   2. Deploy with a VPN that connects your server to your remote mongo
   3. Use a network whitelist on the mongo side to ensure only your server is allowed to connect from the internet (we can do this with render)

