



Robert Tod

[Follow](#)

Javascript Engineer working at Qubit

Dec 20, 2016 · 10 min read

Tutorial: Creating and managing a Node.js server on AWS, part 1

Intro

There are many ways to run servers and plenty of cloud platforms to do so. To start this tutorial series I am going to go with managing an EC2 server using AWS (Amazon Web Services), which is down to the metal and probably the most common way to run Node.js applications.

This tutorial series is intended to help get a better understanding of cloud based servers, though there will be small parts showing how to set up a simple Node.js app.

I find that most tutorials for configuring AWS or cloud servers often expect quite a bit of knowledge and can be quite daunting. For the most part there is quite a bit of reading between the lines to get through. I hope that this is quite simple to follow. To make it as simple as possible I will be writing it with Mac users in mind but if you are using Linux then it won't be much of an issue, we will just be using bash and a text editor.

This part will cover

- starting an AWS server
- SSH into your server
- installing Node.js
- creating a public HTTP endpoint that responds with a static message


Creating and starting the server


If you haven't created an AWS account, open <https://aws.amazon.com/>, and then choose *Create an AWS Account*.

Follow the steps to get an account setup.

Once logged into your AWS account we are going to create an EC2 machine. EC2 stands for *Amazon Elastic Compute Cloud*. The compute cloud is just a load of computers in Amazon's datacenters that are connected to the internet and can be controlled by using the AWS dashboard. We are going to take one of those machines and set it up for our use.

Click **services** on the navigation bar and select EC2, which is under the **compute** category. Click **launch instance**. You will be presented with a bunch of options which are various different images. An image is an exact copy of a hard drive that can be easily loaded onto an empty hard drive, in this case they are being used as presets to get your machine setup easily. Without at least an operating system and SSH, it wouldn't be possible to even configure the instance so some preset software is necessary.


Services
Resource Groups


Robert Tod
Ireland
Support

1. Choose AMI
2. Choose Instance Type
3. Configure Instance
4. Add Storage
5. Add Tags
6. Configure Security Group
7. Review

Step 1: Choose an Amazon Machine Image (AMI)


[Cancel and Exit](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Quick Start

My AMIs
AWS Marketplace
Community AMIs

☐ Free tier only ⓘ



Amazon Linux
Free tier eligible

Amazon Linux AMI 2016.09.0 (HVM), SSD Volume Type - ami-9398d3e0

Select
64-bit

The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.

Root device type: ebs Virtualization type: hvm



Red Hat
Free tier eligible

Red Hat Enterprise Linux 7.3 (HVM), SSD Volume Type - ami-02ace471

Select
64-bit

Red Hat Enterprise Linux version 7.3 (HVM), EBS General Purpose (SSD) Volume Type

Root device type: ebs Virtualization type: hvm



SUSE Linux
Free tier eligible

SUSE Linux Enterprise Server 12 SP2 (HVM), SSD Volume Type - ami-bb7b2dc8

Select
64-bit

SUSE Linux Enterprise Server 12 Service Pack 2 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled.

Root device type: ebs Virtualization type: hvm


Ubuntu Server 16.04 LTS (HVM), SSD Volume Type - ami-0d77397e

Select
64-bit


Ubuntu Server 16.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).

Root device type: ebs Virtualization type: hvm

Selecting an image

The options include different versions of Windows and Linux. Most servers run Linux and it is a great choice for developing Node.js, Windows servers are only really useful for specialized applications such as .NET *shudder*. For the basic things we will be doing in this tutorial, there isn't much difference between the Linux images. I chose to go with **Ubuntu Server** because it is widely used and has tonnes of guides and plenty of questions and answers on Stack Overflow.

Once an image has been selected, we need to select an instance type. Notice that this is a **virtual server**, with **virtual CPUs**. Virtual means that although it will seem like we are connecting to and configuring one computer, in fact Amazon will be running multiple instances on the same machine while pretending it isn't, which is great for scaling and pricing. Let's choose **t2.micro** which is eligible for the free tier, so if your account is less than 12 months old you can run your server for free. Thanks Amazon!


Services ▾
Resource Groups ▾
★
🔔
Robert Tod ▾
Ireland ▾
Support ▾

1. Choose AMI
2. Choose Instance Type
3. Configure Instance
4. Add Storage
5. Add Tags
6. Configure Security Group
7. Review

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Filter by: All instance types ▾ Current generation ▾ [Show/Hide Columns](#)

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

| | Family ▾ | Type ▾ | vCPUs ⓘ ▾ | Memory (GiB) ▾ | Instance Storage (GB) ⓘ ▾ | EBS-Optimized Available ⓘ ▾ | Network Performance ⓘ ▾ |
|-------------------------------------|-----------------|--------------------------------|-----------|----------------|---------------------------|-----------------------------|-------------------------|
| <input type="checkbox"/> | General purpose | t2.nano | 1 | 0.5 | EBS only | - | Low to Moderate |
| <input checked="" type="checkbox"/> | General purpose | t2.micro Free tier eligible | 1 | 1 | EBS only | - | Low to Moderate |
| <input type="checkbox"/> | General purpose | t2.small | 1 | 2 | EBS only | - | Low to Moderate |
| <input type="checkbox"/> | General purpose | t2.medium | 2 | 4 | EBS only | - | Low to Moderate |
| <input type="checkbox"/> | General purpose | t2.large | 2 | 8 | EBS only | - | Low to Moderate |
| <input type="checkbox"/> | General purpose | t2.xlarge | 4 | 16 | EBS only | - | Moderate |
| <input type="checkbox"/> | General purpose | t2.2xlarge | 8 | 32 | EBS only | - | Moderate |
| <input type="checkbox"/> | General purpose | m4.large | 2 | 8 | EBS only | Yes | Moderate |
| <input type="checkbox"/> | General purpose | m4.xlarge | 4 | 16 | EBS only | Yes | Moderate |

Cancel
Previous
Review and Launch
Next: Configure Instance Details

Selecting an instance type

After you have selected your instance type, click **Next: Configure Instance Details**. The following page is more complicated but you can ignore most of it for now, we may explore these options later.

Click **Next: Add Storage**. The default is 8GB of an SSD which is fine for us. Click **Next: Add Tags**. We don't need any tags but they are useful for when you have a large number of instances and you need to filter and search through them.

Click **Next: Configure Security Group**. A security group is a config for your server, telling it which ports it should expose to which IP addresses for certain types of traffic. Name the group something meaningful, I chose **tutorial group**.

To run our app we are going to need SSH access, which by default is on port 22 and uses the TCP protocol. Amazon adds this in for us by default. There is a warning about the source being `0.0.0.0/0` which allows us to SSH from any IP address however this is fine. If you were deploying and configuring your app from a VPN or company network then you could limit SSH access only for that IP which is a good security precaution when running something in production, but SSH access requires the correct SSH key so it is completely secure without this measure.

Since we would like to also serve an app we need to expose a HTTP port publicly, by default this is port 80 (but browsers strip this so you don't see it in URLs). Click **Add Rule** and select the type as **HTTP**, the default settings for this will use TCP as the protocol and expose port 80 to all IPs.

Services
Resource Groups

Robert Tod
Ireland
Support

1. Choose AMI
2. Choose Instance Type
3. Configure Instance
4. Add Storage
5. Add Tags
6. Configure Security Group
7. Review

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: ☒ Create a new security group
☐ Select an existing security group

Security group name:

Description:

| Type | Protocol | Port Range | Source |
|------|----------|------------|------------------|
| SSH | TCP | 22 | Custom 0.0.0.0/0 |
| HTTP | TCP | 80 | Custom 0.0.0.0/0 |

Add Rule

Warning
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

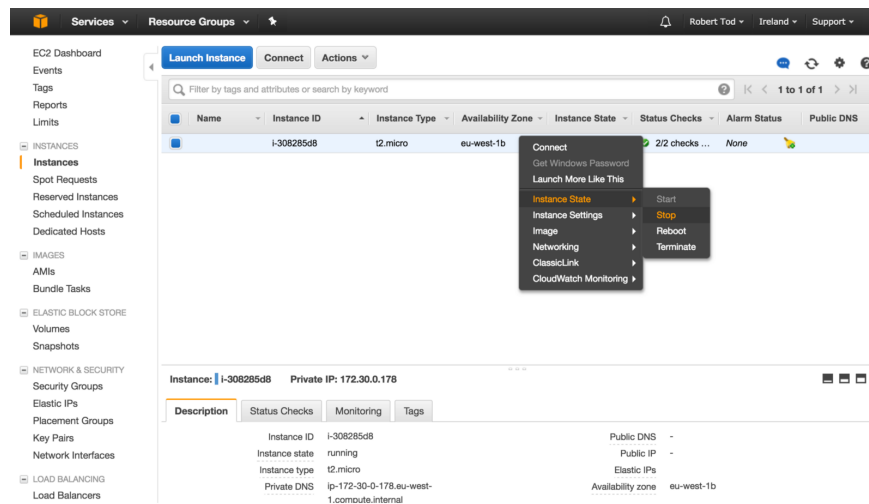
Cancel
Previous
Review and Launch

Configuring a security group

To launch the instance, click **Review and Launch**, then click **Launch**. You will be prompted to setup an SSH key which will give you access to the instance. Choose **create a new key pair**, and name the key, I named it “tutorial”. Click **Download Key Pair**. This should download a .pem file which can be used to SSH into the server. Keep this file safe because anyone can connect to your server using it, if you lose the file you will need to generate a new one.

Click **Launch Instance**. Click **View Instances**. Woo! Your instance should be booting up. Once the **Instance State** is **Running** then you are ready to SSH in.

Note: Remember to stop the instance when you aren’t running it! To do that just right click on the instance, under **Instance State** click **Stop**.



Stopping an instance

SSH into your server

Generally, the correct place to put your `.pem` file is in your `.ssh` folder, in your user directory. The `.ssh` folder is a hidden folder, to open it in finder open terminal and execute the `open` command.

```
# The open command will open the
# given path using the default system
# application for the file type. For
# folders it uses finder!
$ open ~/.ssh
```

Once the `.pem` file is in your `ssh` folder, use `chmod` to set permissions so that it can be used as a key.

```
$ chmod 400 ~/.ssh/whatever-your-key-name-is.pem
```

To SSH we need to have a username, an address and a key. The address is available when we click on our instance in the EC2 instances dashboard.

The screenshot shows the AWS Management Console interface. On the left, there's a navigation menu with categories like INSTANCES, IMAGES, ELASTIC BLOCK STORE, and NETWORK & SECURITY. The main area displays a table of EC2 instances. One instance, 'i-c9b3b421', is highlighted. Below the table, the 'Instance details' section shows various attributes for this instance, including its ID, state (running), type (t2.micro), and public IP address (52.214.64.31).

| Name | Instance ID | Instance Type | Availability Zone | Instance State | Status Checks | Alarm Status | Public DNS |
|------|-------------|---------------|-------------------|----------------|----------------|--------------|--|
| | i-c9b3b421 | t2.micro | eu-west-1b | running | 2/2 checks ... | None | ec2-52-214-64-31.eu-west-1.compute.amazonaws.com |

| Instance: i-c9b3b421 | | Public DNS: ec2-52-214-64-31.eu-west-1.compute.amazonaws.com | |
|----------------------|---|--|--|
| Instance ID | i-c9b3b421 | Public DNS | ec2-52-214-64-31.eu-west-1.compute.amazonaws.com |
| Instance state | running | Public IP | 52.214.64.31 |
| Instance type | t2.micro | Elastic IPs | |
| Private DNS | ip-172-30-0-84.eu-west-1.compute.internal | Availability zone | eu-west-1b |

The instance I setup has a public IP `52.214.64.31` which I could use to connect, giving the `.pem` file as a key using the `-i` flag.

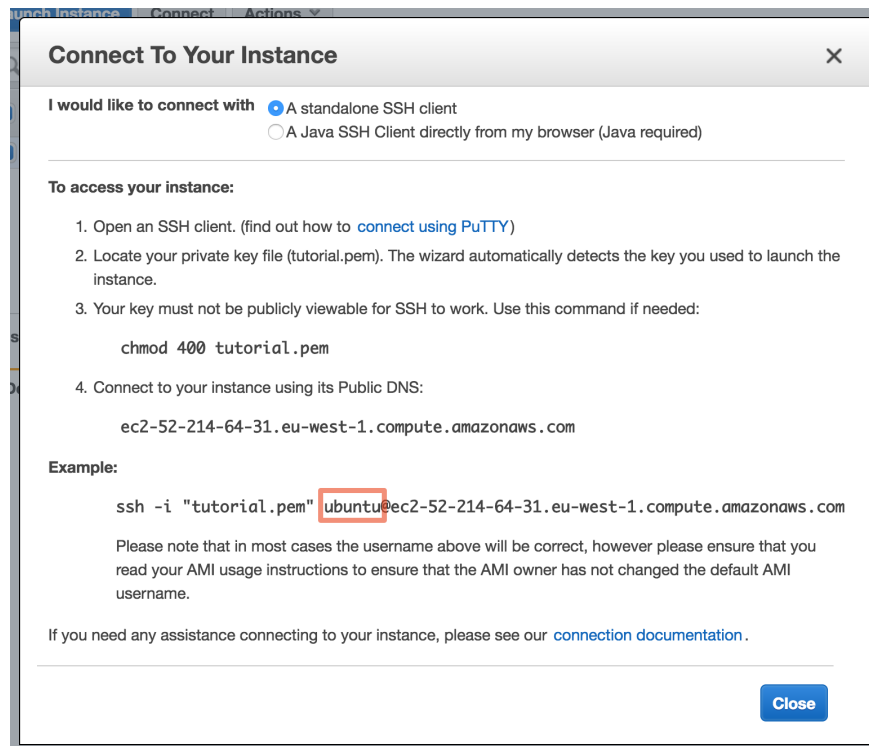
```
# Don't actually run this command
$ ssh -i ~/.ssh/whatever-your-key-name-is.pem 52.214.64.3
```

However it is more correct to use the public DNS (the only real difference though is when you are connecting from EC2 machine to EC2 machine then the DNS will resolve to the private IP rather than the public IP).

```
# Don't run this command either
$ ssh -i ~/.ssh/whatever-your-key-name-is.pem ec2-52-214-64-31.eu-west-1.compute.amazonaws.com
```

Almost there. By default, connecting to your instance without a username will try to login as `root` which is generally not allowed. By right clicking your instance and selecting **connect** we can see what the

correct username is. This is dependant on the image you chose, for ubuntu the username is by default `ubuntu`. You can see the username in the *Example* section of the dialogue, it is the part before the `@` symbol.



Finding the SSH username

To connect, the SSH command should look something like

```
# Fill in your specific details for this to work.  
$ ssh -i ~/.ssh/whatever-your-key-name-is.pem ubuntu@ec2-52-214-64-31.eu-west-1.compute.amazonaws.com
```

This should connect you to your instance, just type `yes` when prompted so that you can add your instance as a known host. You're connected!

Installing node and system dependencies

Once in an SSH session the first thing to do is get Node.js. NVM (Node Version Manager) is a pretty great way to install Node.js and allows you

to easily switch versions if required.

To install NVM just run this command (same as in the NVM installation instructions).

```
$ curl -o-  
https://raw.githubusercontent.com/creationix/nvm/v0.32.1/install.sh | bash
```

This command pulls down a script from a remote URL and runs it. You now have NVM! But if you run `nvm ls` you will notice it isn't found. This is because NVM adds some code to your `~/.bashrc`. This file is a special file that is run every time you log in to your instance, so to get NVM running you could logout and login again. However you can just run the file manually by using the `source` command.

```
$ source ~/.bashrc
```

Now running `nvm ls` works! But there aren't any node versions installed! To get the latest version, just `nvm install <latest version number>`.

```
$ nvm install 7
```

To check node is ready to go just echo the version.

```
$ node --version
```

```
ubuntu@ip-172-30-0-182:~$ nvm i 7
##### 100.0%
Computing checksum with sha256sum
Checksums matched!
Now using node v7.2.1 (npm v3.10.10)
Creating default alias: default -> 7 (-> v7.2.1)
ubuntu@ip-172-30-0-182:~$ node --version
v7.2.1
ubuntu@ip-172-30-0-182:~$
```

Installing node

Node.js is installed!

Creating a public HTTP endpoint

Before concluding this part of the tutorial we are going to make a public URL anyone can request from the browser to get a response from the server.

Make a directory for the server and `cd` into it.

```
$ mkdir server
$ cd server
```

Now you are in your server directory, you need to `npm init`

```
$ npm init
```

This will create a `package.json` file which will be used to track any dependencies we use. `npm init` will ask for a load of info, I tend to just press `enter` to use all the defaults.

All we will need to run our server is the `express` package. To install `express` and add it to `package.json`.

```
$ npm install express --save-dev
```

Note that now you should have a `node_modules` directory and `package.json`

```
$ ls
```

```
[ubuntu@ip-172-30-0-182:~/server$ ls  
node_modules package.json
```

Now we just need to add some code to run the server. We will use nano to write the server in an `index.js` file.

```
$ nano index.js
```

The server I used just responds with “HEY!” when you do a request to `/`. I listen to requests on port 3000.

```
const express = require('express')  
const app = express()  
  
app.get('/', (req, res) => {  
  res.send('HEY!')  
})  
  
app.listen(3000, () => console.log('Server running on port 3000'))
```

Press `ctrl+x` to exit, ensuring you save when you exit by pressing `y` followed by `enter`.

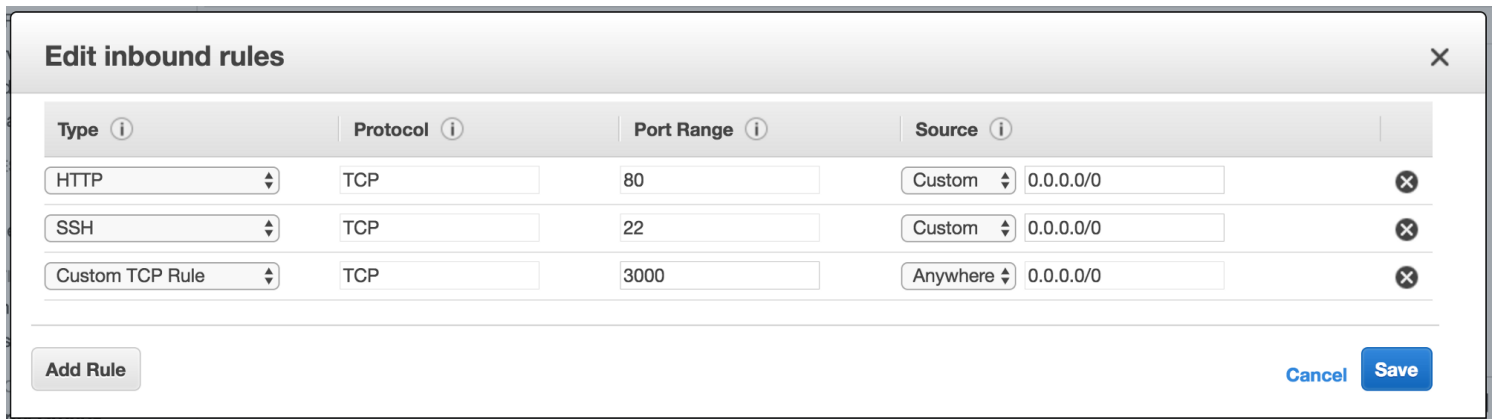
Now we can use node to start the server!

```
$ node index.js
```

Once listening, this should log “Server running on port 3000”. You may have noticed however we didn’t open our server traffic to port 3000 , we opened it to port 80 . Port 80 is a privileged port and running the server there using Node.js is unusual, generally using a router is better. If you change the index.js file to use 80 and then run node index.js you will notice you get a permission denied error.

In the next tutorial we will go through adding a router to use port 80 , but for now let’s just open up port 3000 so we can test our server.

Leave your server running and go to the **Security Groups** tab in the EC2 console. Right click the security group you setup and click **edit inbound rules**. Click **Add Rule**. This time we are going to use a custom TCP rule on port 3000, open to anywhere.



| Type | Protocol | Port Range | Source |
|-----------------|----------|------------|--------------------|
| HTTP | TCP | 80 | Custom 0.0.0.0/0 |
| SSH | TCP | 22 | Custom 0.0.0.0/0 |
| Custom TCP Rule | TCP | 3000 | Anywhere 0.0.0.0/0 |

Add Rule **Cancel** **Save**

Opening up port 3000 to TCP traffic

Click Save. We should now have access to our server! Using a browser, visit your public DNS URL with port 3000 and you should see the HEY! response.



The response from your server

To leave the server running when we log out, we need to press `ctrl+z` to pause the process (this only works when your server is running, `node index.js`). When you press `ctrl+z` you will be presented with all jobs, in this case the only one there will be the Node.js job that was paused.

```
[ubuntu@ip-172-30-0-182:~/server$ node index.js
Server running on port 3000
^Z
[1]+  Stopped                  node index.js
ubuntu@ip-172-30-0-182:~/server$ |
```

You can see that the job number for `node index.js` is 1 (as noted by `[1]+`). To run that in the background, use the `bg` command.

```
$ bg %1
```

Then logout

```
$ exit
```

You should still be able to access your URL and see the “HEY!” response.

Conclusion

If everything went well we should have a simple Node.js app which is accessible from a public URL anywhere in the world!

To stop the instance just navigate to the **Instances** tab in the EC2 dashboard, right click on your instance and in **Instance State** click **Stop**.

In the next tutorial I intend to cover

- running the app on port 80 using nginx
- using PM2 to keep the app running after a restart
- using PM2 to deploy the app from a local directory

Let me know in the comments section if there is anything I missed or is a bit confusing!

Part 2 is now available.



like!



tweet!



about!

Hacker Noon is how hackers start their afternoons. We're a part of the @AMI family. We are now accepting submissions and happy to discuss advertising & sponsorship opportunities.

If you enjoyed this story, we recommend reading our latest tech stories and trending tech stories. Until next time, don't take the realities of the world for granted!



