Deploying a Web Service in the Cloud

Luigi Libero Lucio Starace

luigiliberolucio.starace@unina.it

Università degli Studi di Napoli Federico II, Naples, Italy

November 19, 2021

Goals of this demonstration

- Download and run a simple Node.js web service locally
  - Requirements: Node.js (recommended v14.16.1 or above)
- Deploy the web service in the Cloud using AWS (on an AWS EC2 instance)

Download the web service

Clone the repo: `https://github.com/luistar/cloud-ws-demo.git`

```
>> git clone https://github.com/luistar/cloud-ws-demo.git
Cloning into 'cloud-ws-demo'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 11 (delta 2), reused 7 (delta 1), pack-reused 0
Unpacking objects: 100% (11/11), done.
```

Then move to the newly-created directory

```
>> cd cloud-ws-demo
```

Check out the app.js file (if you want)

Two get endpoints: `/greet` and `/fortune`

```
const express = require('express')
const app = express()
const port = 3000

app.get('/greet', (req, res) => {
  res.send('Hello World!')
})

app.get('/fortune', (req, res) => {
  // [...] computes a random fortune message (omitted for brevity)
  res.send(fortune)
})
```

Install dependencies and run the web service

Install dependecies

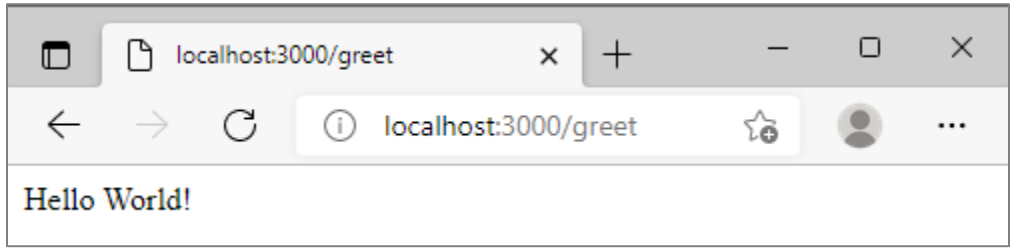
```
>> npm install
```

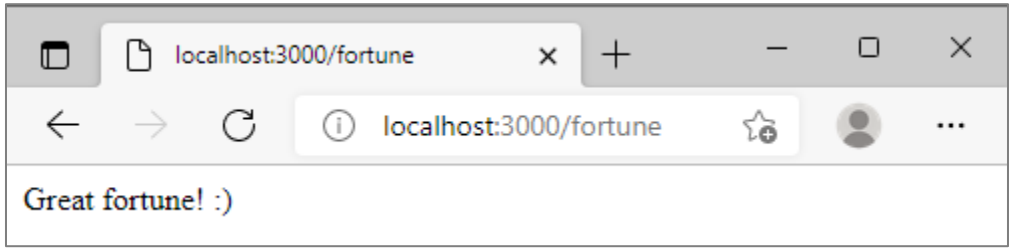
Run the web service

```
>> node app.js

Web service listening at http://localhost:3000
```

And try it with your web browser!





Let’s move to the Cloud

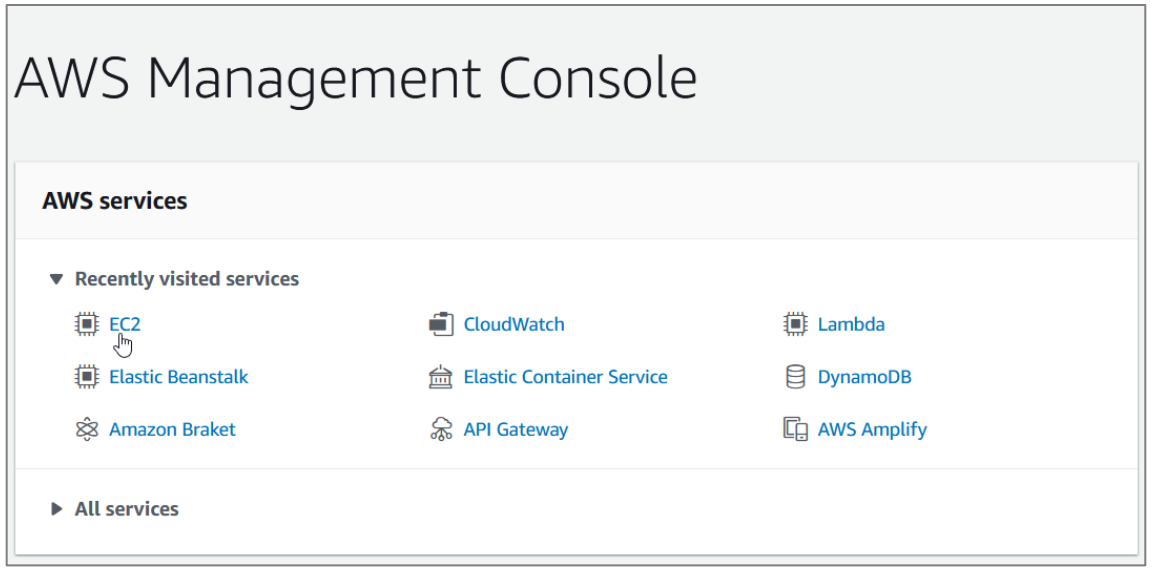
We’ll use AWS for this tutorial

Requirements

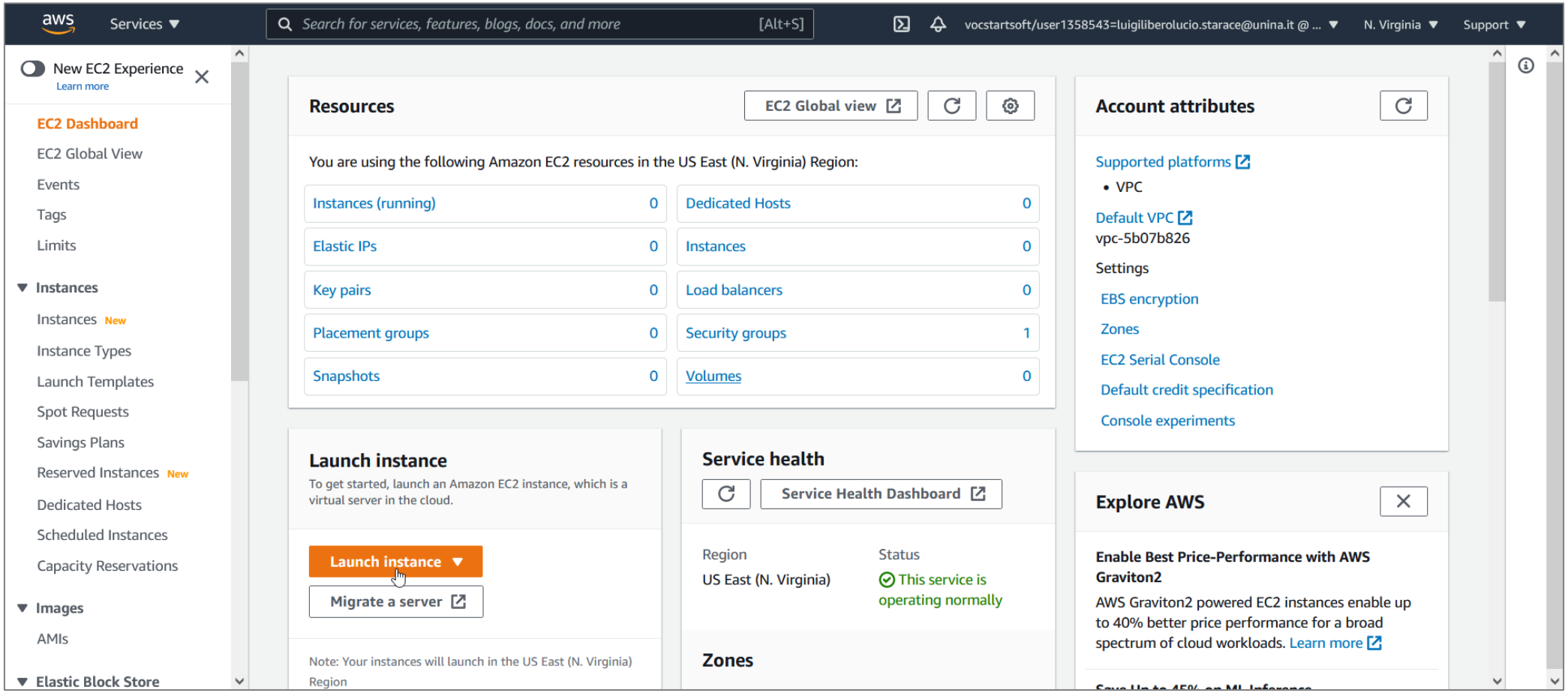
- To replicate this demo you'll need an AWS account. If you're eligible for the free trial, it'll cost you nothing!
- If you prefer a different provider (such as Azure) the procedure is basically the same (so a little of google-fu will surely suffice!)

EC2 deployment

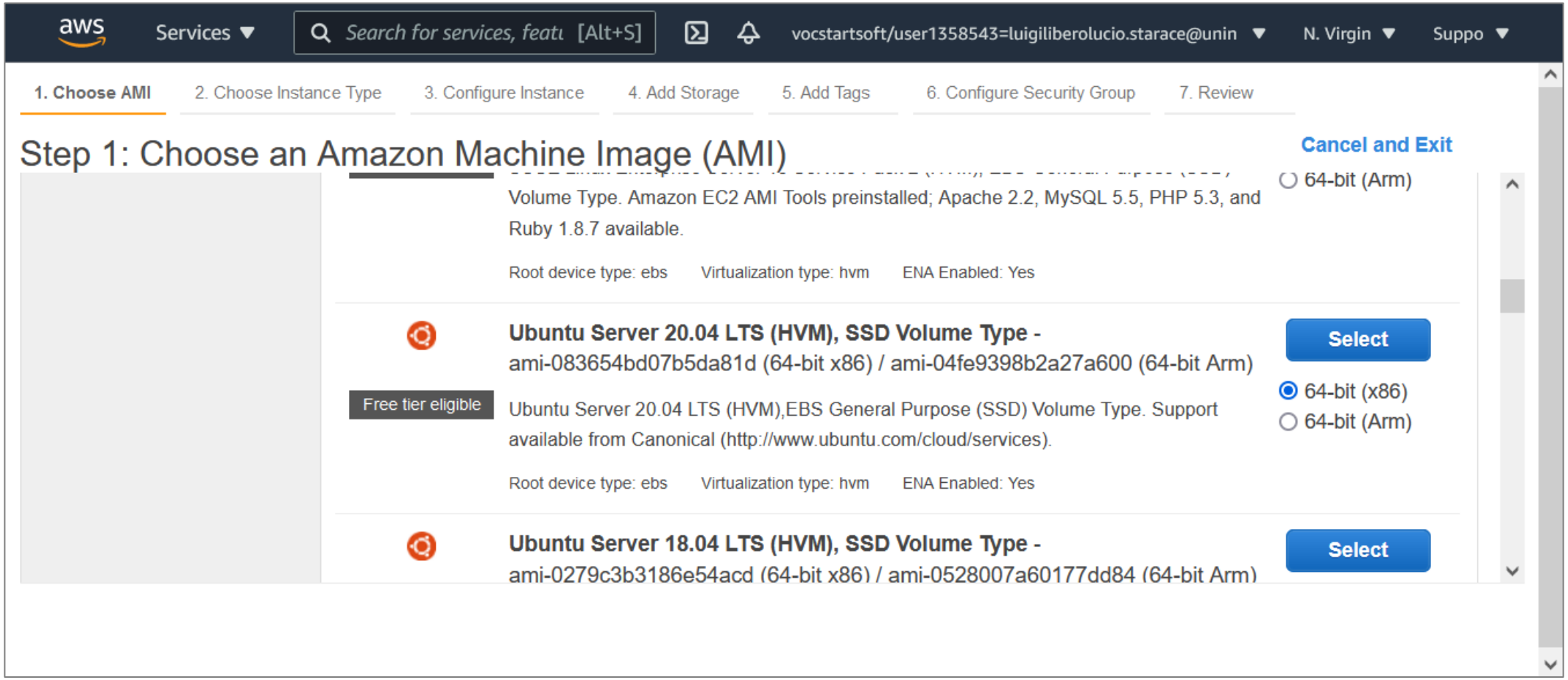
- We'll start from the most basic deployment: running the web service on a virtual server in the Cloud.
- Let's start by selecting the EC2 service from the AWS web console



Click on the «Launch instance» button



Select the Ubuntu Server 20.04 LTS Machine Image



Select the t2.micro instance type

aws Services ▼ Search for services, features, blogs, docs, and more [Alt+S] vocstartsoft/user1358543=luigiliberolucio.starace@unina.it @ ... ▼ N. Virginia ▼ 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 families | Current generation | Show/Hide Columns

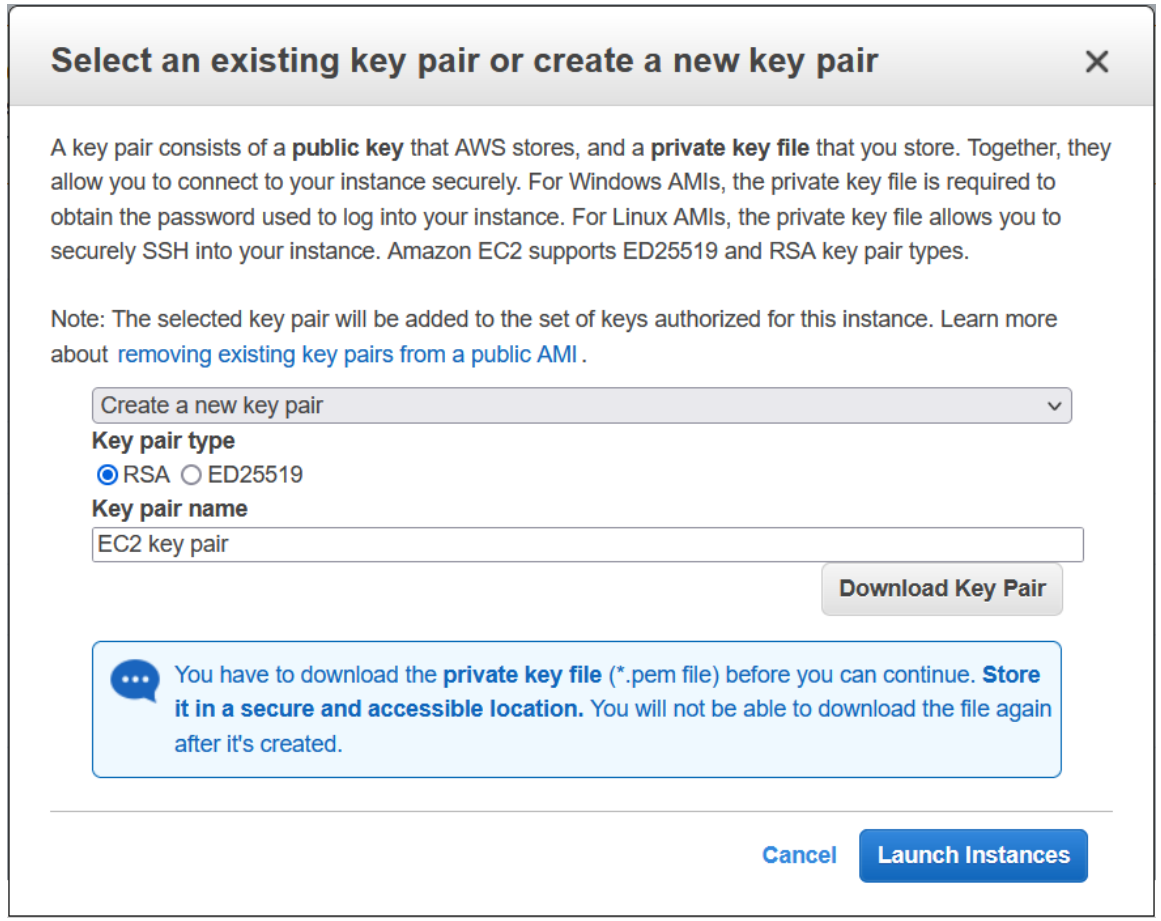
Currently selected: t2.micro (- ECUs, 1 vCPUs, 2.5 GHz, -, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
☐	t2	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
☑	t2	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes
☐	t2	t2.small	1	2	EBS only	-	Low to Moderate	Yes
☐	t2	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
☐	t2	t2.large	2	8	EBS only	-	Low to Moderate	Yes
☐	t2	t2.xlarge	4	16	EBS only	-	Moderate	Yes
☐	t2	t2.2xlarge	8	32	EBS only	-	Moderate	Yes

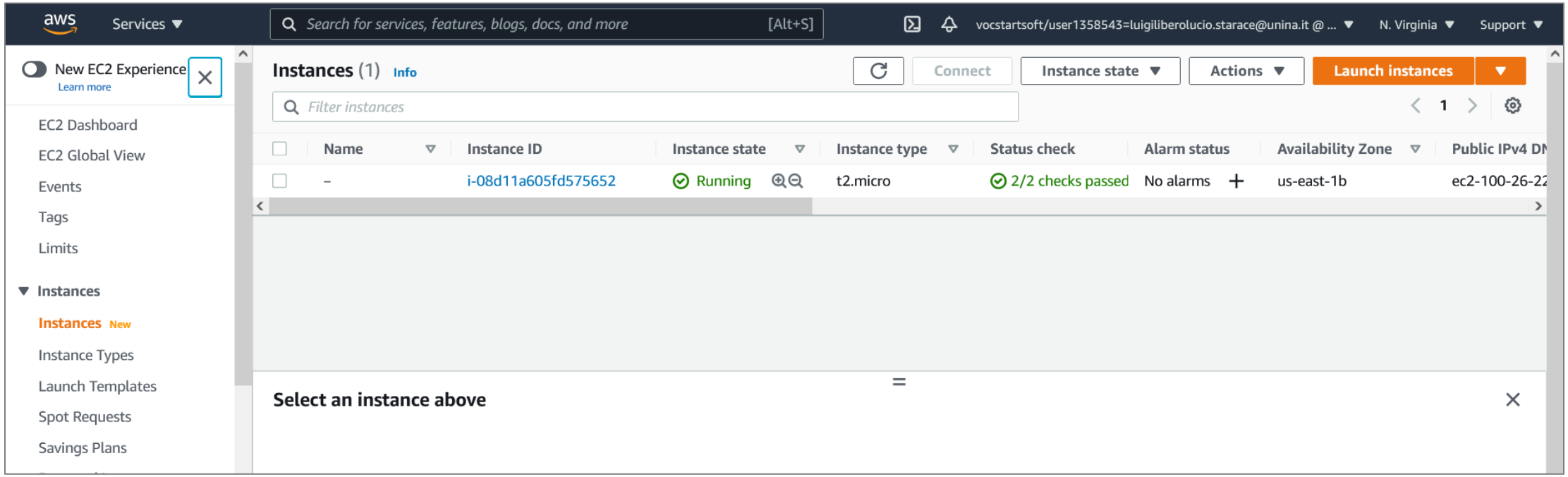
Cancel | Previous | Review and Launch | Next: Configure Instance Details

Generate a new key pair to connect to the instance

- Make sure to download the .pem file

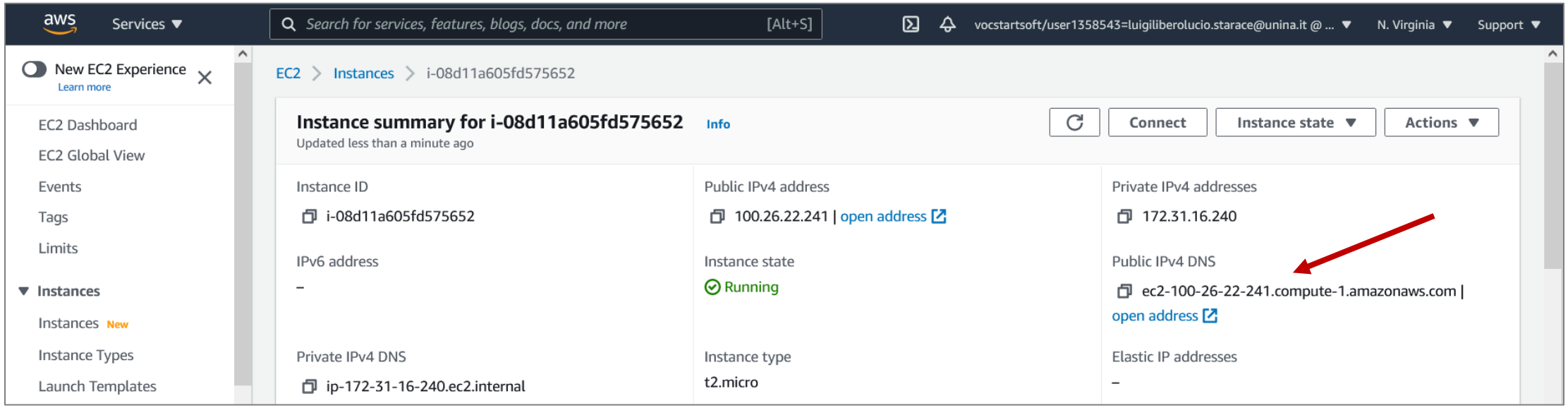


Wait for the instance to start



Click on the instance id to know more

Note the public IPv4 DNS name!



SSH into your new (virtual) server

SSH into your instance

```
>> ssh -i /path/to/key.pem ubuntu@<the-ipv4-dns-name-we-noted>
```

If everything worked, you should be in!

```
ubuntu@ip-172-31-16-240:~$
```

Notice: you might be getting an «**unprotected private key file**» error. In that case, make sure to correctly assign permissions to the pem file, so that only you can see it. Use chmod if on a unix derivative, or see this if you're on Windows.

Install Node.js on your virtual server

Update and then install nodejs and npm

```
ubuntu@ip-172-31-16-240:~$ sudo apt update

ubuntu@ip-172-31-16-240:~$ sudo apt install nodejs npm
```

Then clone the web service

Clone and install dependencies

```
ubuntu@ip-172-31-16-240:~$ git clone https://github.com/luistar/cloud-ws-demo.git

ubuntu@ip-172-31-16-240:~$ cd cloud-ws-demo

ubuntu@ip-172-31-16-240:~/cloud-ws-demo$ npm install
```

And then you're ready to start our Web Service

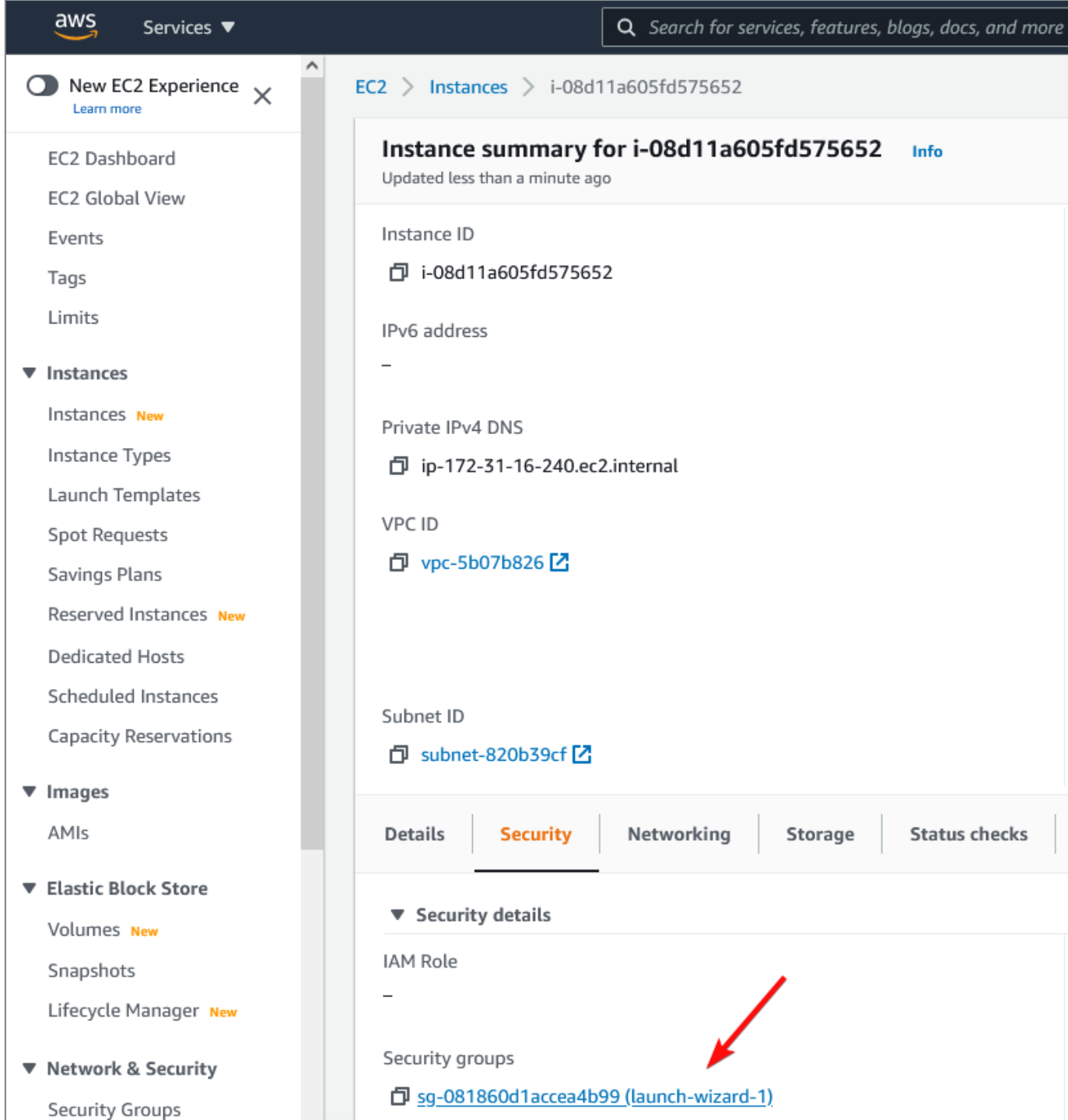
```
ubuntu@ip-172-31-16-240:~/cloud-ws-demo$ node app.js
Web service listening at http://localhost:3000
```

We’re almost there!

Our web service is up and running in our virtual server, but we still cannot reach it from the internet!

Allow incoming traffic

- Select the security group in the security tab in your instance summary page

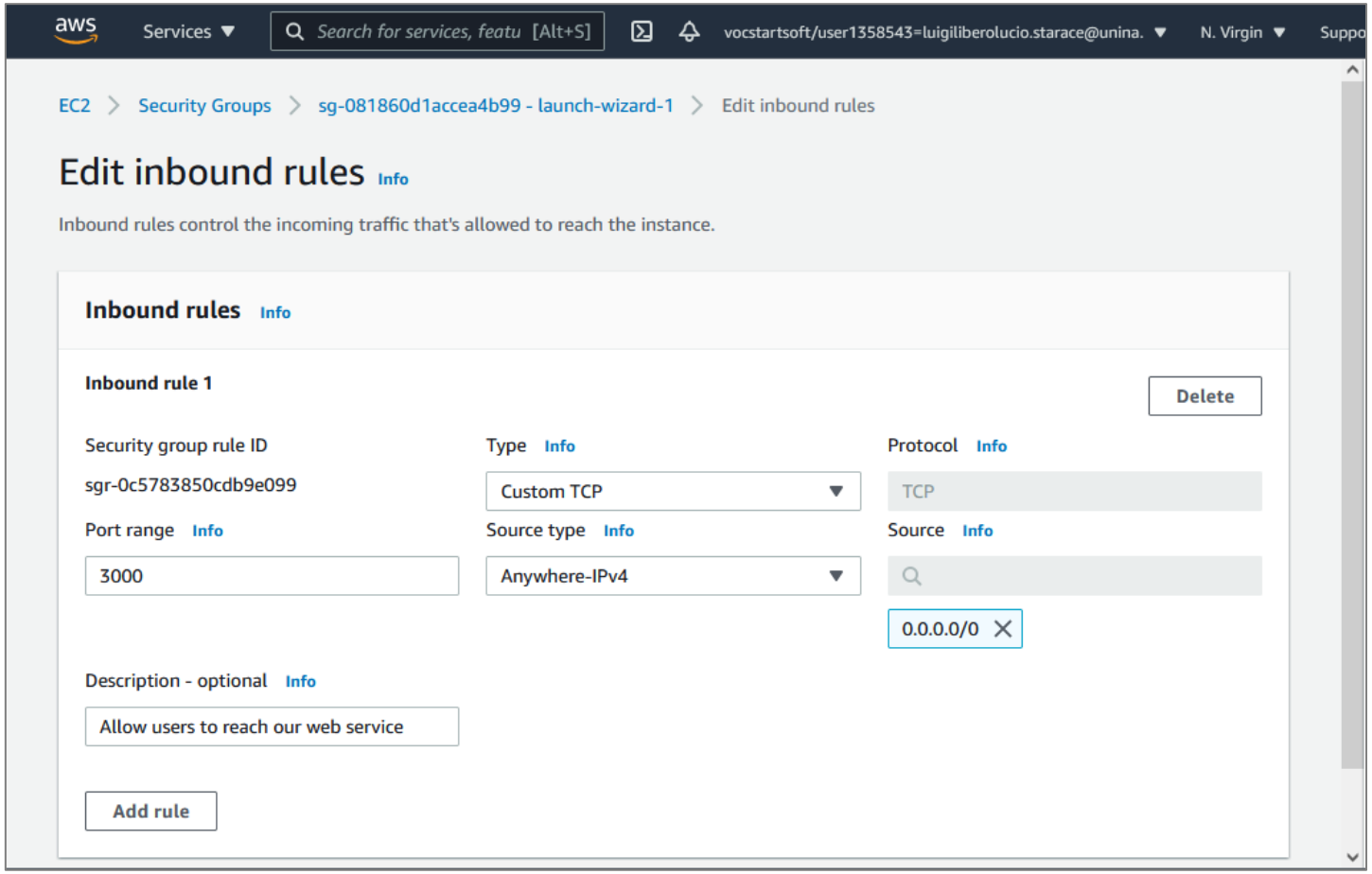


Add a new inbound traffic rule

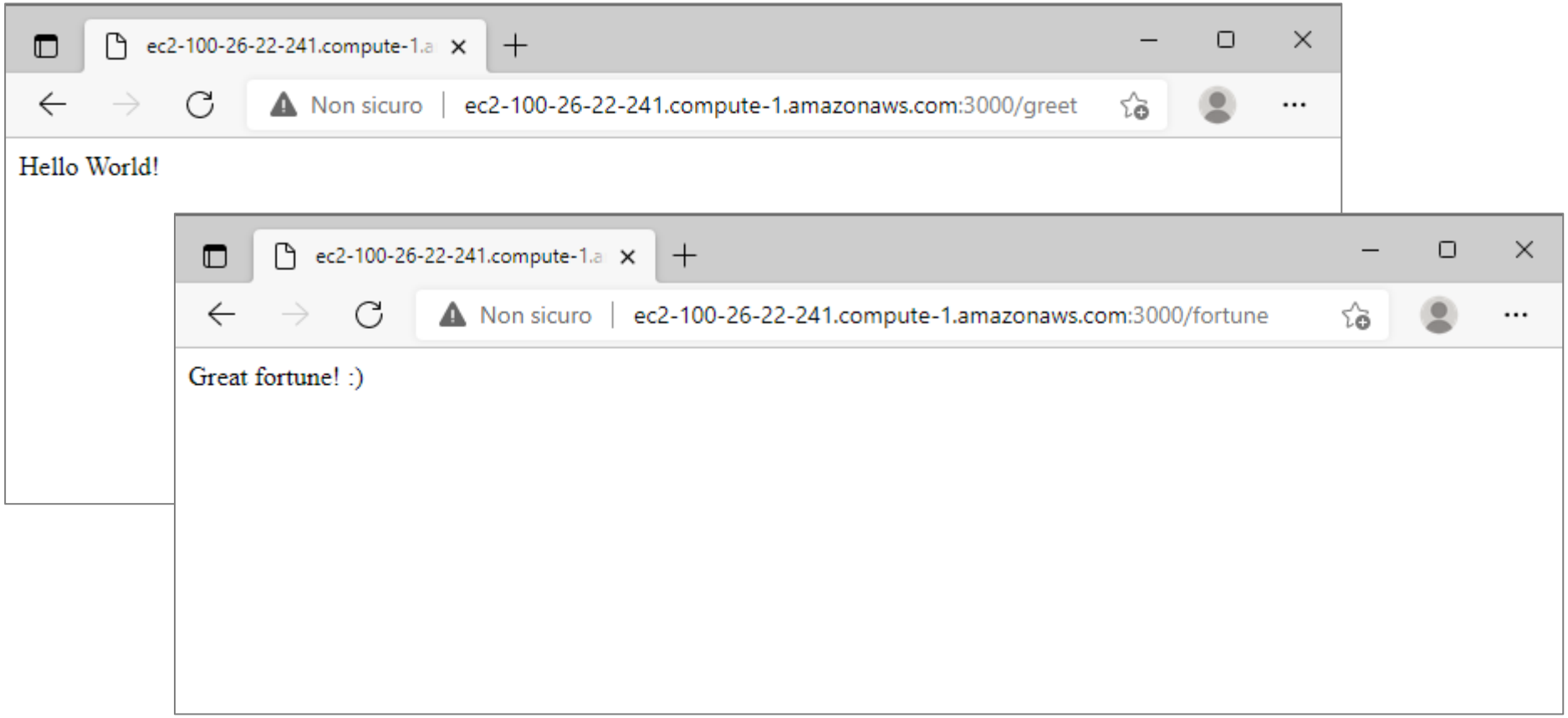
Type: Custom TCP

Port range: 3000

Source: Anywhere IPv4



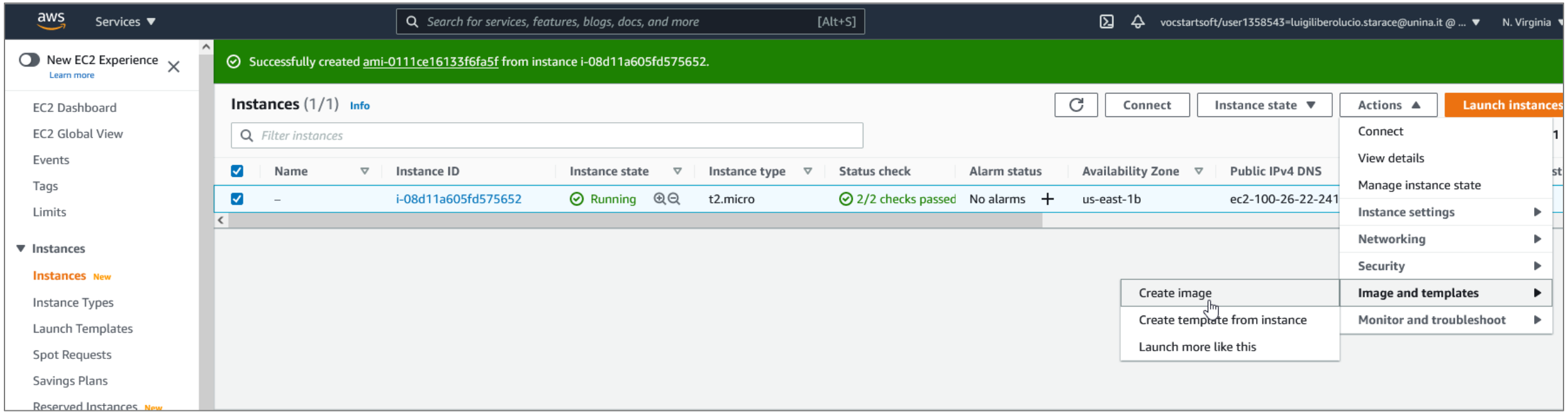
Guess what? Now it works!



Creating an image from our instance

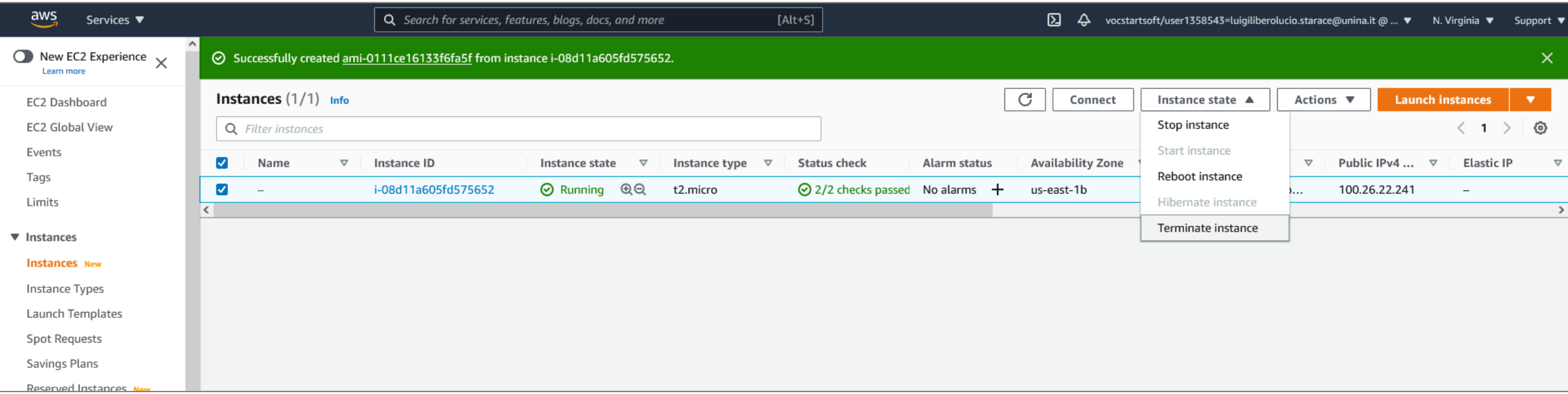
Updating the repositories, installing Node.js and npm, downloading and installing our web service was **fun**, sure.

Can we skip all that next time? Yes, we can create an image!



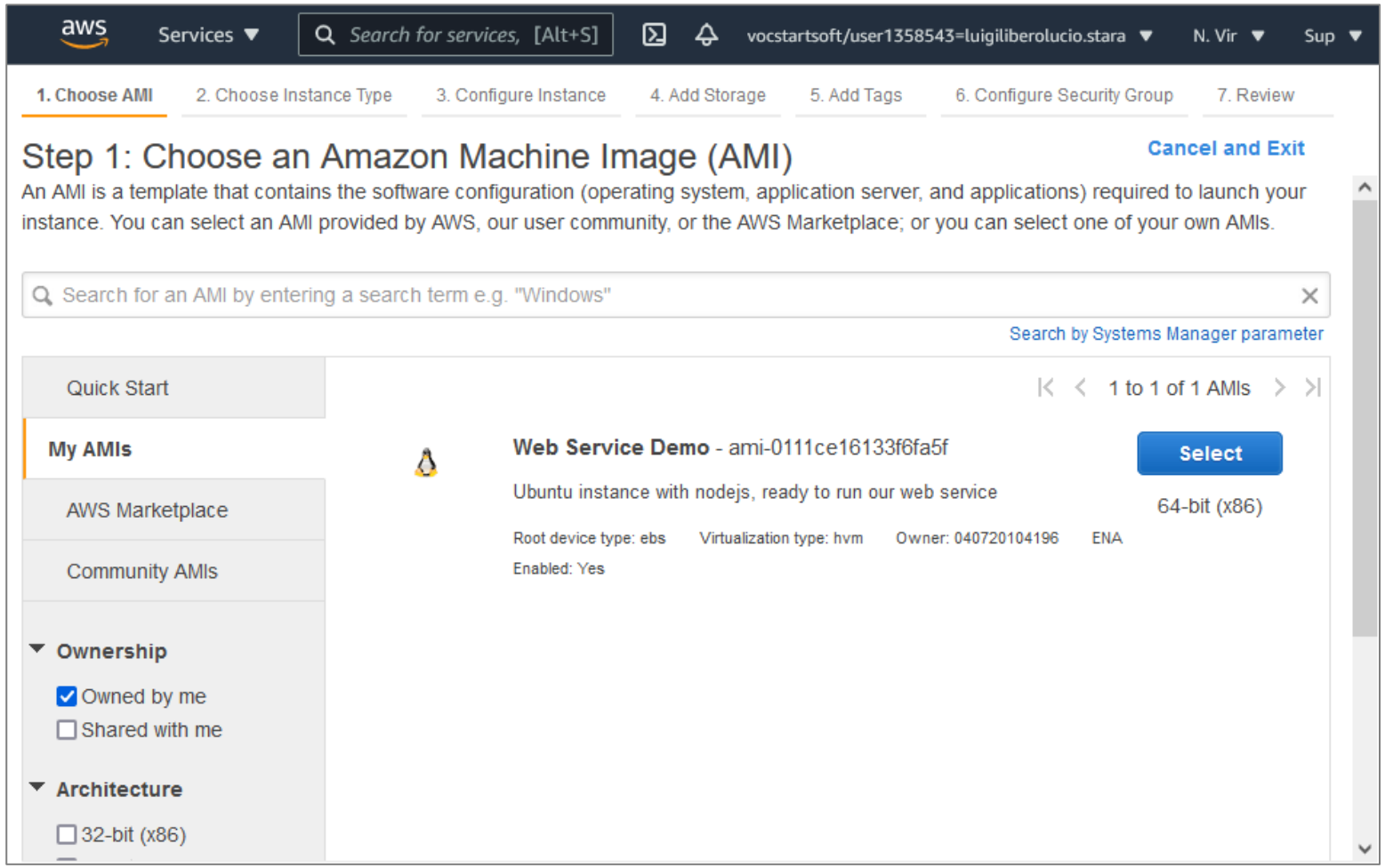
Terminate our instance

- Remember that we're paying as long as that instance is up! Terminate it when it's not needed anymore!



Start the instance from the image we saved

This time we can select our image!



Running the image we saved

- This time, when starting the EC2 instance, make sure to use the same security group we already set up! (otherwise, you will need to add the inbound rule to port 3000 again!)

Suggested next steps

i. Check out **AWS Elastic IP Address** if you need a static IP that does not change when you stop and restart an instance
ii. Try adding an **auto-scaling group**!
iii. After doing (i) and (ii), try deploying our web service using **AWS ElasticBeanstalk!**