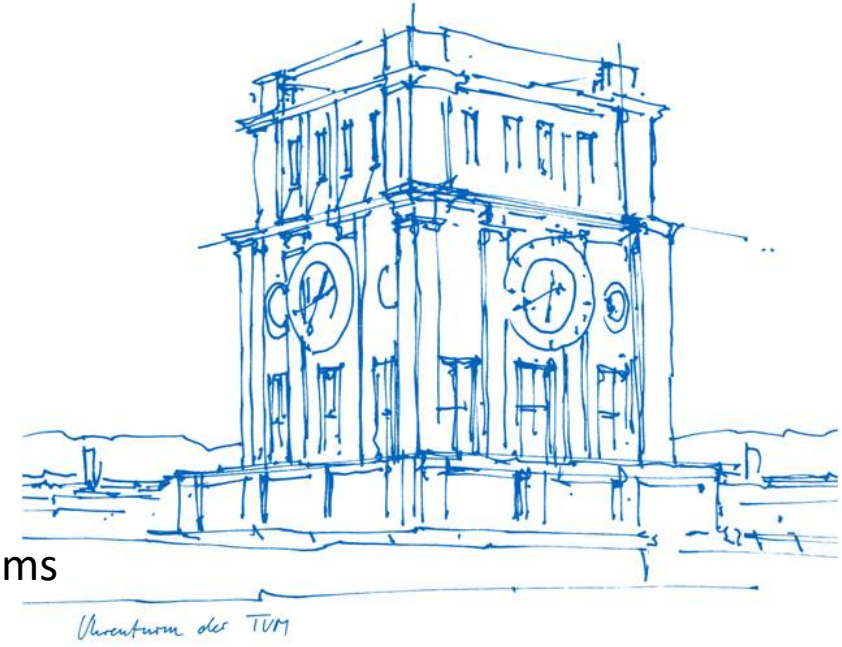
Cloud Computing Exercise – 1

Application Development using Node.js



Anshul Jindal (M.Sc. Informatics)

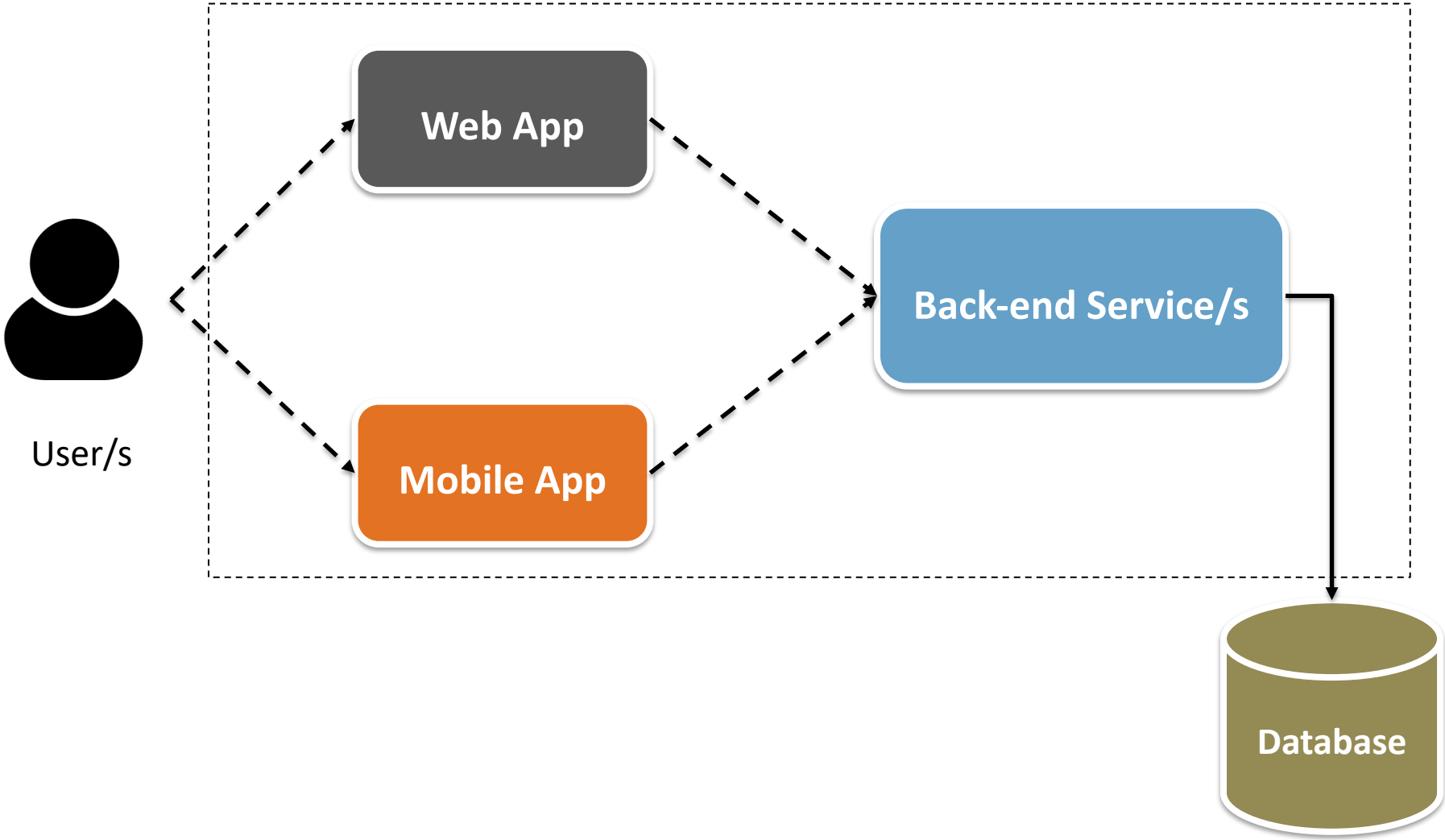
anshul.jindal@tum.de

Chair of Computer Architecture and Parallel Systems

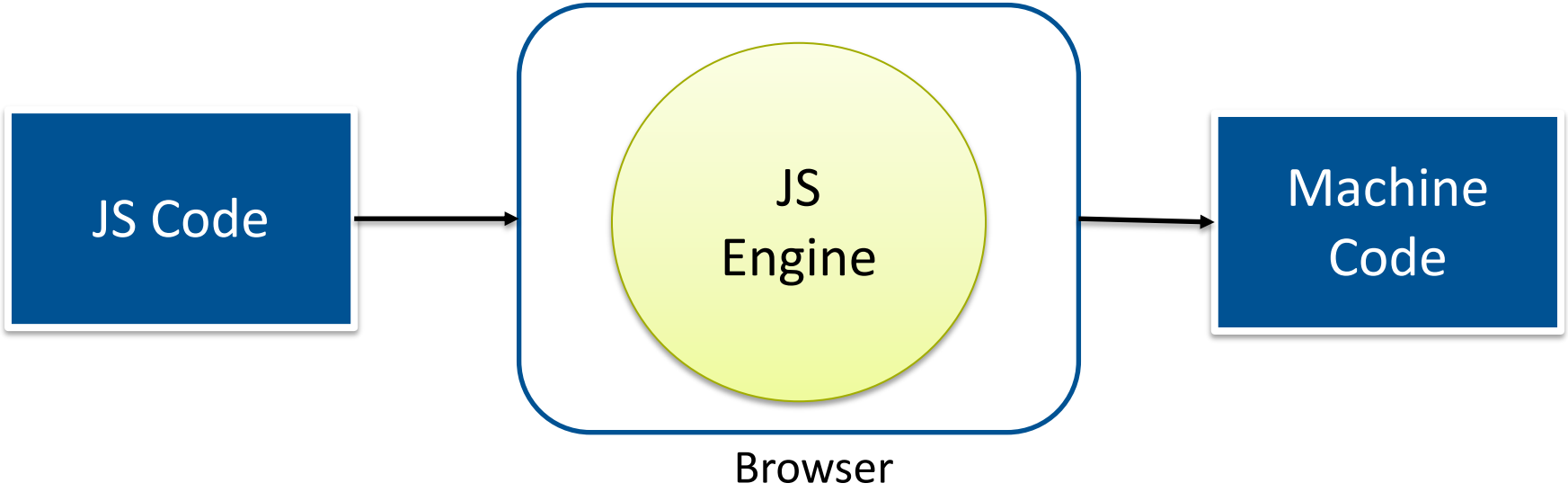
Technical University of Munich (TUM), Germany

Introduction to Node.js

Example of a Request to an Application Flow



How JavaScript (JS) is running on a browser ?



JS Engine in Browsers

Microsoft edge
has Chakra

Firefox
has Spider Monkey

Chrome
has V8

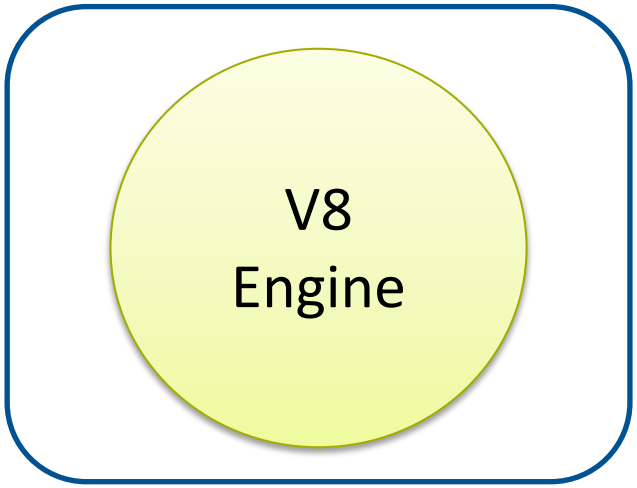
Node

- Browser provides Runtime Environment for JS code.
- Till 2009 only way to run JS code was inside a browser.

- Ryan Dahl inventor of Node.js in 2009 [1]

Chrome
has V8

Took V8 Engine

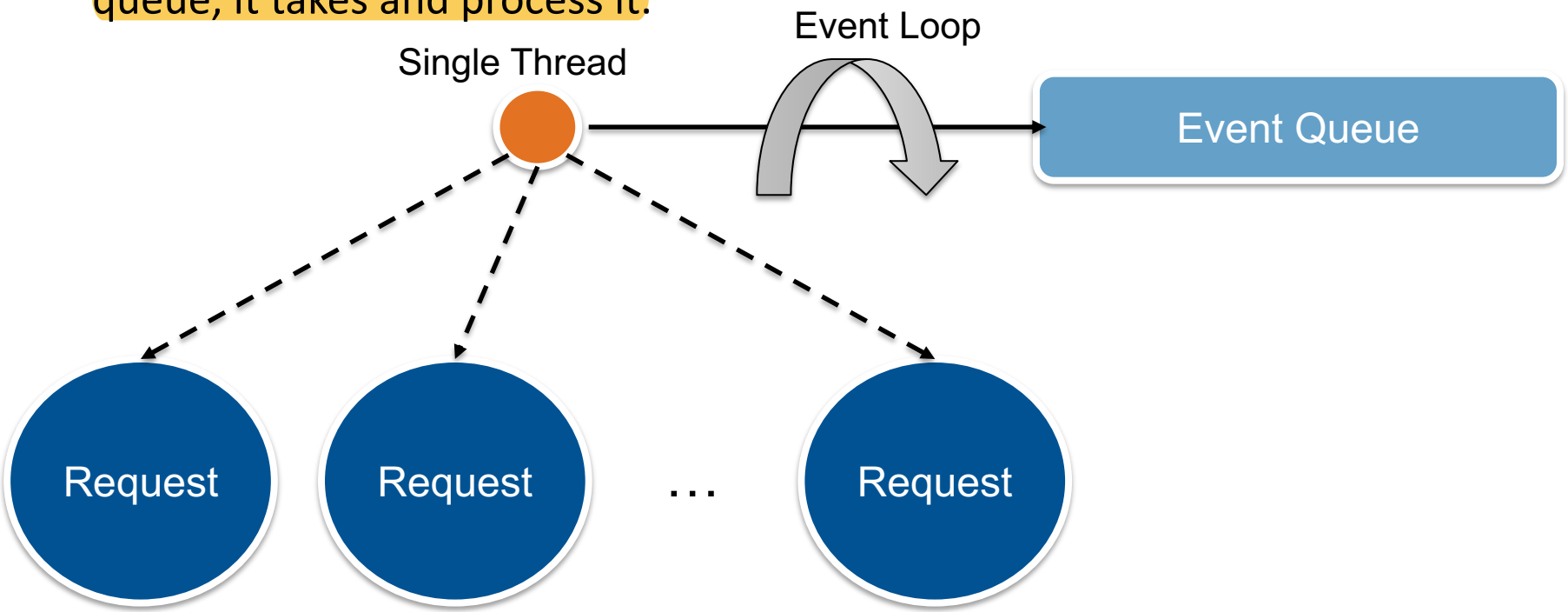


Embedded V8 inside a
C++ program called as
Node.exe

- Similar to browser Node is a Runtime Environment for JS code.
- ".js" was added to Node to just name the product.

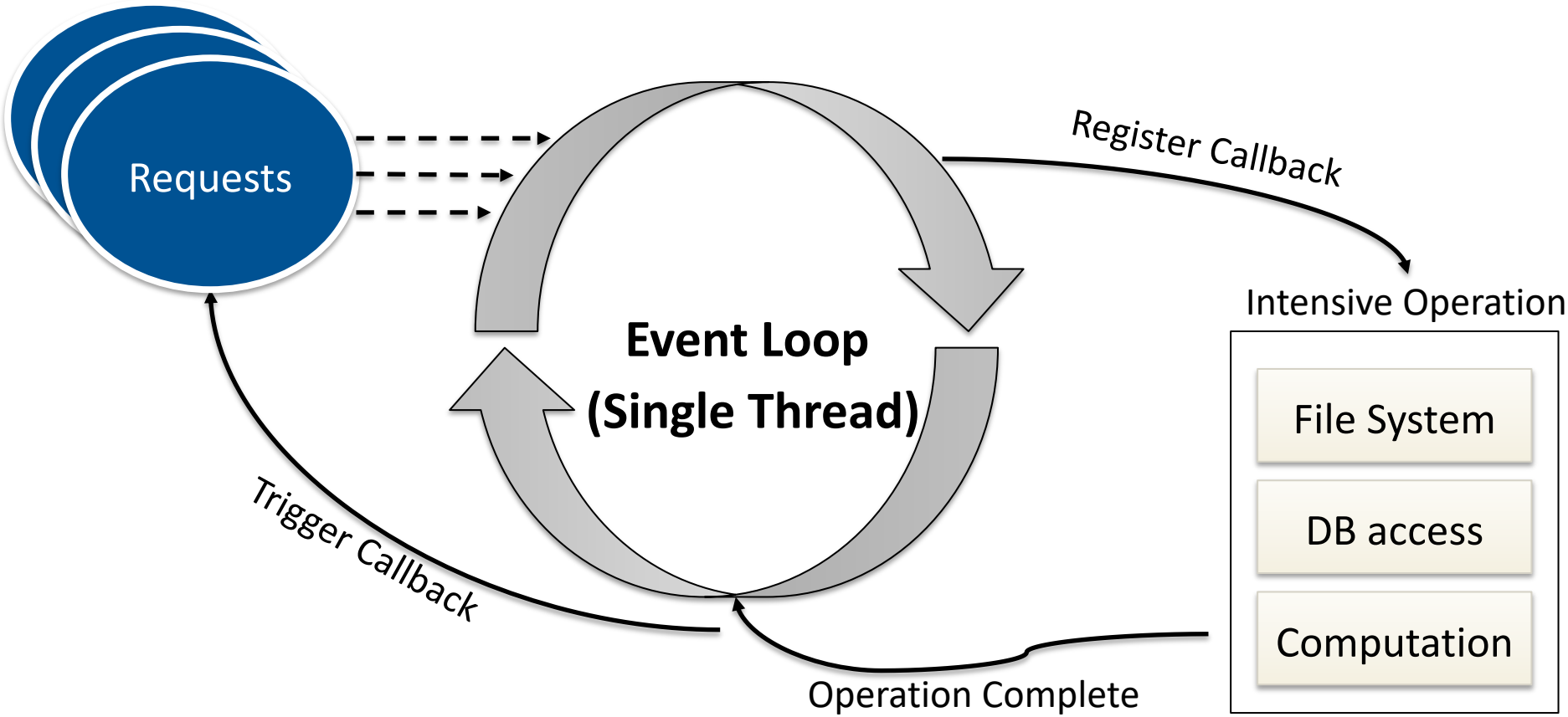
Node.js Non-Blocking/Asynchronous Nature

- A single thread is used to handle multiple requests.
- If a request needs to query a Database, thread does not wait for the database to return data.
- It moves to the next request for handling.
- When the database result is ready, its added to event queue.
- Node monitors this queue continuously and whenever it finds an event in this queue, it takes and process it.



Node.js Event Loop

The event loop simply iterate over the event queue which is basically a list of events and callbacks of completed operations.



Node.js

“A platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. [2]”

- Core in c++ on top of Chrome v8 engine.
- Can handle thousands of Concurrent connections with Minimal overhead (cpu/memory) on a single process.
- It’s NOT a web framework, and it’s also NOT a language.
- It is a Runtime Environment for JS code.
- Works on single thread using non-blocking I/O calls

When and when not to use Node.js

When to use ?

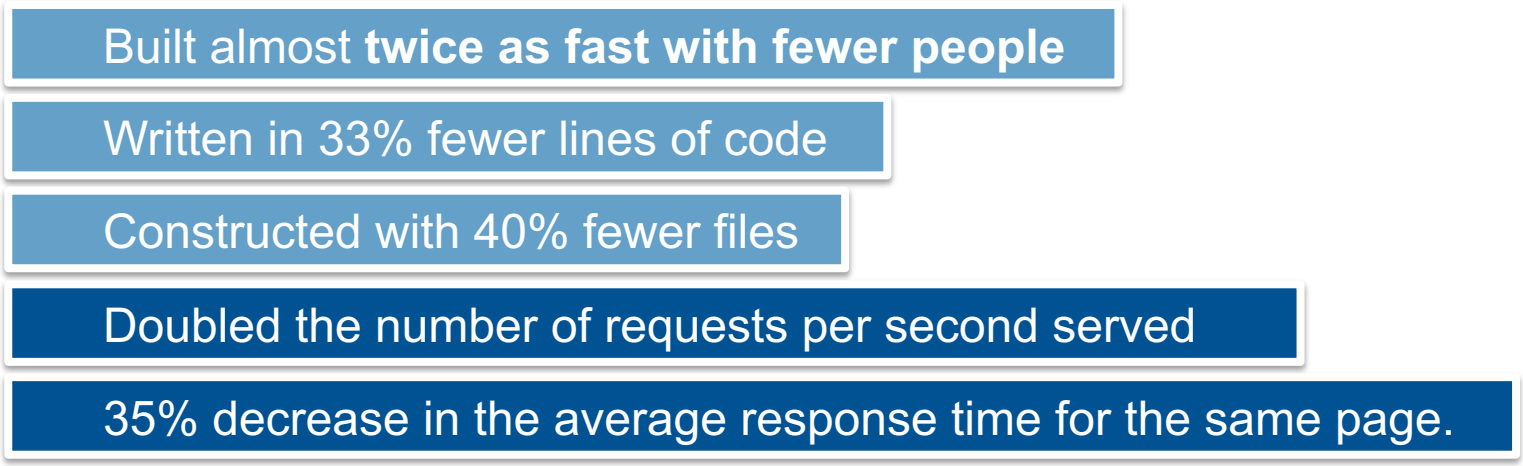
- **Real time applications :** applications that have to process a high volume of short messages requiring low latency.
- **Fast and scalable environment:**
  - ability to process many requests with low response times.
  - makes it a great fit for modern web applications that carry out lots of processing on the client's side. Ex. single-page applications.

When not to use ?

- **CPU-heavy jobs:**
  - Node.js is based on an event-driven, non-blocking I/O model, and uses only a single CPU core.
  - CPU-heavy operations will just block incoming requests, rendering the biggest advantage of Node.js useless.
- **Simple Web Applications:** Using Node.js would be superfluous for simple HTML applications in which you don't need a separate API, and all data comes directly from the server.

PayPal Case Study [3]

- PayPal, a worldwide online payments system, were using the Java on server side and JavaScript on the browser side.
- This gave them lots of problems and eventually they moved their backend development from Java to JavaScript and Node.js.
- They found that, the Node.js app was:



Node Package Manager (NPM) [4]

- Used to install node programs/modules
- Easy to specify and link dependencies
- Modules get installed inside "**node_modules**" folder

```
npm install express          // install express module
npm install express --save   // install express module and add in package.json
npm install -g express       // install express globally
```

- Popular Modules
  - **Express** – Web development framework
  - **Connect** – Extensible HTTP server framework
  - **Socket.io** – Server side component for WebSocket
  - **Mongo/Mongoose** – Wrappers to interact with mongo database

package.json

- Present inside the root folder of your package/application
- Tells npm how the package is structured and what all dependencies need to be installed.

```
{
  "name": "clientapp",
  "main": "clientServer.js",
  "repository": {
    "type": "git",
    "url": ""
  },
  "scripts": {
    "start": "node clientServer.js"
  },
  "author": "Anshul Jindal",
  "license": "",
  "dependencies": {
    "child_process": "^1.0.2",
    "express": "~4.0.0"
  },
  "homepage": "",
  "version": "1.0.0",
  "description": "client application for CCS"
}
```

Main file of the app

Called when **npm start** command is run

List of all dependencies

REST API

- REST – **REpresentational State Transfer**
- **Resources Based** – Deals with things instead of actions (like a person, address)
- **Representations** –

  Means how resources get manipulated.

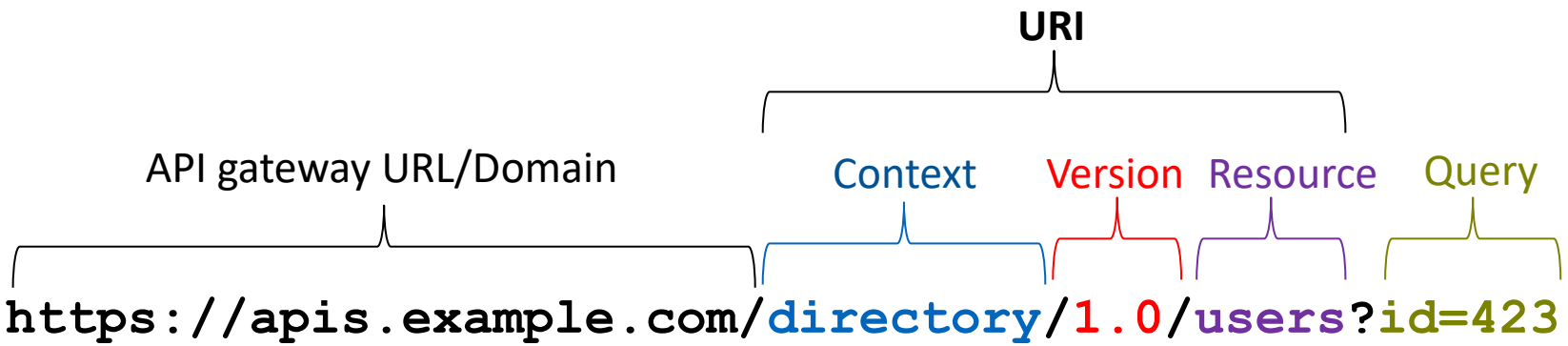
  Transferred between client and server (Typically JSON or XML)
- 6 constraints, which makes a web service – a true RESTful API [5]

1. **Uniform Interface** –
   - HTTP actions (GET, POST, PUT, DELETE)
   - Uniform Resource Identifiers (URIs) (resource names)
   - HTTP response (status, body)
2. **Stateless** –
   - Server contains no client state (No session, no history)
   - Each request contains enough context to process message.
3. **Client–server** –
   - Client application and server application MUST be able to evolve separately without any dependency on each other

REST API Continue..

4. **Cacheable –**
   - caching can be applied to resources when applicable and these resources MUST be declared cacheable.
   - Caching can be implemented on the server or client side.
5. **Layered system** –
   - Uses layered system architecture where for example we deploy the APIs on server A, and store data on server B and authenticate requests in Server C.
6. **Code on demand (optional)** –
   - Most of the time we will be sending the static representations of resources in form of XML or JSON.
   - Can return executable code to support a part of your application e.g. clients may call API to get a UI widget rendering code.

REST API example




```
{
   "id": "423",
   "name": "xyz",
   "status": "student"
}
```

Example JSON Response

MongoDB

- MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling [6].
- NoSQL Database, uses JSON like documents to store information.
- A record in MongoDB is a **document**, which is a data structure composed of field and value pairs.

```
{
  "id": "423",
  "name": "xyz",
  "status": "student"
}
```

field:value
field:value
field:value

- **Collection:** A grouping of MongoDB documents.
- **Databases** hold collections of documents.
- More Information here

Installation

Installation of Node and NPM

- Get a recent version of Node.js by adding a PPA (personal package archive) maintained by NodeSource for Ubuntu*

  wget -qO- https://deb.nodesource.com/setup_10.x | sudo -E bash -

- You can now install the Node.js package.

  sudo apt-get install -y nodejs

- For some npm packages to work (such as those that require building from source), you will need to install the build-essentials package

  sudo apt-get install -y build-essential

- Test Node: node -v (This should print a version number)
- Test NPM: npm -v (This should print NPM's version number)

*For other OS check: https://nodejs.org/en/download/

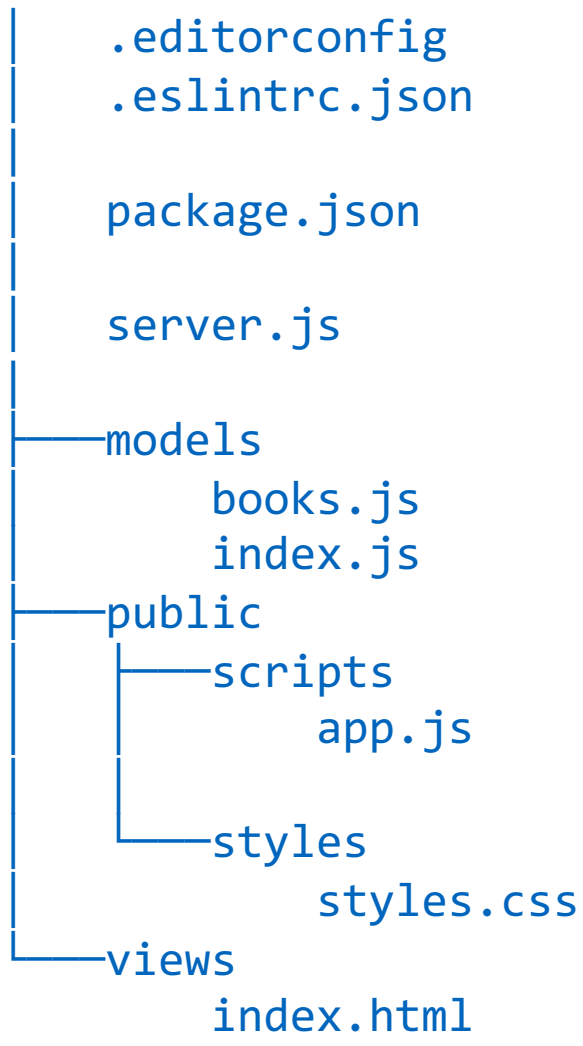
Install Mongo Database

Check here for installation steps for MongoDB

https://docs.mongodb.com/manual/administration/install-community/

Lets see some code…

Directory Structure of the Provided Application*



Used for editor configuration, like Indentation, spaces etc.

For handling dependencies

Main file for running back-end web server

Contains schema of the object to be stored in MongoDb

For HTML file styling and front-end JS

Contains HTML file

*Code is based upon ECMAScript 6 specification.

Models – Books.js

- We are using Mongoose to connect to MongoDB. See queries here.
- Schema Used in this application:

```
const mongoose = require('mongoose'),
 Schema = mongoose.Schema;
```

Importing Mongoose

```
const BooksSchema = new Schema({
      title: String,
      author: String,
      releaseDate: String,
      genre: String,
      rating: String,
      language: String
});
```

Creating a schema for the object that will be stored inside the mongo database

```
const BooksModel =
mongoose.model('books', BooksSchema);

module.exports = BooksModel;
```

Creating the model **books** and exporting it. The name **books** will be used by mongo DB for creating collection

Models – Index.js

- Similarly we can have more schemas as shown in previous slide in different files.
- We can now consolidate all schemas in one file and export them for further use.

```
const mongoose = require("mongoose");
mongoose.connect( process.env.MONGODB_URI ||
    "mongodb://localhost:27017/booksData",{
useNewUrlParser: true });
```

Connection to Mongo DB which is running locally at **localhost:27017** and database name as **booksData**

```
module.exports.books =
require("./books.js");
```

We export our **books** schema as **books** module from here.

server.js

- This is the main back-end service file.
- It is responsible for starting web server and contains all the API endpoints.

```
const express = require('express');
const app = express();
```

Express is used to create web server. More Info here

extract the entire body portion of an incoming request stream and exposes it on req.body.

```
app.use(express.urlencoded({extended: true}));
```

Accepts only UTF-8 encoding of the body

```
app.use(express.json());
```

Accepts any type of encoding of the body

```
// Set Static File Directory
app.use(express.static(__dirname +
'/public'));
```

Directory for static files like CSS files or images

server.js continue..

Routes/Endpoints

- **Root Endpoint "/" [GET]**

```
app.get('/', function homepage(req, res) {
  res.sendFile(__dirname +
'/views/index.html');
});
```

Notice these two calls.. These both are same

request

response

On accessing Root endpoint like on http://example.com/ calls the function **homepage** which is sending the **index.html** file

- **API information Endpoint "/api" [GET]**

GET request

```
app.get('/api', (req, res) => {

  res.json([…])
});
```

Called on endpoint **/api** like on http://example.com/api, which is sending some **JSON** information

server.js continue..

- **All stored books information Endpoint "/api/books" [GET]**

Empty means to get all

```
app.get('/api/books/', (req, res) => {

  db.books.find({}, function (err, books) {
    if (err) throw err;

    res.json(books);

  });
});
```

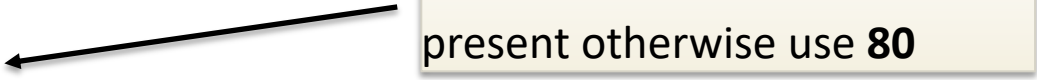
uses the books model and query to mongo database to get all objects stored.

books is an array of found JSON objects and it is returned back to the user.

- Similarly there are other API endpoints for POST, PUT, DELETE

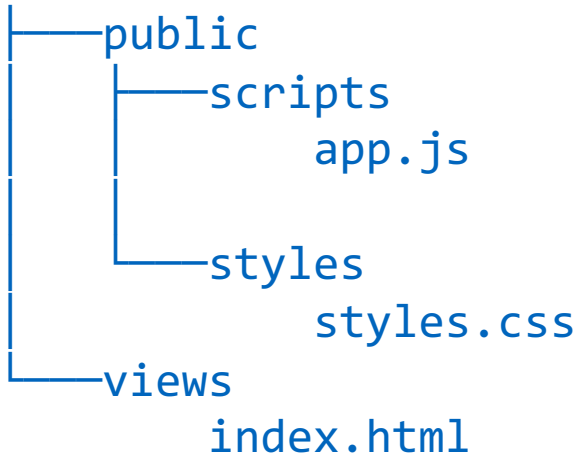
server.js continue..

- To Listen on a particular port for the webserver.
- Here it runs on port 80.
- Webserver can be accessed at http://localhost:80 from browser




```
app.listen(process.env.PORT || 80, () => {
  console.log('Express server is up and
running on http://localhost:80/');
});
```

Public and Views Directory



- Here the code related to front-end is specified like how to style and display the webpage.
- You can access the html webpage at the address http://localhost:80 , once the server is running.

Running the application

Download the Provided Application

1. Download the provided application source zip file from Moodle.
2. Unzip the file

```
sudo apt-get install unzip

unzip cloud-computing-exercise-app.zip -d cloud-computing-exercise-app
```

Start MongoDB

1. Open a Terminal and make the current directory as “cloud-computing-exercise-app”

   cd cloud-computing-exercise-app

2. Make a directory with the name “data”. This will store all mongo database information

   mkdir data

3. Run the following command to start mongo database

   mongod --dbpath=./data

```
[initandlisten] MongoDB starting : pid=18352 port=27017 dbpath=data 64-bit host=DESKTOP-67ONKOH
[initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
[initandlisten] db version v3.4.7
[initandlisten] git version: cf38c1b8a0a8dca4a11737581beafef4fe120bcd
[initandlisten] OpenSSL version: OpenSSL 1.0.1u-fips  22 Sep 2016
[initandlisten] allocator: tcmalloc
[initandlisten] modules: none
[initandlisten] build environment:
[initandlisten]     distmod: 2008plus-ssl
[initandlisten]     distarch: x86_64
[initandlisten]     target_arch: x86_64
[initandlisten] options: { storage: { dbPath: "data" } }
```

Start Node.js Application

1. Open another Terminal and make the current directory as “cloud-computing-exercise-app”

   cd cloud-computing-exercise-app

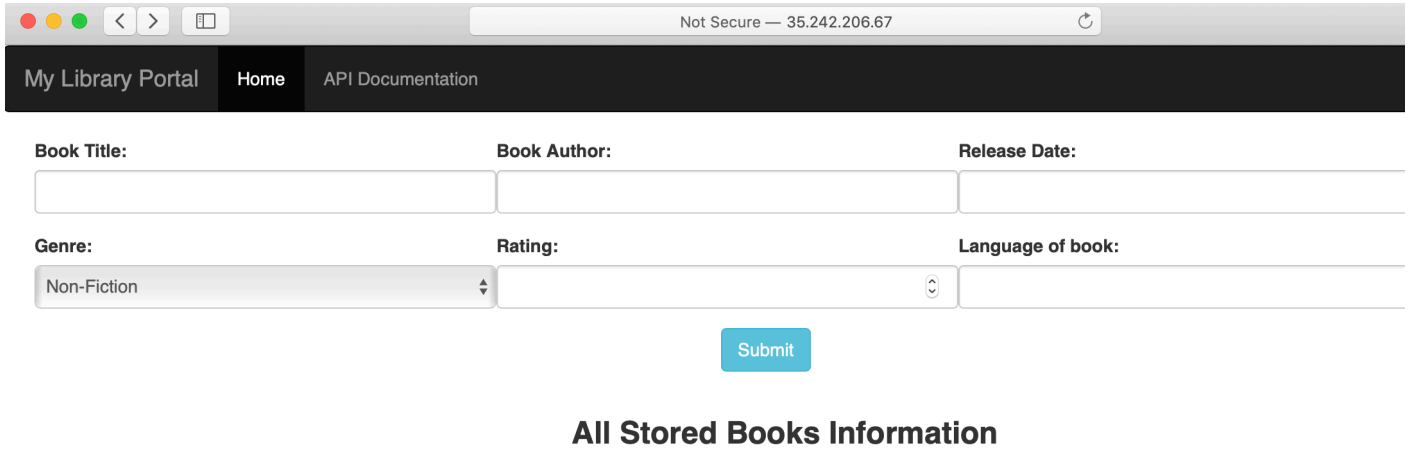
2. Install All node.js dependencies present inside **package.json** file

   npm install

3. Start the application

   sudo node server.js

4. You can now check the browser at address http://localhost:80



GCP VM Creation

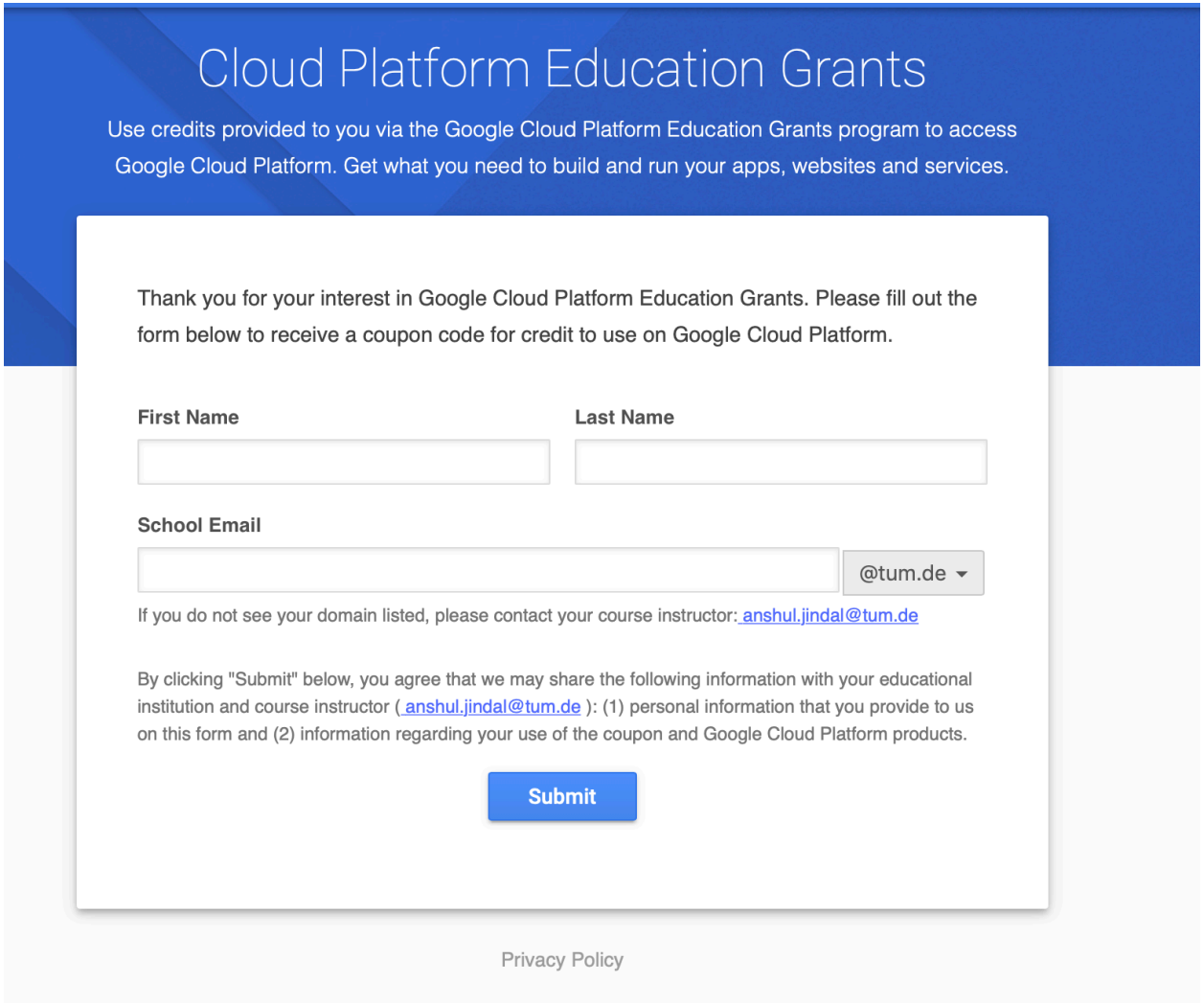
GCP Registration Link

- Click on the given GCP registration link: Will be added on Piazza and Moodle
- Instructions:

Will be added on Piazza and Moodle

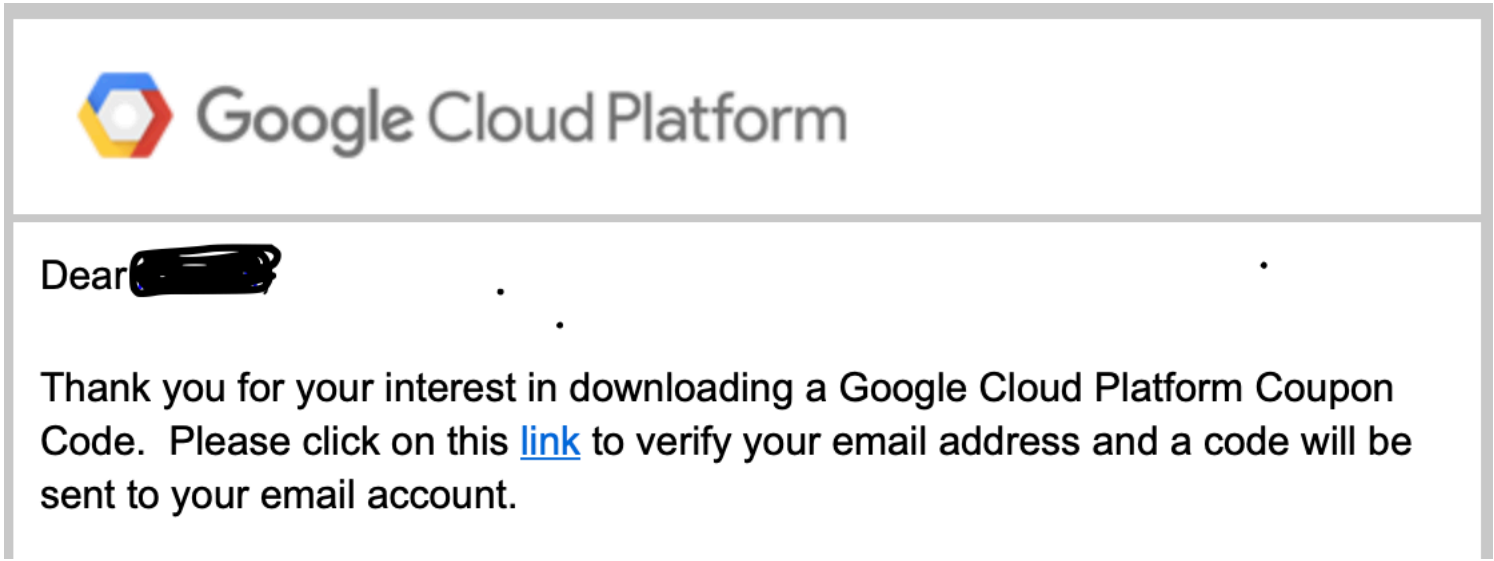
GCP Registration

- After clicking the link, fill the below shown information.
- Use official TUM email id only.



GCP Registration Cont..

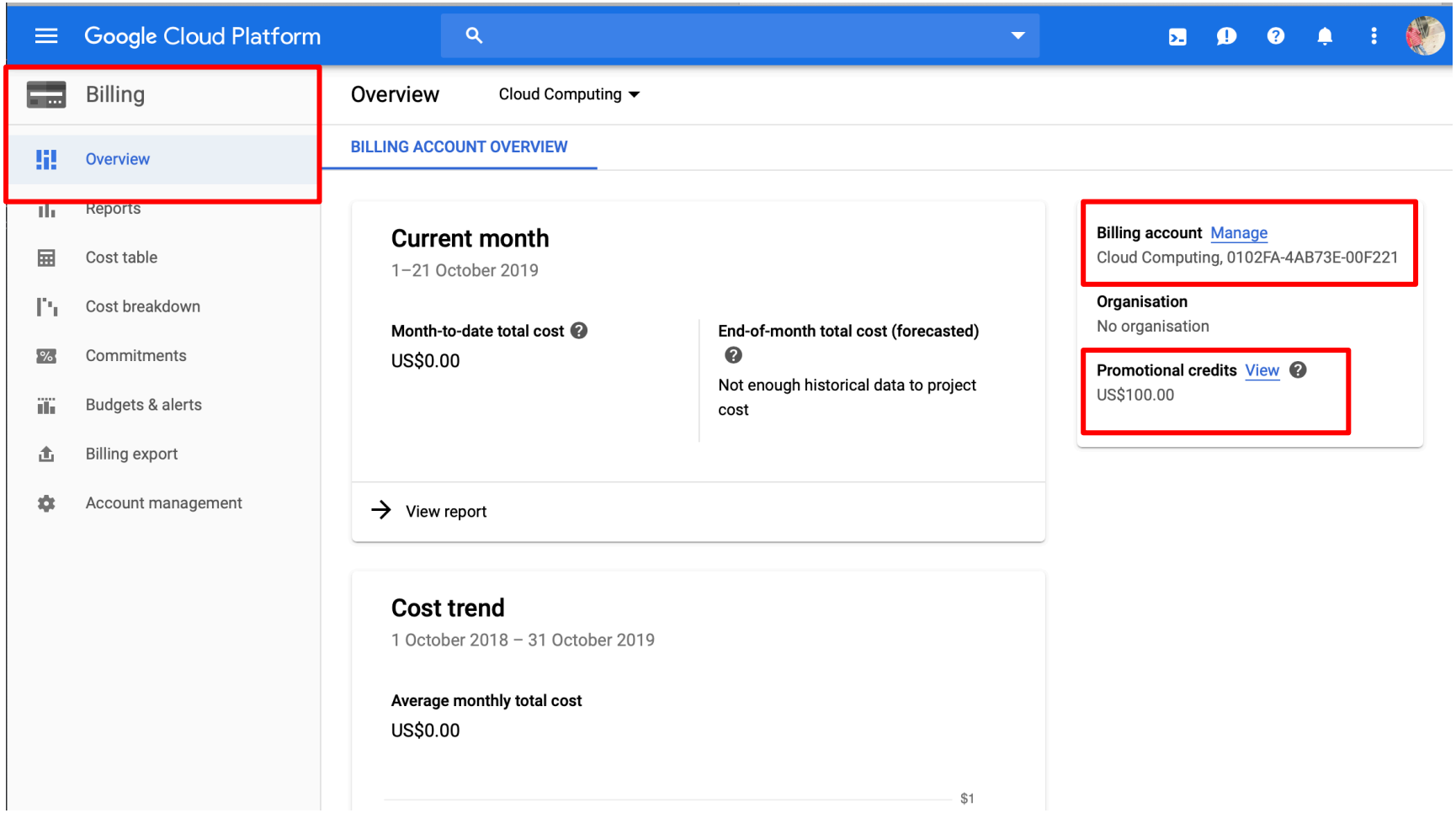
- After submitting the information you will receive an email like this:



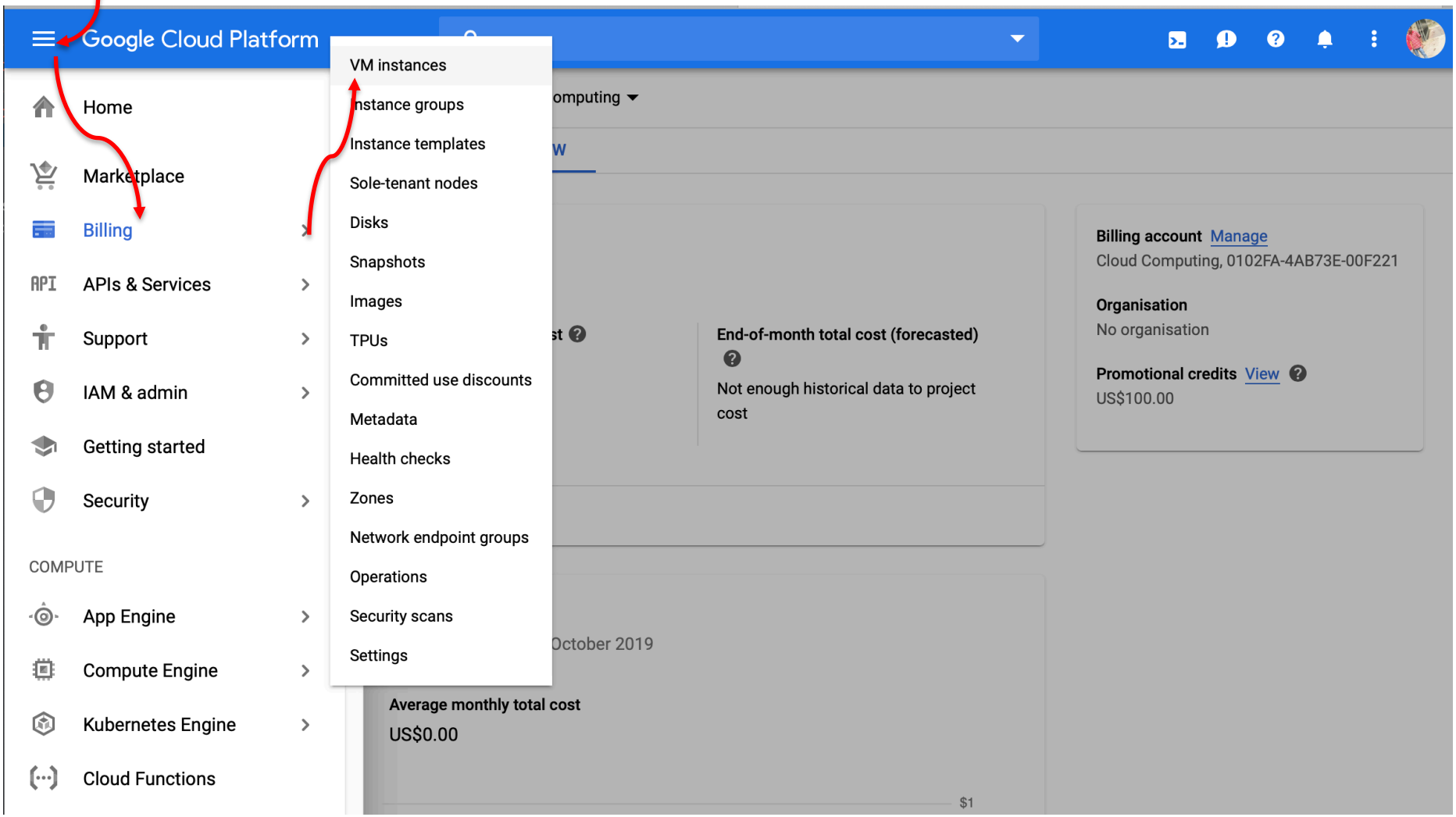
- Verify the email id and you will then get the coupon code to your email.
- Click the link in the new email from GCP to redeem it .

Login into GCP : Home page

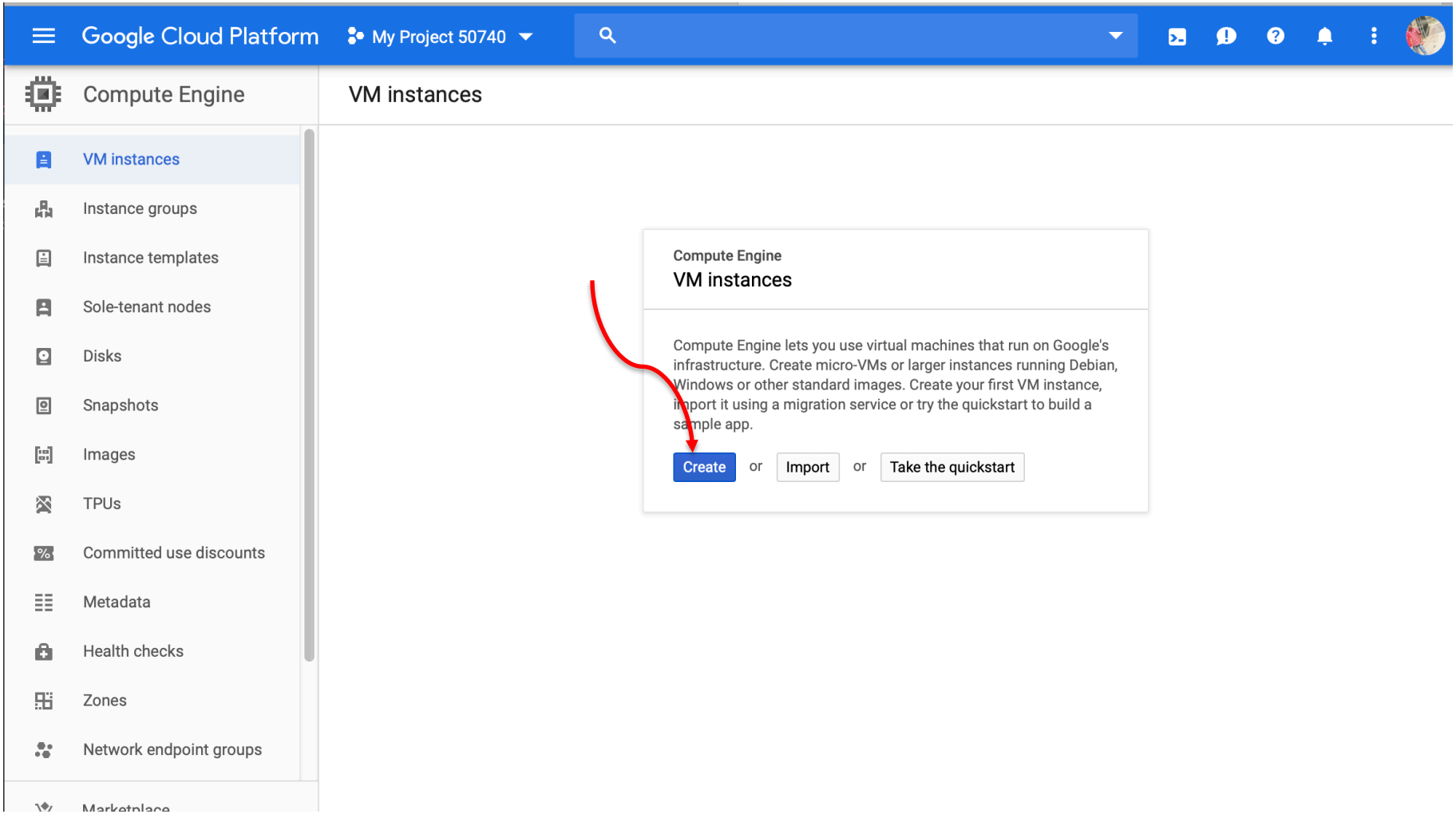
Below is the homepage displayed, after logging into GCP



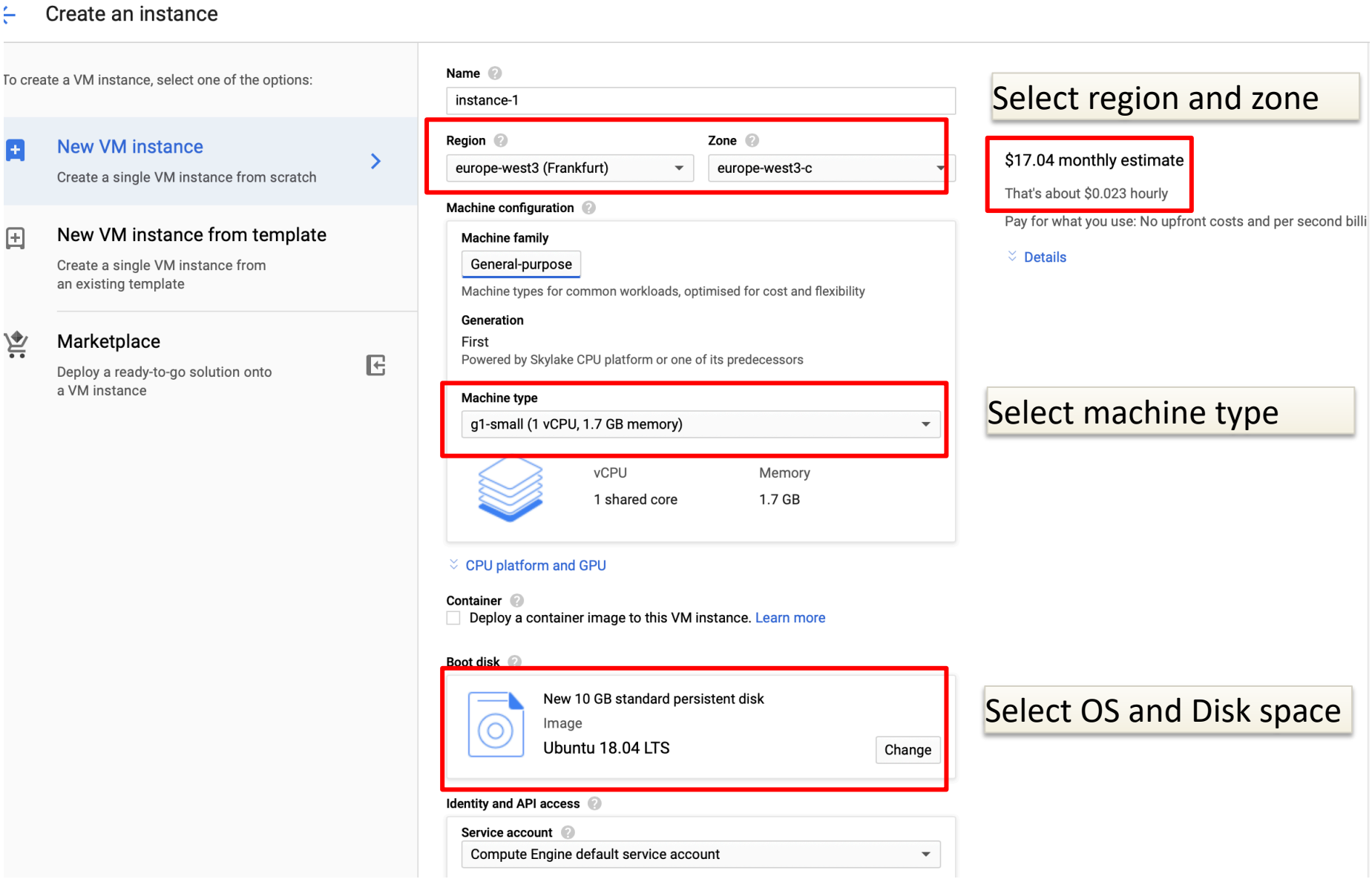
Start a VM: Step1



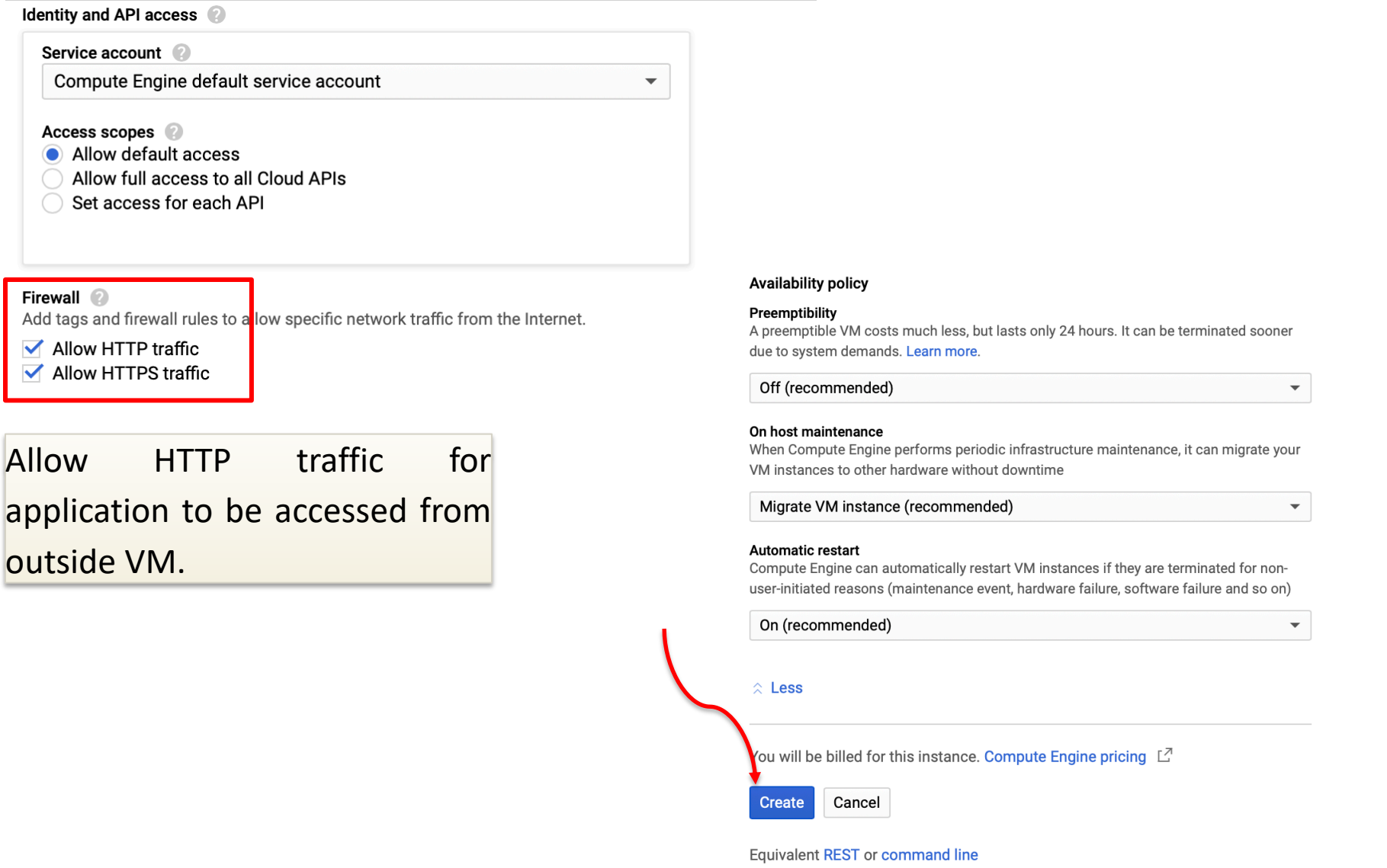
Start a VM: Step2



Start a VM: Step 3



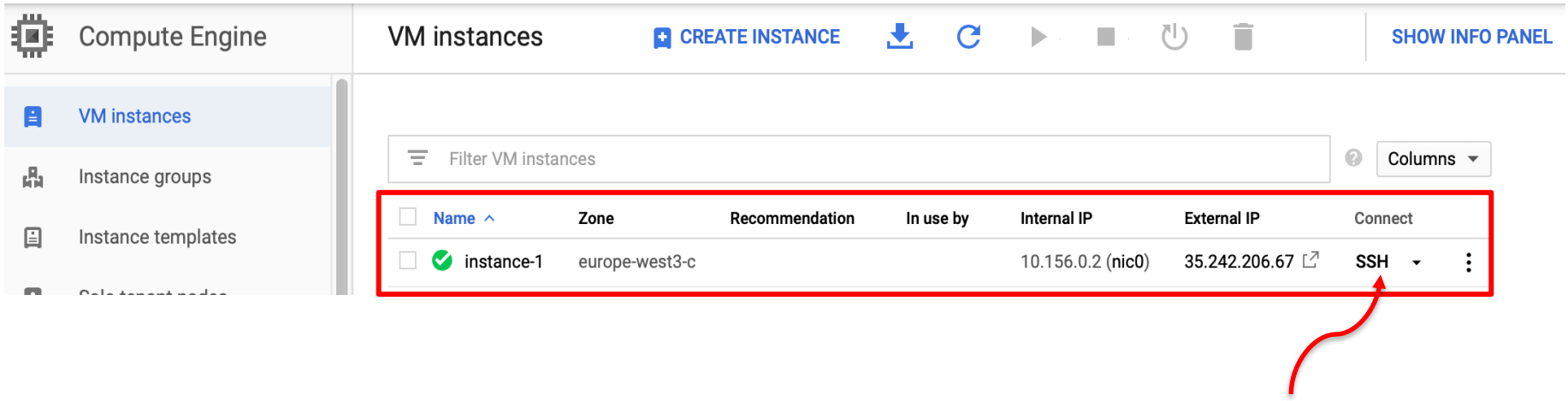
Start a VM: Step4



Allow HTTP traffic for application to be accessed from outside VM.

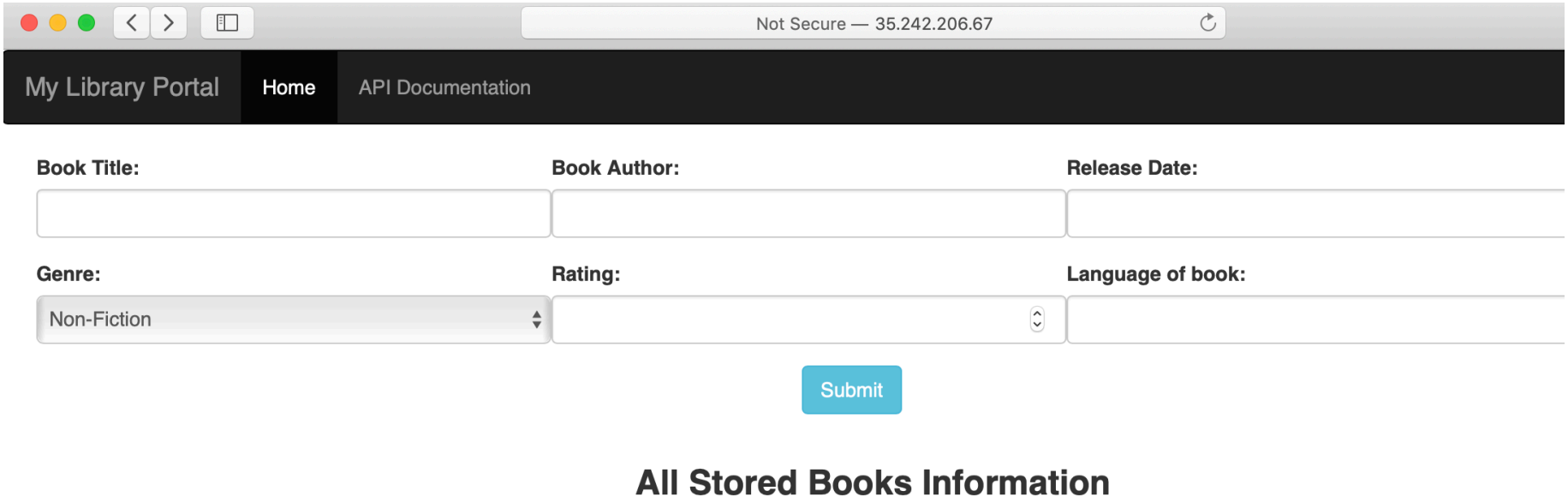
Start a VM: Step5

- SSH into VM



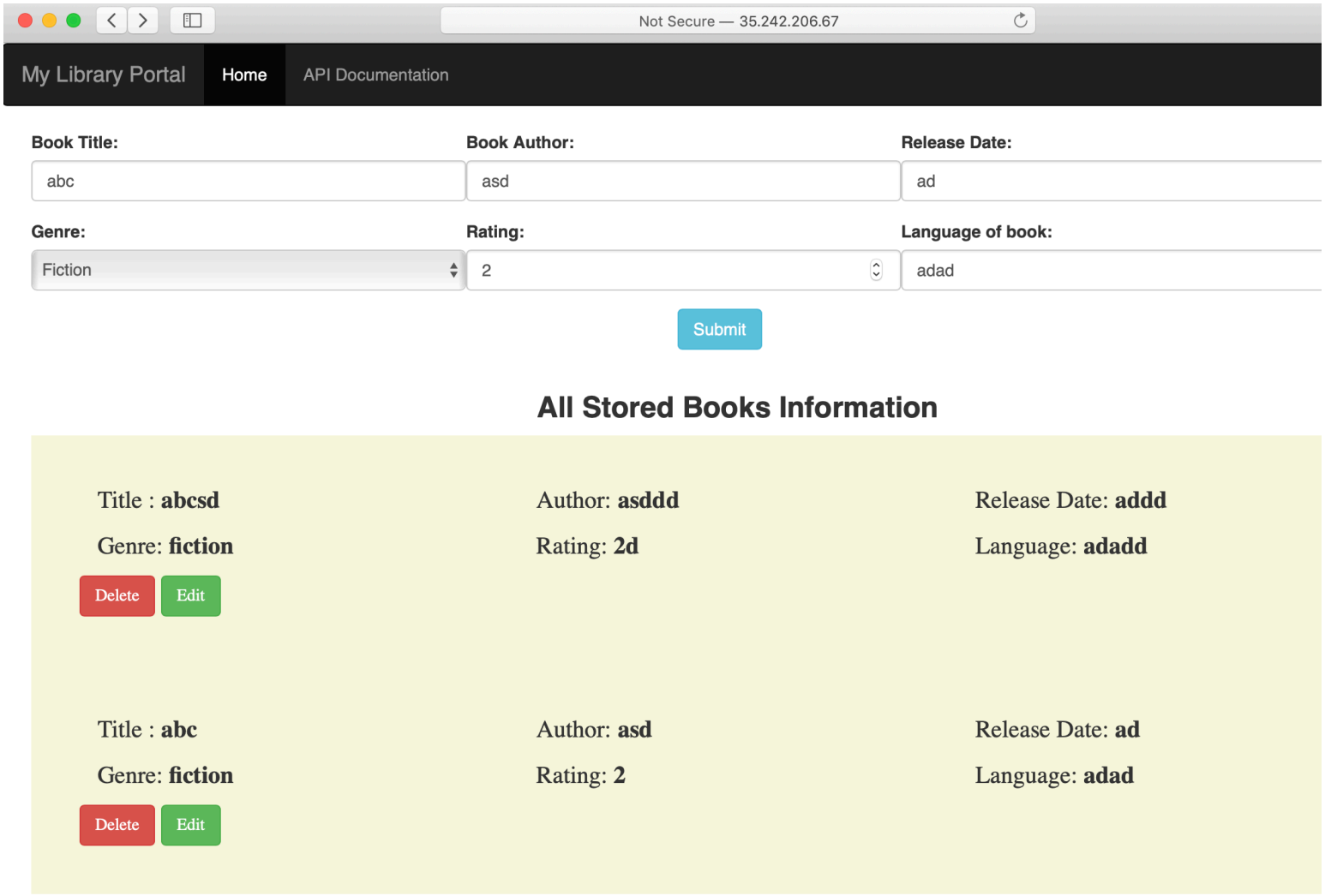
Run and access the application

- Run the given application as mentioned in the instructions.
- Go to URL : http://VM_PUBLIC_IP:80



Run and access the completed application

- Follow the same instructions to run the application.
- Go to URL : http://VM_PUBLIC_IP:80 and test the queries.



Tasks to be completed

Tasks to be completed

1. Document all your api endpoints in a simple hardcoded JSON object in the "/api" endpoint. Only some are provided as an example, write others

```
app.get('/api', (req, res) => {
  // TODO: Document all your api endpoints below as a
  res.json({
    message: 'Welcome to my app api!',
    documentationUrl: '', //leave this also blank for
    baseUrl: '', //leave this blank for the first exe
    endpoints: [
      {method: 'GET', path: '/api', description: 'Des
      {method: 'GET', path: '/api/profile', descriptio
      {method: 'GET', path: '/api/books/', descriptio
      // TODO: Write other API end-points description
    ]
  })
});
```

Tasks to be completed Continue..

2. Complete the /api/profile endpoint. You can add here fake information too, to make it more interesting like Name as Jon Snow, homeCountry as winterfell ☺

```
// TODO:  Fill the values
app.get('/api/profile', (req, res) => {
  res.json({
    'name': '',
    'homeCountry': '',
    'degreeProgram': '',//informatics or
    'email': '',
    'deployedURLLink': '',//leave this bl
    'apiDocumentationURL': '', //leave th
    'currentCity': '',
    'hobbies': []

  })
});
```

Tasks to be completed Continue..

3. Complete other missing APIs. Study mongoose queries here.

- /api/books [POST] : To store new book information and return the stored information as JSON.

```
app.post('/api/books/', (req, res) => {

  /*
   * New Book information in req.body
   */
  console.log(req.body);
  /*
   * TODO: use the books model and create a new object
   * with the information in req.body
   */
  /*
   * return the new book information object as json
   */
  var newBook = {};
  res.json(newBook);

});
```

Tasks to be completed Continue..

3. Complete other missing APIs. Study mongoose queries here.

- /api/books/:id [PUT] : To Update a book information based upon the provided id and new information. After updating return the updated JSON.

```
app.put('/api/books/:id', (req, res) => {
  /*
   * Get the book ID and new information of book from
   * the request parameters
   */
  const bookId = req.params.id;
  const bookNewData = req.body;
  /*
   * TODO: use the books model and find using the bookId and
   * update the book information
   */
  /*
   * Send the updated book information as a JSON object
   */
  var updatedBookInfo = {};
  res.json(updatedBookInfo);

});
```

Tasks to be completed Continue..

3. Complete other missing APIs. Study mongoose queries here.

- /api/books/:id [DELETE]: To delete a book information based upon the id. Return the delete book information as the JSON

```
app.delete('/api/books/:id', (req, res) => {
  /*
   * Get the book ID of book from the request
parameters
   */
  const bookId = req.params.id;
 /*
   * TODO: use the books model and find using
   * the bookId and delete the book
   */
  /*
   * Send the deleted book information as a JSON object
   */
  var deletedBook = {};
  res.json(deletedBook);
});
```

Submission

Submission Instructions

To submit your application results you need to follow this :

1. Open the Cloud Class server url : https://cloudcom.caps.in.tum.de/
2. Login with your provided username and password.
3. After logging in, you will find the button for **exercise1**
4. Click on it and a form will come up where you must provide
   - VM IP on which your application is running

   Example:

   10.0.23.1

5. Then click submit.
6. You will get the correct submission from server if everything is done correctly.

Deadline for submission is two weeks from now (exact date and time is mentioned on the server)

Further Information

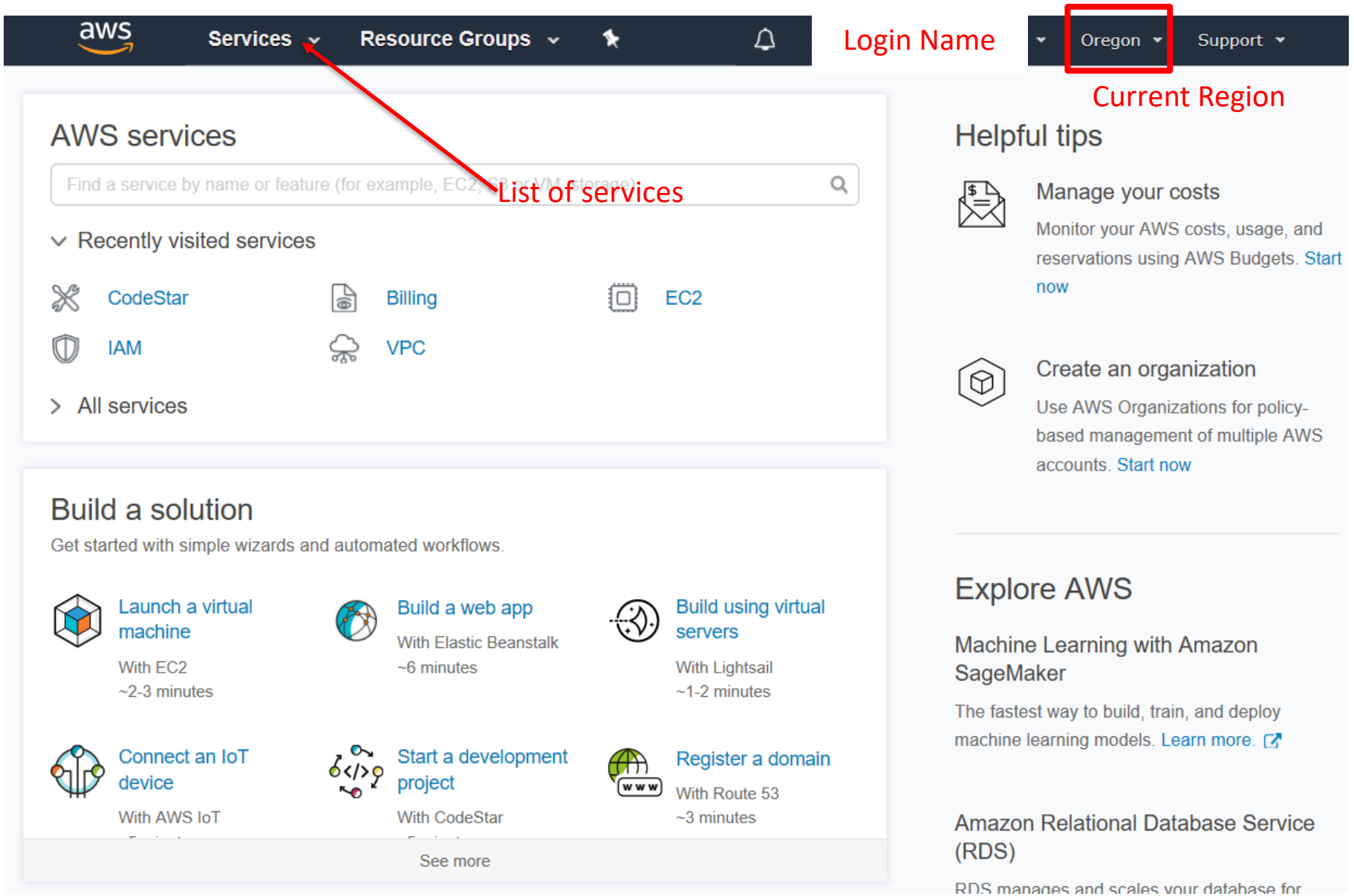
- There are many source code management systems out there -- But we recommend you to use GitLab at LRZ https://gitlab.lrz.de/users/sign_in

- Editors (You can get the student licensed version for them):
  - WebStorm https://www.jetbrains.com/webstorm/
  - Visual Studio Code https://code.visualstudio.com/

- For further Node.js learning you check this https://www.youtube.com/watch?v=-u-j7uqU7sI&list=PL6gx4Cwl9DGBMdkKFn3HasZnnAqVjzHn_&index=1 tutorial

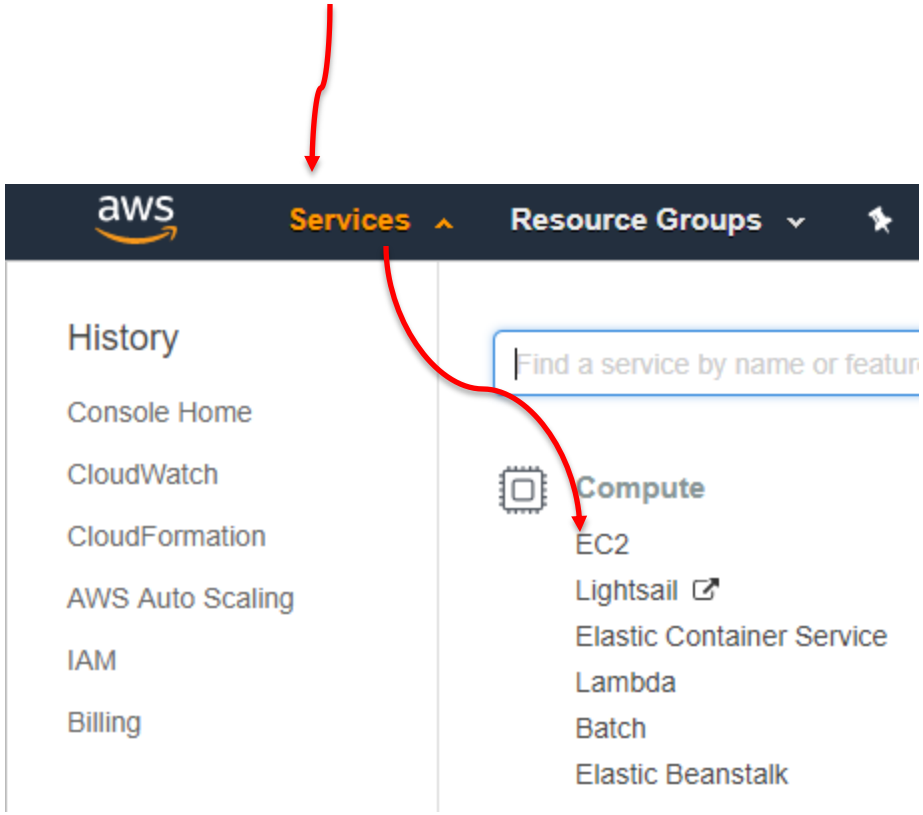
AWS VM Creation

AWS Management Console



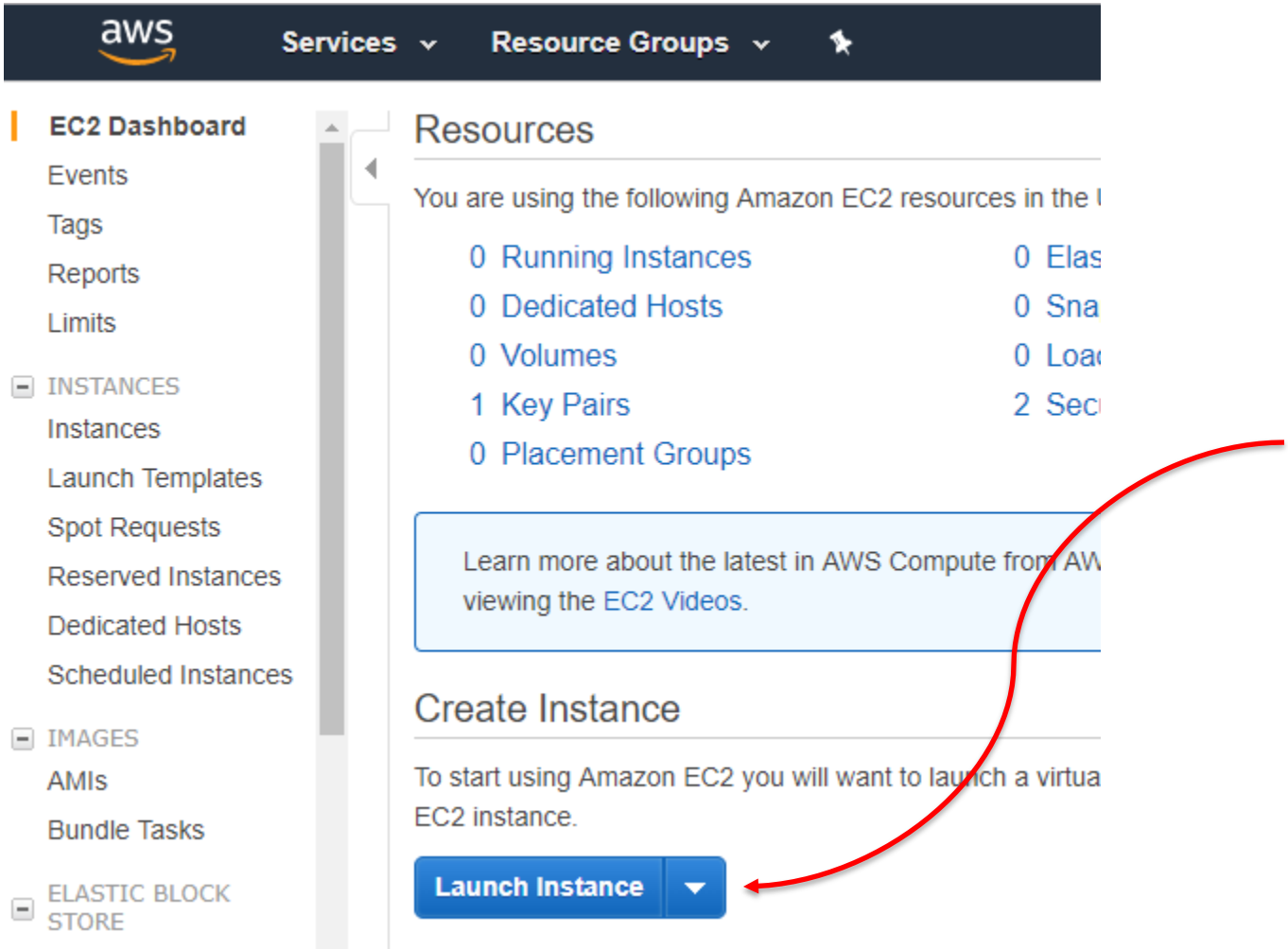
Start EC2 Instance

4. Now we will be starting the EC2 instance. Click on services and choose EC2



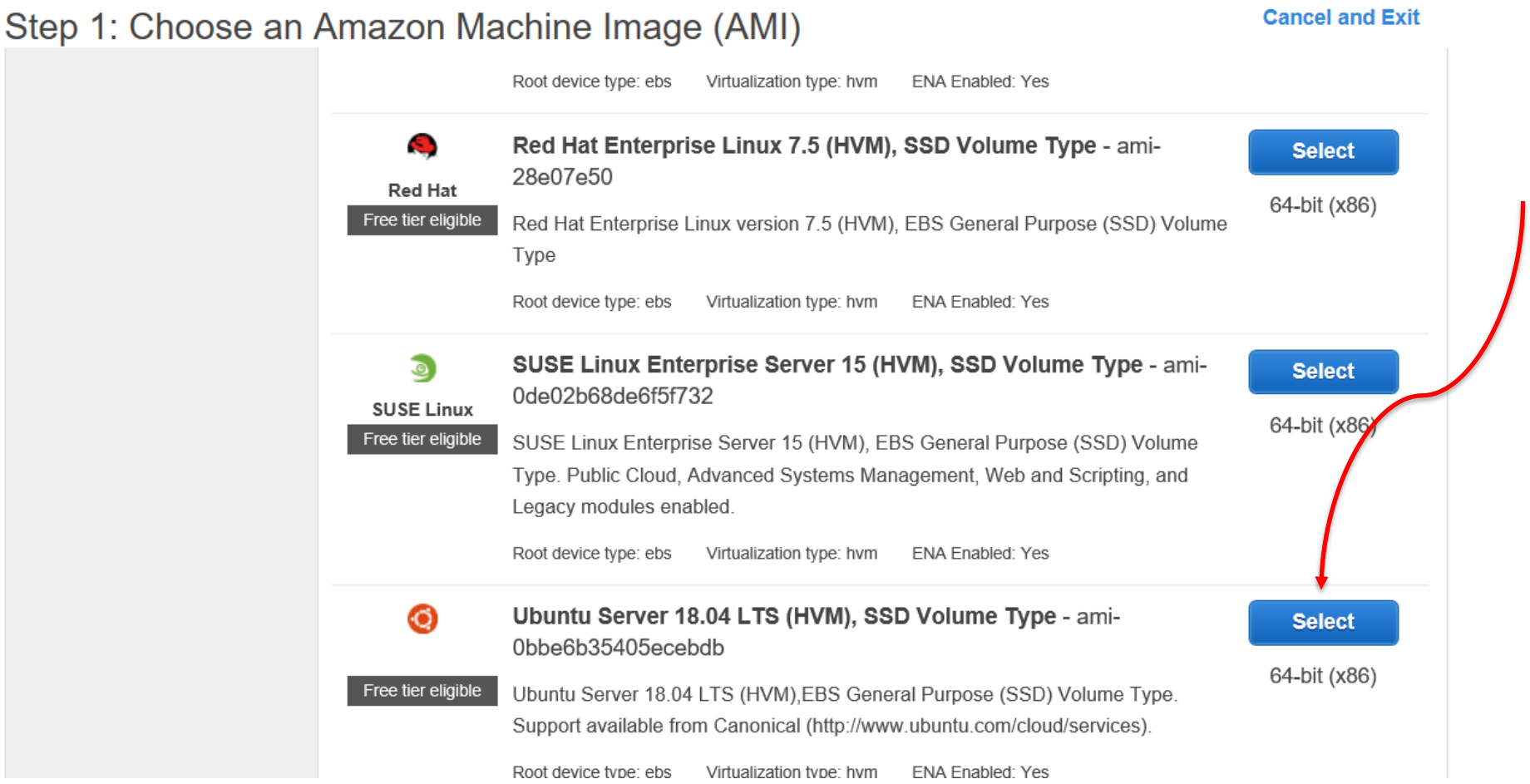
EC2 Dashboard

5. After clicking on EC2, you would go into EC2 dashboard. Now here click on **Launch Instance**



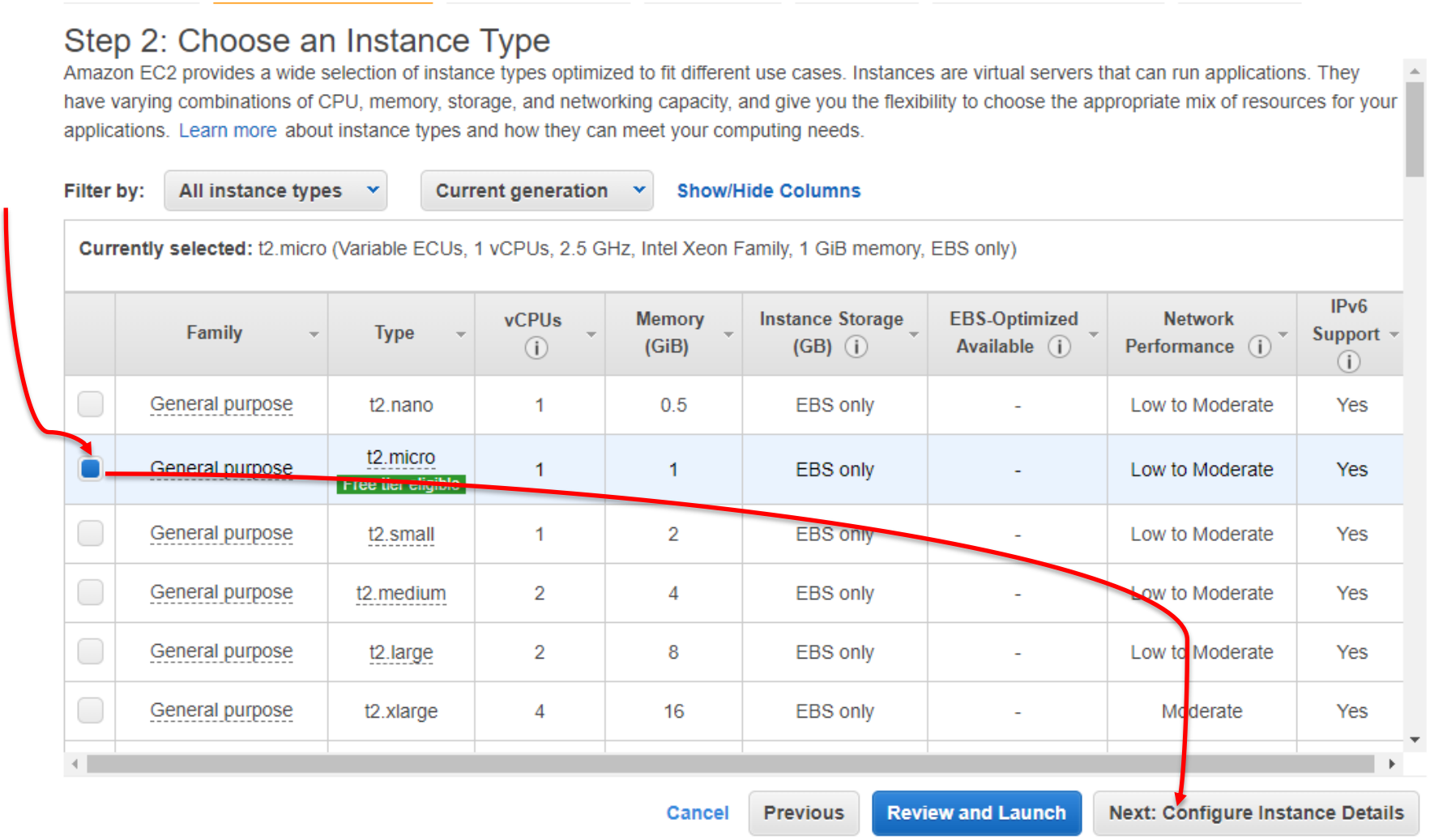
Launching an EC2 Instance : Step 1 (Selecting AMI)

6. Select an AMI. We will be selecting here **Ubuntu Server 18.04 LTS (HVM), SSD Volume Type**



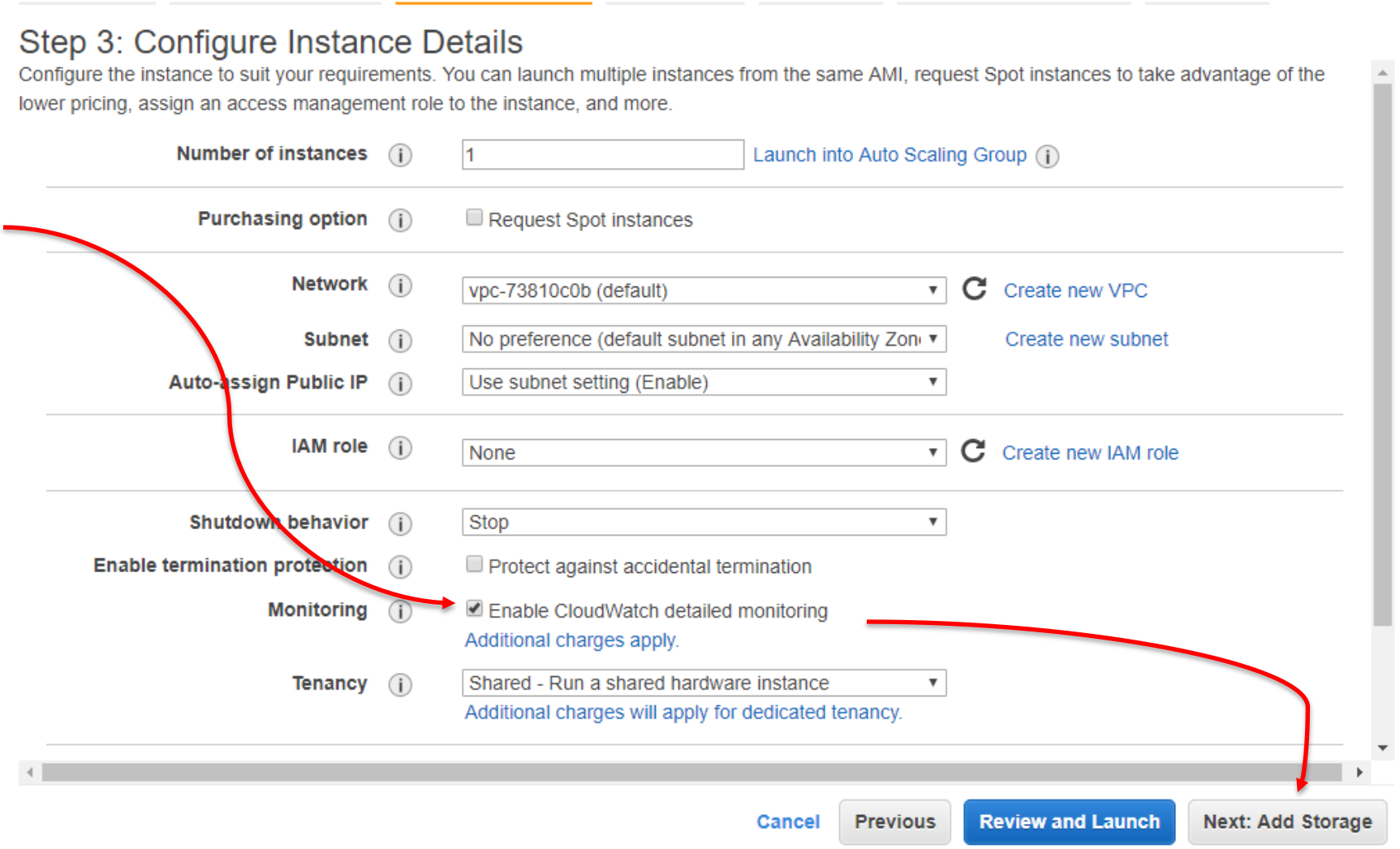
Launching an EC2 Instance : Step 2 (Instance Type)

7. Choose an Instance type. We will be selecting here **t2.micro.** For instance types list check here.



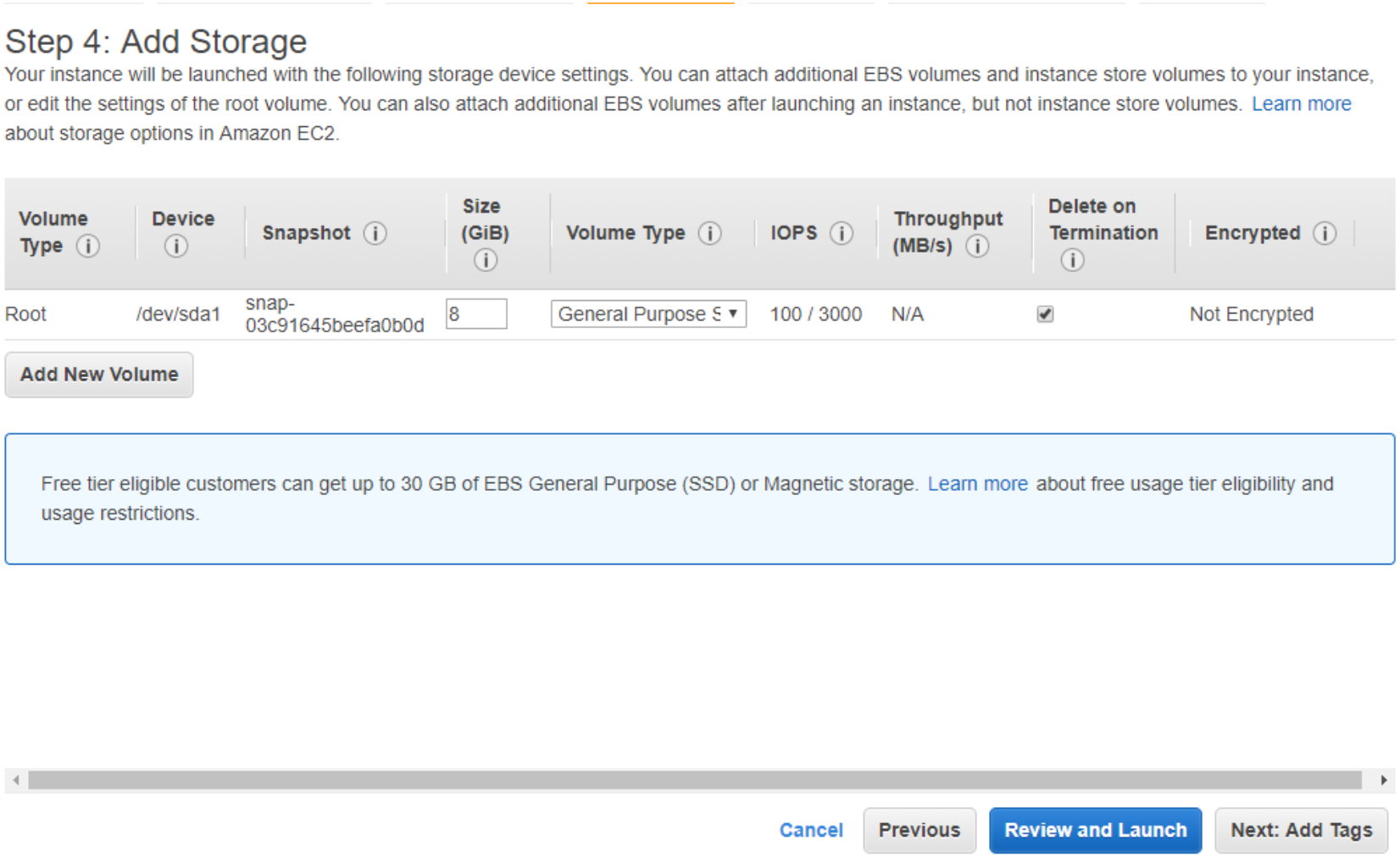
Launching an EC2 Instance : Step 3 (Instance Config.)

8. Configure the Instance details. You can leave everything to default.



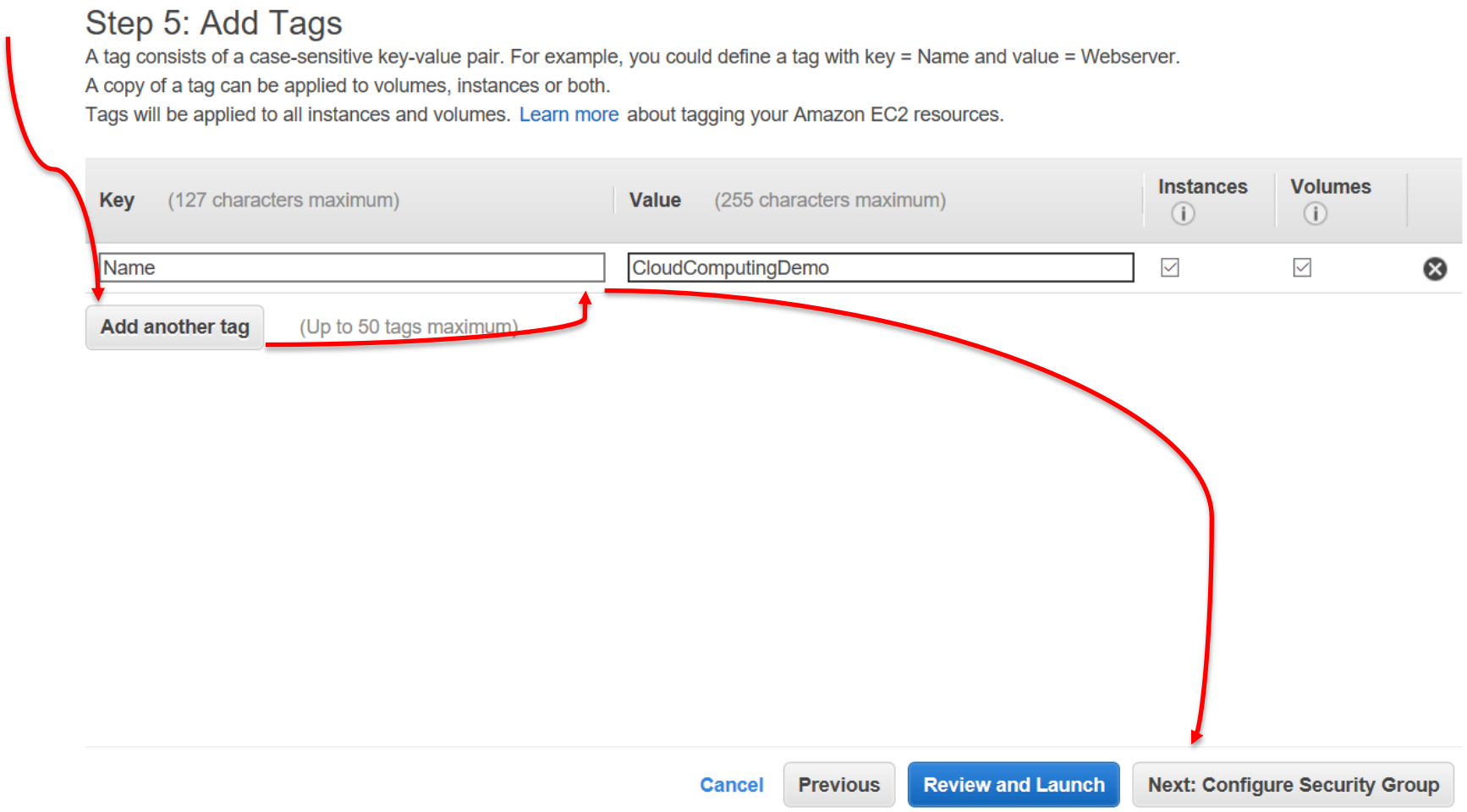
Launching an EC2 Instance : Step 4 (Add Storage)

9. Next step is to add the storage to the VM, we will leave here also to default **8GiB SSD.**



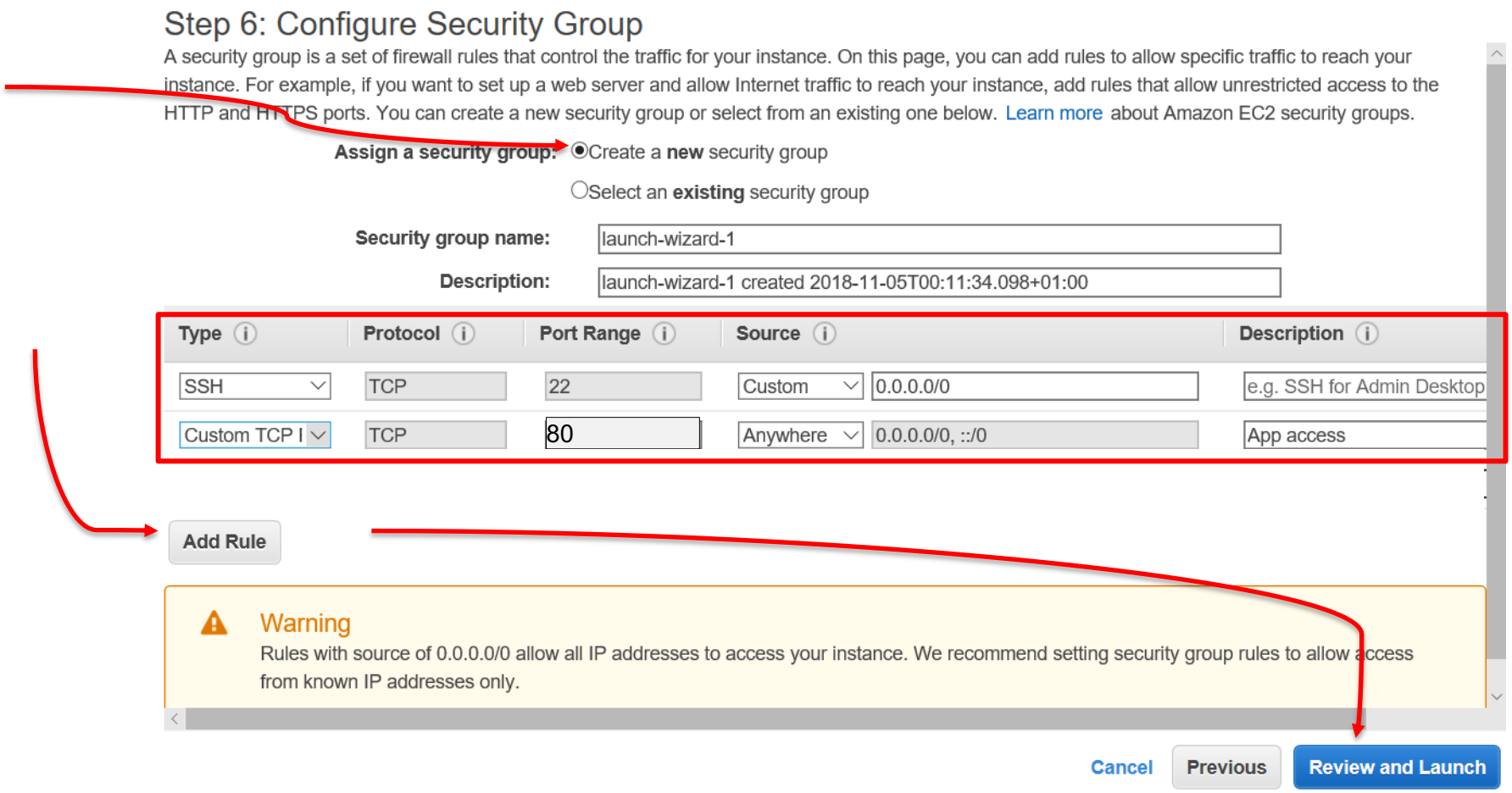
Launching an EC2 Instance : Step 5 (Add Tags)

10. Here we can add tag to our Instance. It's key value pair. We will be adding one tag which is the **Name** as shown below. You can give any name.



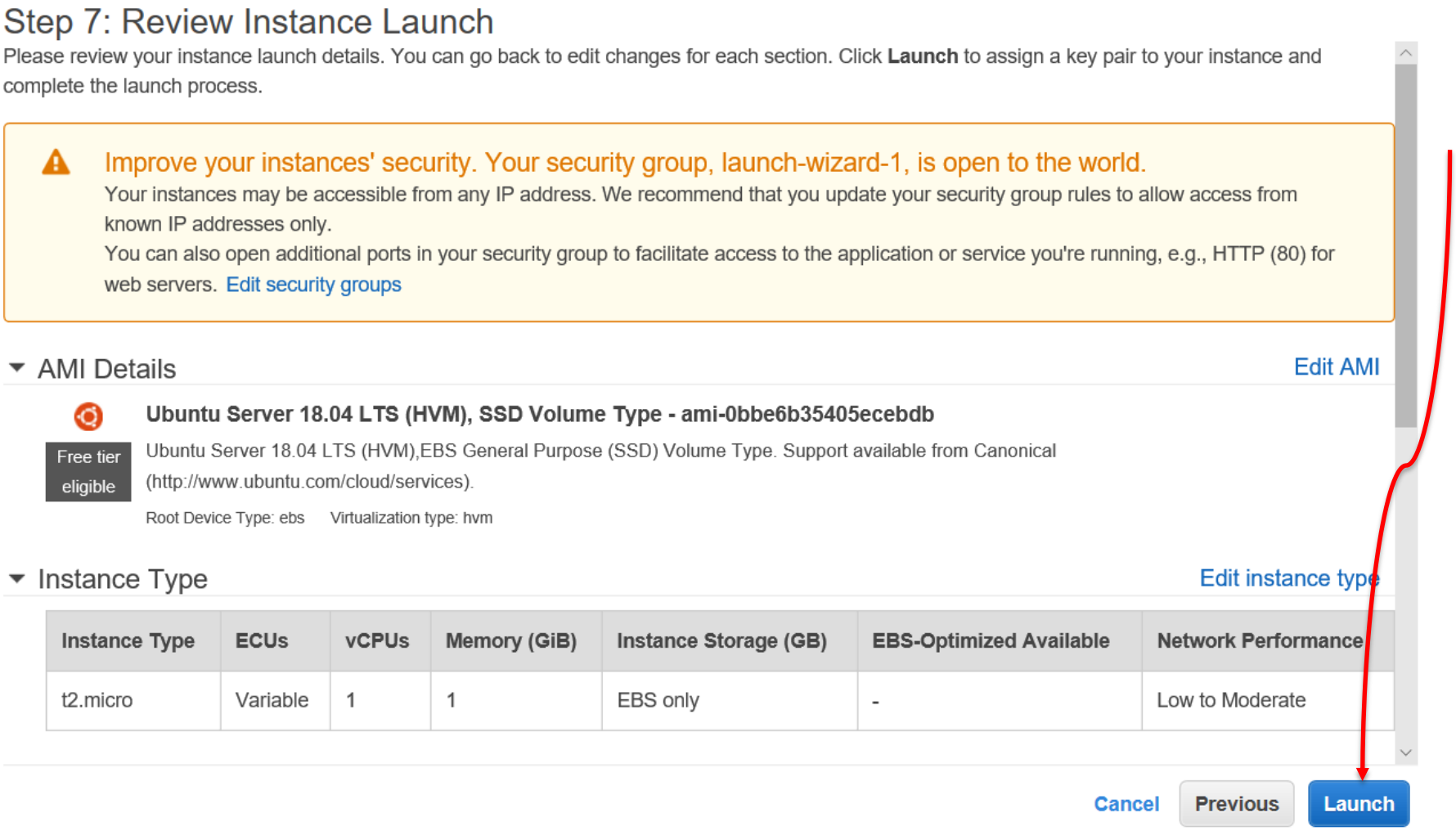
Launching an EC2 Instance : Step 6 (Configure SG)

11. We will create here a security group
    - Allow from all IPs the SSH port **22**
    - Allow from all IPs the port **80** (Our Application will be hosted on this port)



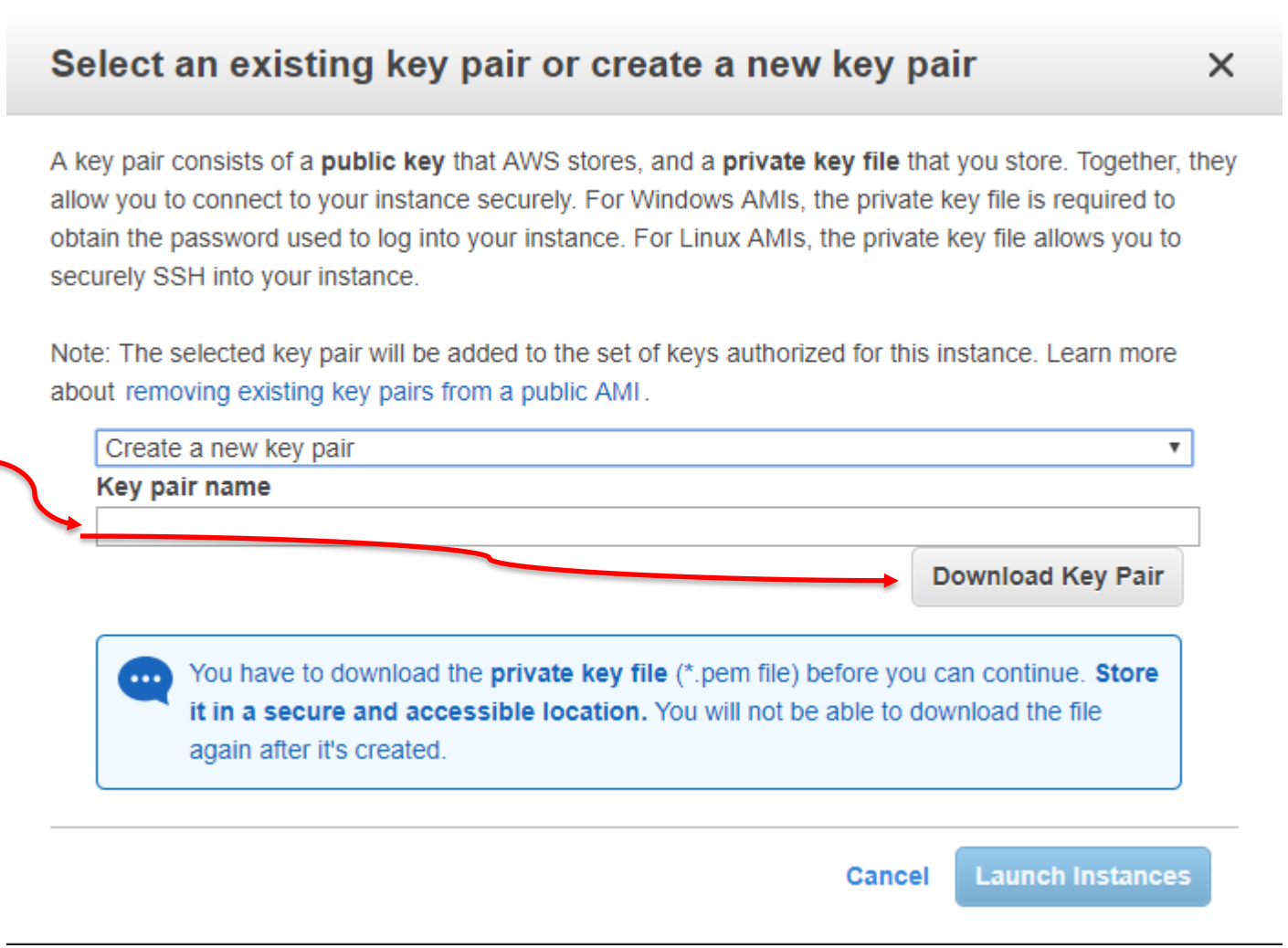
Launching an EC2 Instance : Step 6 (Review & Launch)

12. Now all the configuration is done, you can review your configuration. After review click on Launch.



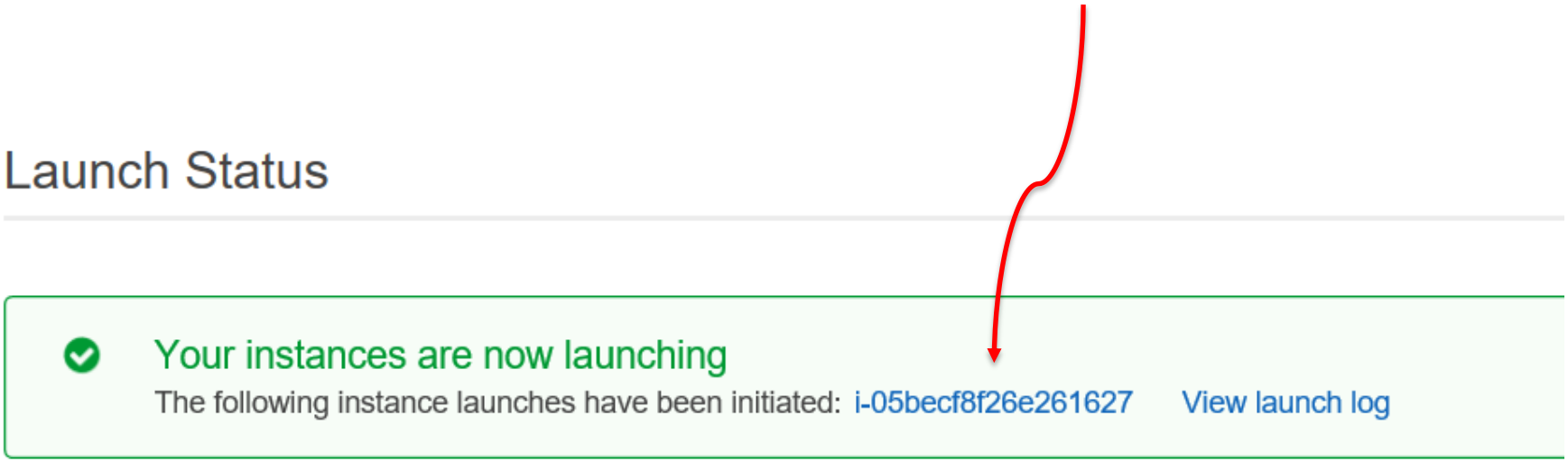
Launching an EC2 Instance : Step 7 (Download Key)

13. If you don't have already a key pair **create a new one** and download the pem file. You can give any name to this file.



Launching an EC2 Instance : Instance Started

The Instance has started, click on the name to see it's status



Launching an EC2 Instance : Instance Status

Check for the instance status and it's public ip.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
CloudCompu…	i-05becf8f26e261627	t2.micro	us-west-2a	running	Initializing

Instance: i-05becf8f26e261627 (CloudComputingDemo) **Public DNS:** ec2-34-217-36-129.us-west-2.compute.amazonaws.com

Description | Status Checks | Monitoring | Tags

Instance ID	i-05becf8f26e261627	Public DNS (IPv4)	ec2-34-217-36-129.us-west-2.compute.amazonaws
Instance state	running	IPv4 Public IP	34.217.36.129
Instance type	t2.micro	IPv6 IPs	-
Elastic IPs		Private DNS	ip-172-31-39-153.us-west-2.compute.internal
Availability zone	us-west-2a	Private IPs	172.31.39.153
Security groups	launch-wizard-1. view inbound rules. view outbound rules	Secondary private IPs	

Launching an EC2 Instance : Instance SSH Login (Linux)

For Logging into instance

- Linux or mac:
  - ssh -i “KEY_FILE_PATH" ubuntu@PUBLIC_IP

For Example:

ssh -i "aws_key.pem" ubuntu@34.217.36.219

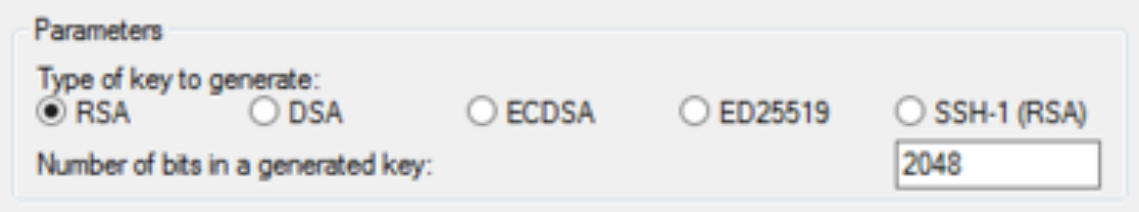
Launching an EC2 Instance : Instance SSH Login (Win.)

For Windows:

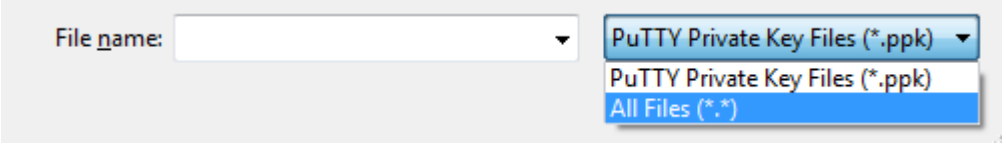
- Download **Putty** from here

  https://www.chiark.greenend.org.uk/~sgtatham/putty/

- Download **PuttyGen**
  - 32 bit https://the.earth.li/~sgtatham/putty/latest/w32/puttygen.exe
  - 64 bit https://the.earth.li/~sgtatham/putty/latest/w64/puttygen.exe
- After Download open puttygen
- Under **Type of key to generate**, choose **RSA**

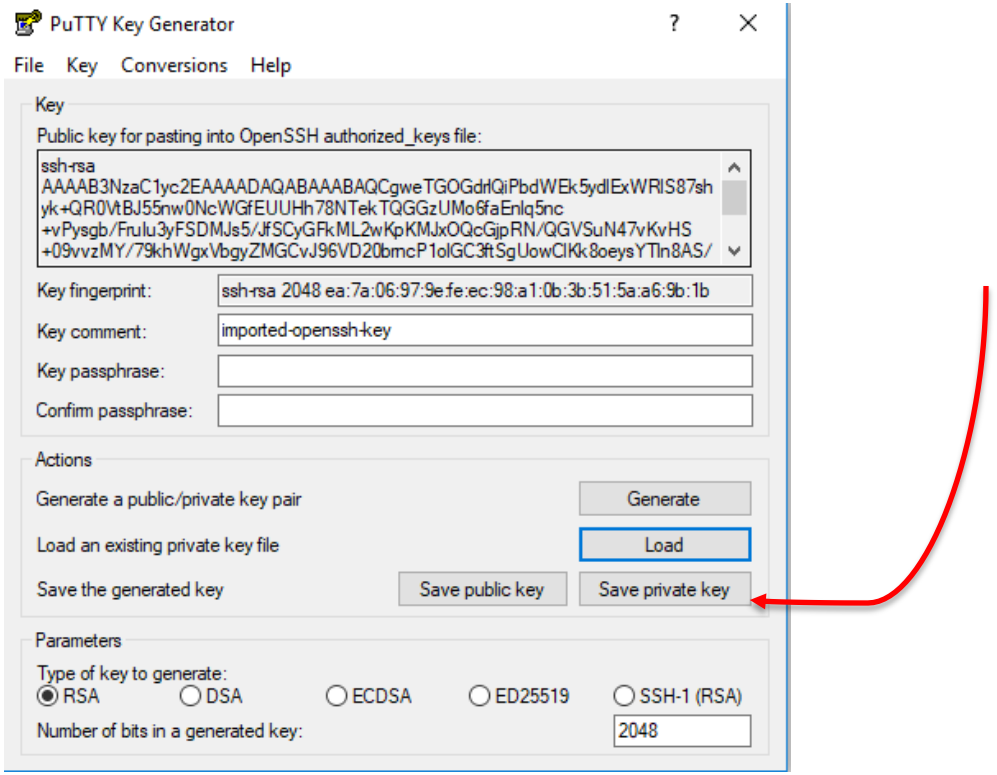


- Choose Load. By default, PuTTYgen displays only files with the extension .ppk. To locate your .pem file, select the option to display files of all types.



Launching an EC2 Instance : Instance SSH Login (Win.)

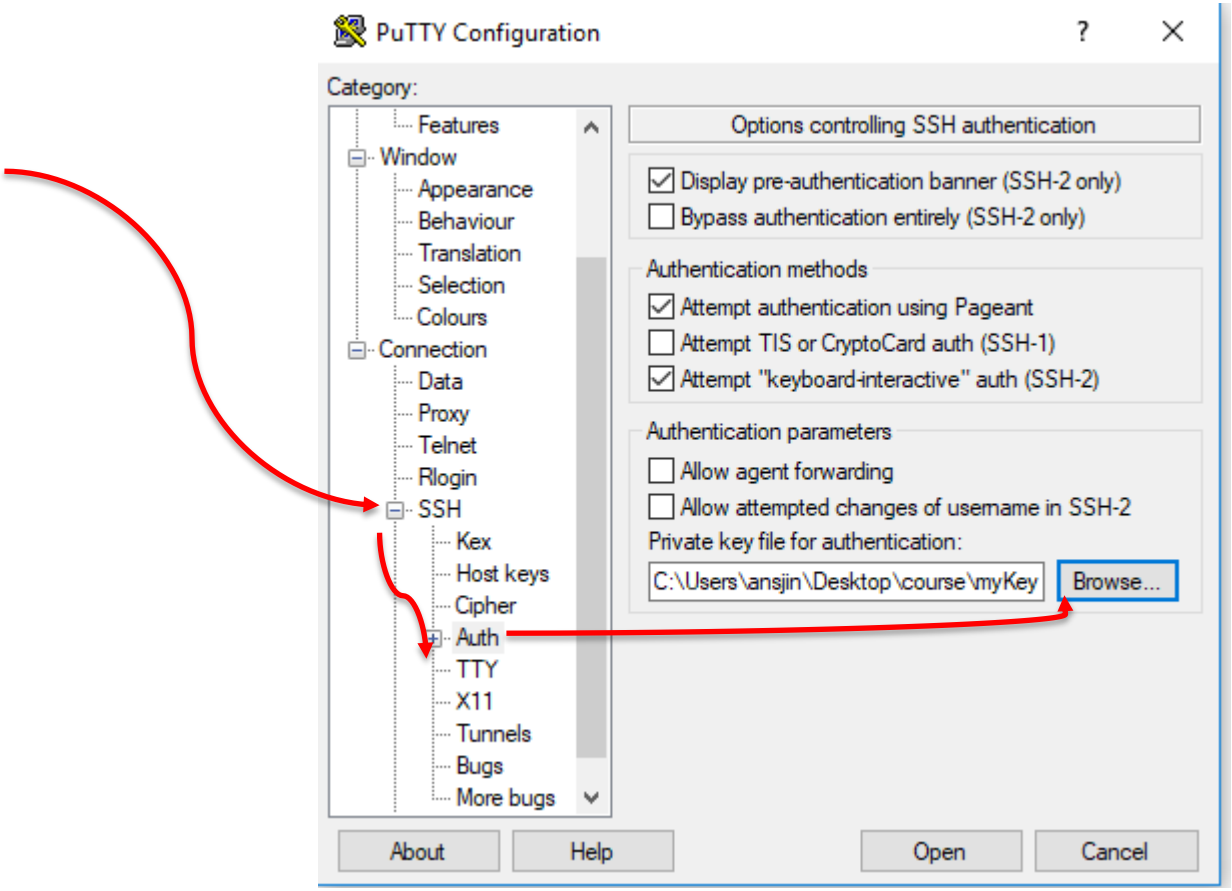
- Select your .pem file for the key pair that you specified when you launched your instance, and then choose Open. Choose OK to dismiss the confirmation dialog box.
- Choose **Save private key** to save the key in the format that PuTTY can use. PuTTYgen displays a warning about saving the key without a passphrase. Choose **Yes**.



Launching an EC2 Instance : Instance SSH Login (Win.)

- Specify the same name for the key that you used for the key pair (for example, my-key-pair). PuTTY automatically adds the .ppk file extension.

1. Now, Open Putty
2. Click on **SSH - > Auth** on left panel and browse for the ppk file



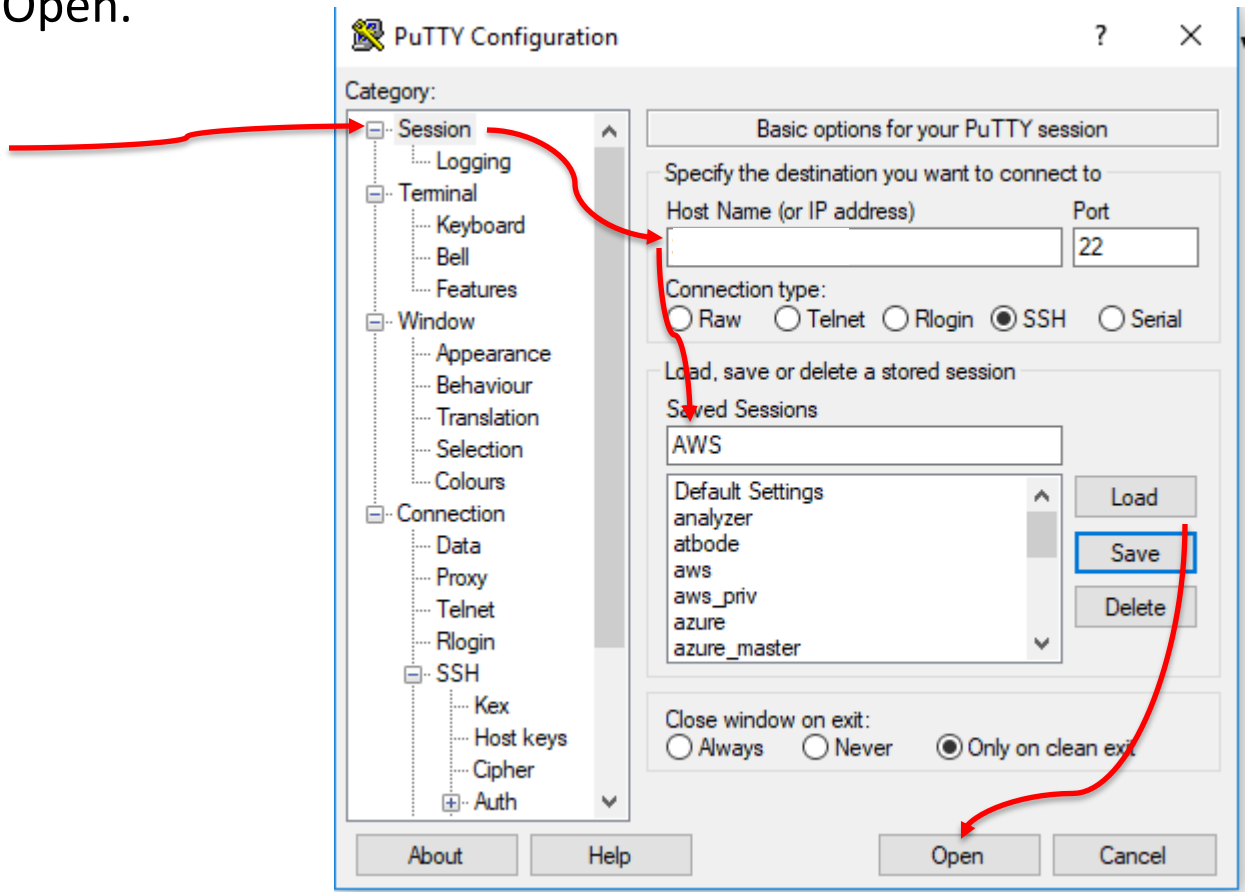
Launching an EC2 Instance : Instance SSH Login (Win.)

3.Click on Session

4.Add the **hostname** which is the **public IP** of your instance

5.Write a name to save the configuration for future use, click save

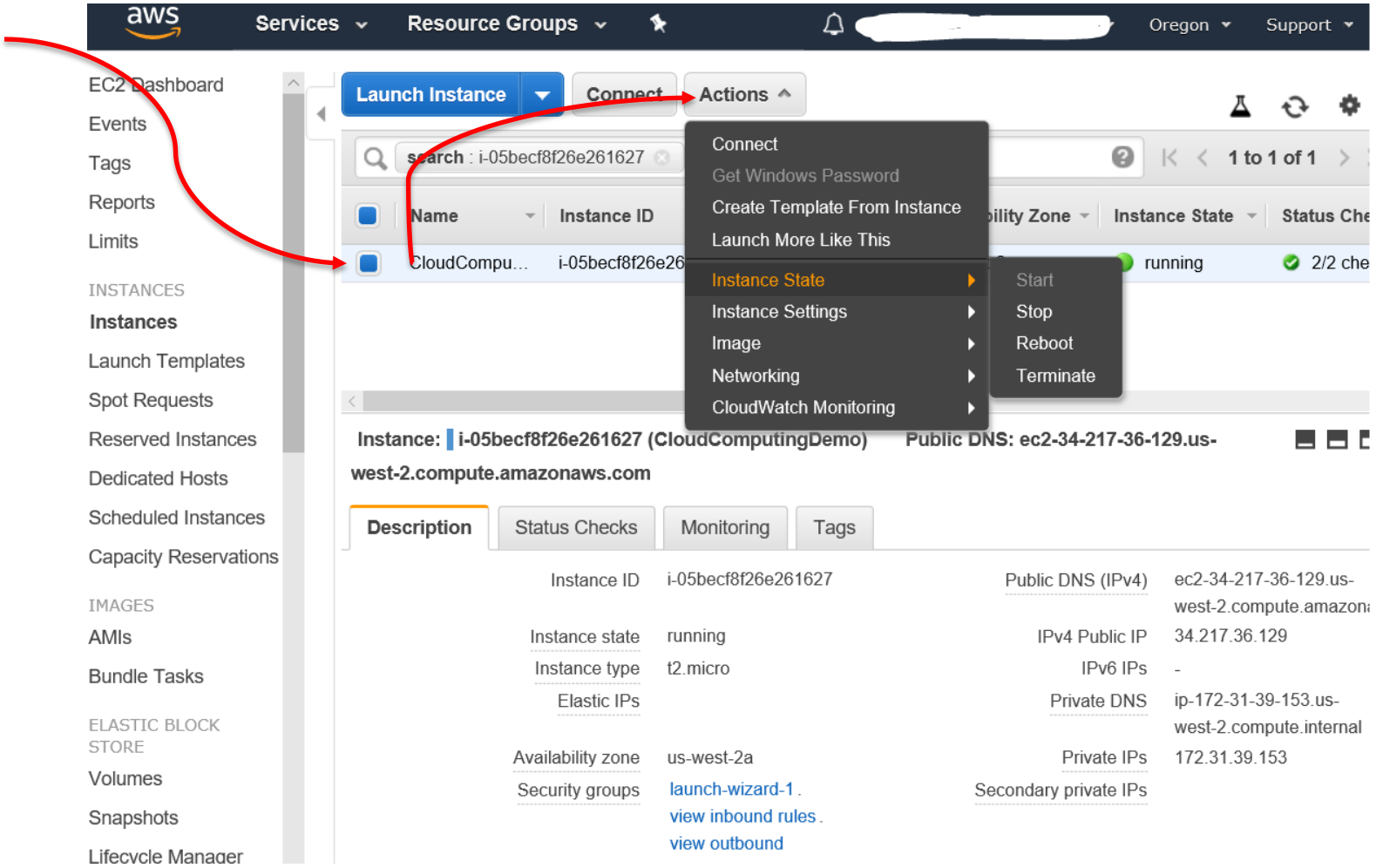
6.Click Open.



7.Use username
ubuntu

Instance Stop/Reboot/Termination

You can either stop or reboot or terminate the instance from the EC2 Dashboard



References

[1] http://blog.builtinnode.com/post/a-history-of-node-js

[2] https://nodejs.org/en/about/

[3] https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/

[4] https://www.npmjs.com/about

[5] https://restfulapi.net/rest-architectural-constraints/

[6] https://docs.mongodb.com/manual/introduction/

Short Demo

Thank you for your attention!

Questions?