

CSCI3100 Project Tutorial

4

Server-side Technologies

Yun PENG

csci3100@cse.cuhk.edu.hk

Department of Computer Science and
Engineering
The Chinese University of Hong Kong

Outline

- Introduction to Web Development
 - Socket
 - Protocols (TCP/UDP/HTTP)
 - Express Framework
 - Database in Node.js
 - Node.js IDE: WebStorm
 - Django, Node.js & Flask

Outline

- Introduction to AWS
 - Amazon and AWS
 - Create Micro Instance on EC2
 - Basic shell commands
- Database Basics
 - Relational Database (MySQL)
 - NoSQL Database (MongoDB)
- Reference

Access to a website

Chrome Menu->View->Developer->Developer Tools

Firefox Menu->Web Developer->Toggle Tools

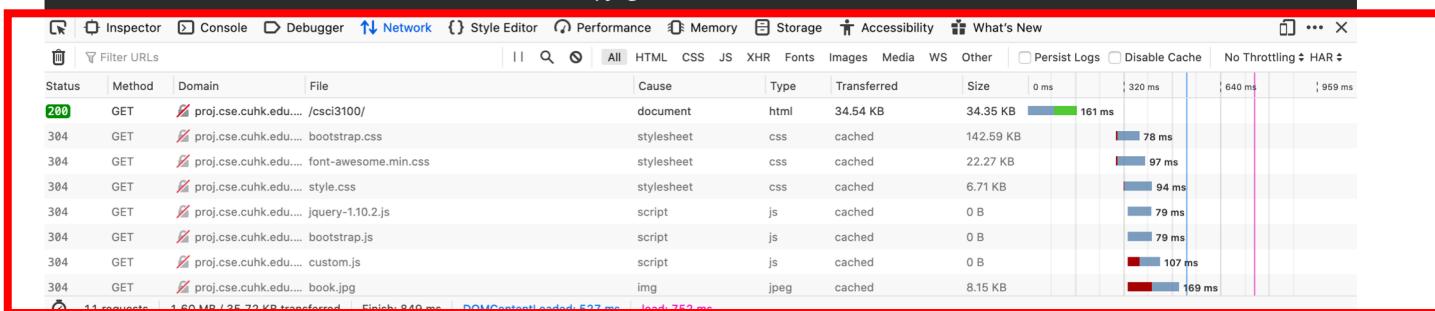


☆ NEWS

18-Feb: Assignment 2 has been released. The due is **11:00am, 2nd Mar, 2020 (Monday)**. Please submit your answers together with the VeriGuide receipt to Blackboard.

14-Feb: We will conduct ZOOM online lectures and tutorials. Please refers to the links in the following *Course Time* table for the upcoming lectures and tutorials.

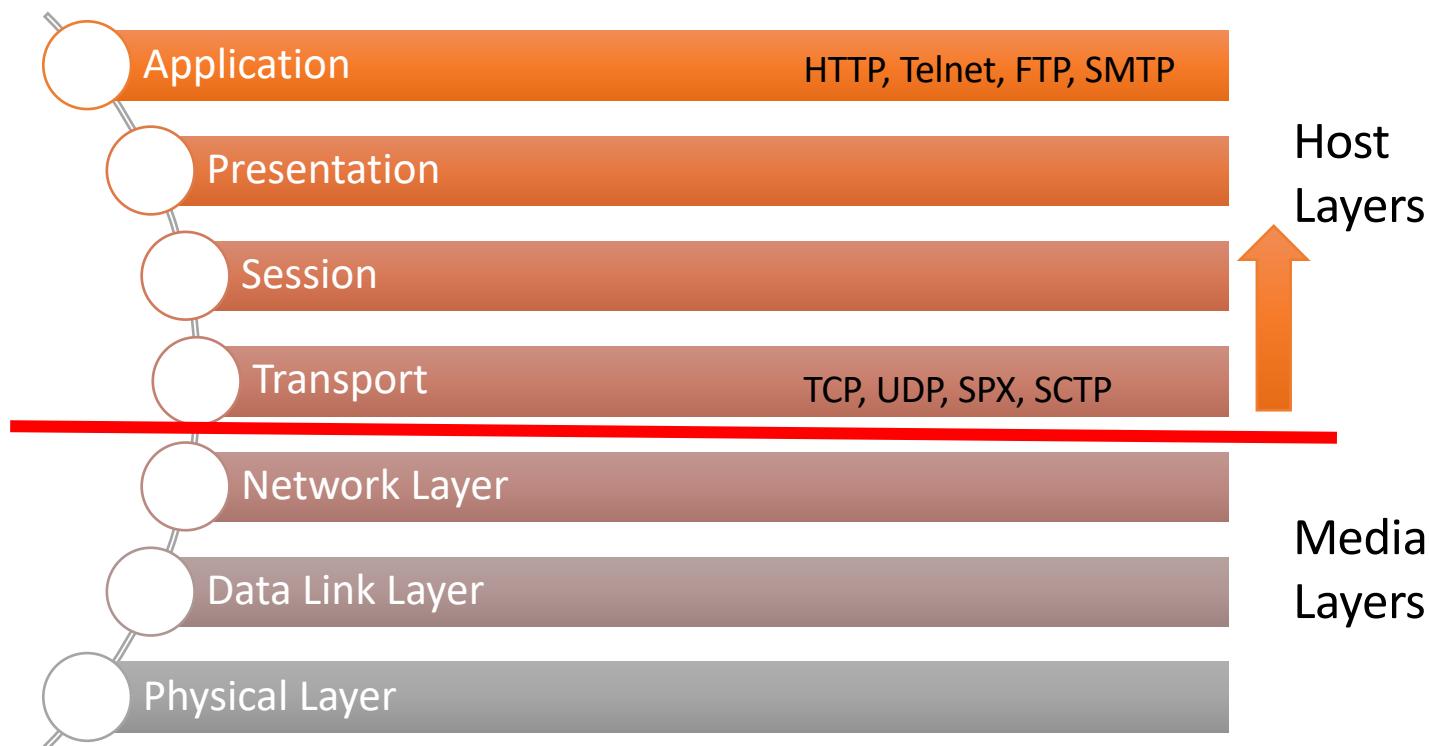
Copyright © 2020 CSE@CUHK



How browsers work

- What happened when you visit a webpage?
 - The browser *communicates* with a **process** in the remote server
 - The remote process *send* data (HTML, etc.) back
 - The browser *renders* (display) the data
- In detail
 - [How Browsers Work: Behind the scenes of modern web browsers](#)

OSI (Open Systems Interconnection) Model



Motivation of Using Node.js

- Save time to learn new languages
 - Node.js is a **JavaScript** runtime built on Chrome's V8 JavaScript engine
- Node.js is very efficient
 - event-driven, non-blocking I/O model
- Learn new paradigm to catch up the tendency of Web development
- Using other frameworks are also accepted
 - Will not affect your grade

Installation

- Follow: <https://nodejs.org>

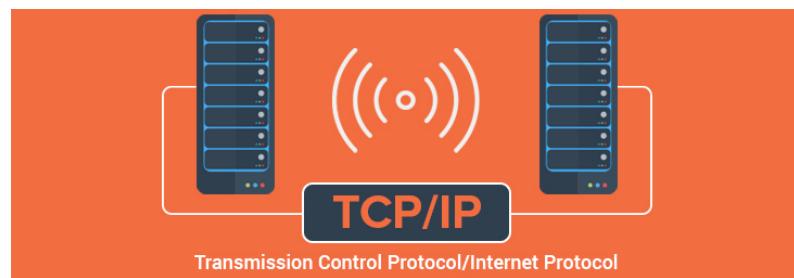
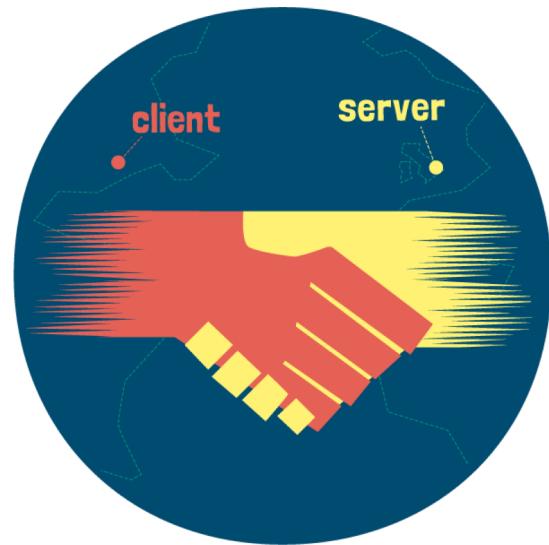
- After installation:

- Test the version:

```
yangty@Tianyi-MBP ~ ➤ node -v  
v13.6.0  
yangty@Tianyi-MBP ~ ➤ █
```

- Command line interface:

```
yangty@Tianyi-MBP ~ ➤ node  
Welcome to Node.js v13.6.0.  
Type ".help" for more information.  
> var a = 1  
undefined  
> a  
1
```



Transmission Control Protocol (TCP)

```
var net = require("net");           // Get "Net" Module

var server = net.createServer({
  allowHalfOpen: false
}, function(socket) {
  // handle connection
  socket.end("Hello and Goodbye!\n");
});

server.listen(8000, "127.0.0.1");    // Register the address and port
```

Demo of TCP Connection

```
$ telnet localhost 8000
```

```
Trying 127.0.0.1...
```

```
Connected to localhost.
```

```
Escape character is '^]'.
```

Hello and Goodbye!

Connection closed by foreign host.

p.s: enabling 'telnet' in win10 with cmd (administrator):

```
$ dism /online /Enable-Feature /FeatureName:TelnetClient
```

```
https://www.rootusers.com/how-to-enable-the-telnet-client-in-windows-10/
```

For mac, it should support telnet by default, but if your macOs is high Sierra(10.13), you should refer to: <https://medium.com/ayuth/bring-telnet-back-on-macos-high-sierra-11de98de1544>

User Datagram Protocol (UDP)-Server

```
var dgram = require("dgram"); // Get "dgram" module
var server = dgram.createSocket("udp4", function(msg,
rinfo) {
    console.log("received " + rinfo.size + " bytes");
    console.log("from " + rinfo.address + ":" + rinfo.port);
    console.log("message is: " + msg.toString());
});

server.bind({address: "localhost", port: 8012}, function() {
    console.log("bound to ");
    console.log(server.address());
});
```

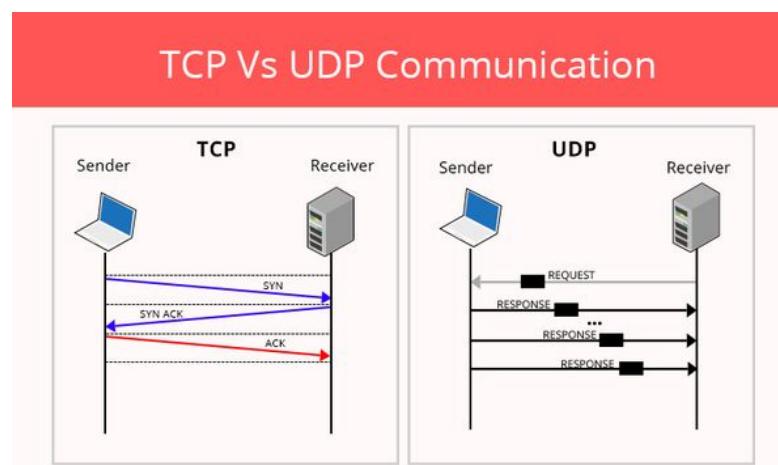
Sending Data via UDP-Client

```
var dgram = require("dgram");
var client = dgram.createSocket("udp4");
var message = new Buffer("Hello UDP");

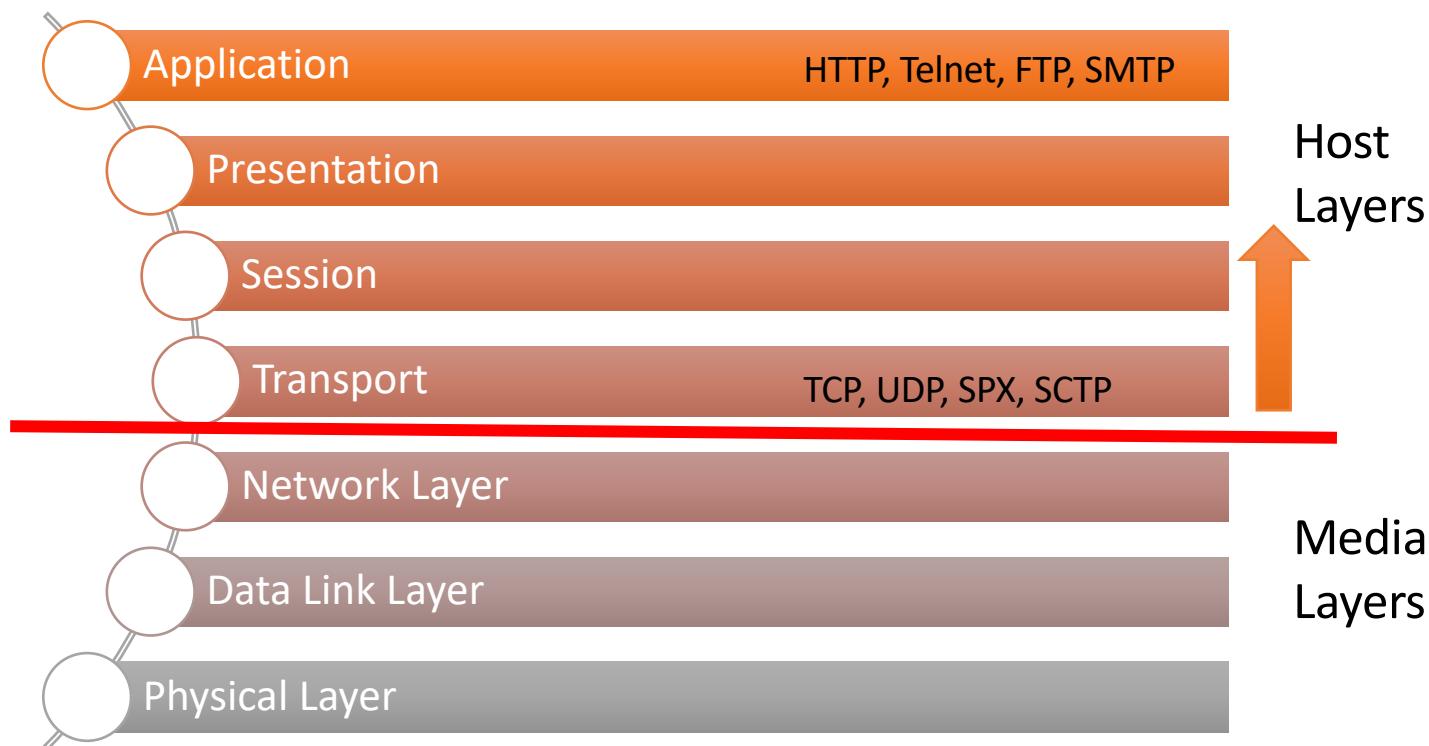
client.send(message, 0, message.length, 8012,
"127.0.0.1", function(error, bytes) {
  if (error) {
    console.error("An error occurred while sending");
  } else {
    console.log("Successfully sent " + bytes +
bytes");
  }
  client.close();
});
```

TCP vs UDP

TCP	UDP
Reliable	Unreliable
Numbers each data packet	Does not keep data packet in order
Confirms receipt of the data packet	Does not confirm data packet receipt
Include error-check capabilities to prevent data corruption	Maximize performance
Consumes more network bandwidth	Consumes less bandwidth
Overall slower processing	Faster processing
three-way handshake	no



OSI (Open Systems Interconnection) Model



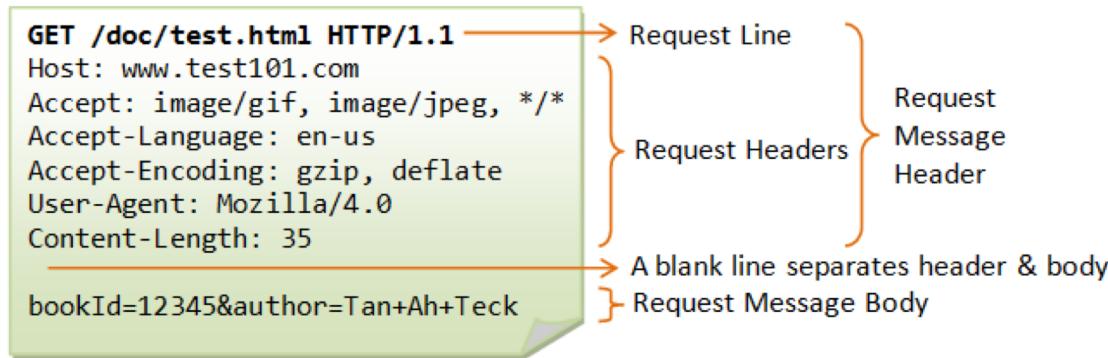
HyperText Transfer Protocol (HTTP)

- Higher layer than TCP and UDP

```
var http = require("http");
var server = http.createServer(function(request,
response) {
  response.write("Hello HTTP!");
  response.end();
});
server.listen(8000);
```

Open 127.0.0.1:8000 in browser

HTTP Request & Request Method



GET	A GET request should not alter the state of the server and is essentially a read operation.
POST	POST requests are the submission of HTML forms and the addition of data to a database.
HEAD	Retrieves the same data as an equivalent GET request, except the response body should be omitted.
PUT	PUTs are used to update existing resources on the server.
DELETE	Used to delete a resource from a server.
CONNECT	Used to create a tunnel through a proxy server.

Get the Info of Incoming Connection

```
var http = require("http");
var server = http.createServer(function(request,
response) {
    var requestLine = request.method + " " + request.url +
        " HTTP/" + request.httpVersion;
    console.log(requestLine);
    response.end();
});
server.listen(8000);
```

Request Header

Accept	Specifies the Content-Types that the client is willing to accept for this request.
Accept-Encoding	Provides a list of acceptable encodings
Cookie	Small pieces of data that the server stores on the client
Content-Length	The length of the request body in octets.
Host	The domain and port of the server
User-Agent	A string identifying the type of client

```
var http = require("http");
http.createServer(function(request, response) {
  console.log(request.headers);
  response.end();
}).listen(8000);
```

Response Code

200	OK	Indicates that the HTTP request was handled successfully
201	Created	Indicates that the request has been fulfilled, and a new resource has been created on the server
301	Moved Permanently	The requested resource has permanently moved to a new URL
400	Bad Request	Indicates that the request was malformed and could not be understood.
404	Not Found	The server could not locate the requested URL
500	Internal Server Error	The server encountered an error while attempting to fulfill the request

Demo of Response Code

```
var http = require("http");
http.createServer(function(request, response) {
  if (request.url === "/foo") {
    response.end("Hello HTTP");
  } else {
    response.statusCode = 404;
    response.end();
  }
}).listen(8000);
```

Response Header

Cache-Control	Specifies whether a resource can be cached
Content-Type	Specifies the MIME type of the response body
WWW-Authenticate	If an authentication scheme is implemented for a given resource, this header is used to identify the scheme
Content-Encoding	Specifies the encoding used on the data

```
var http = require("http");
var server = http.createServer(function(request, response) {
  response.setHeader("Content-Type", "text/html");
  response.write("Hello <strong>HTTP</strong>!");
  response.end();
});
server.listen(8000);
```

Multiple Resources

```
var http = require("http");
http.createServer(function(request, response) {
  if (request.url === "/" && request.method === "GET") {
    response.writeHead(200, {
      "Content-Type": "text/html"
    });
    response.end("Hello <strong>home page</strong>");
  } else if (request.url === "/bar" && request.method === "GET") {
    response.writeHead(200, {
      "Content-Type": "text/html"
    });
    response.end("Hello <strong>bar</strong>");
  } else {
    response.writeHead(404, {
      "Content-Type": "text/html"
    });
    response.end("404 Not Found");
  }
}).listen(8000);
```

Terrible?



express

A Simpler Solution

```
var express = require("express");
var http = require("http");
var app = express();
app.get("/", function(req, res, next) {
    res.send("Hello <strong>home page</strong>");
});
app.get("/bar", function(req, res, next) {
    res.send("Hello <strong>bar</strong>");
});
http.createServer(app).listen(8000);
```

Note: Install Express Framework using “npm install express”

Route Parameter

```
var express = require("express");
var http = require("http");
var app = express();
app.get(/\products\/([^\\/]+)\?$/, function(req, res,
next) {
  res.send("Requested " + req.params[0]);
});
http.createServer(app).listen(8000);
```

PS: must use a JS regular expression object instead if string by
wrapping your regular expression in slashes (/) instead of quotes
("")

<https://www.kevinleary.net/regex-route-express/>

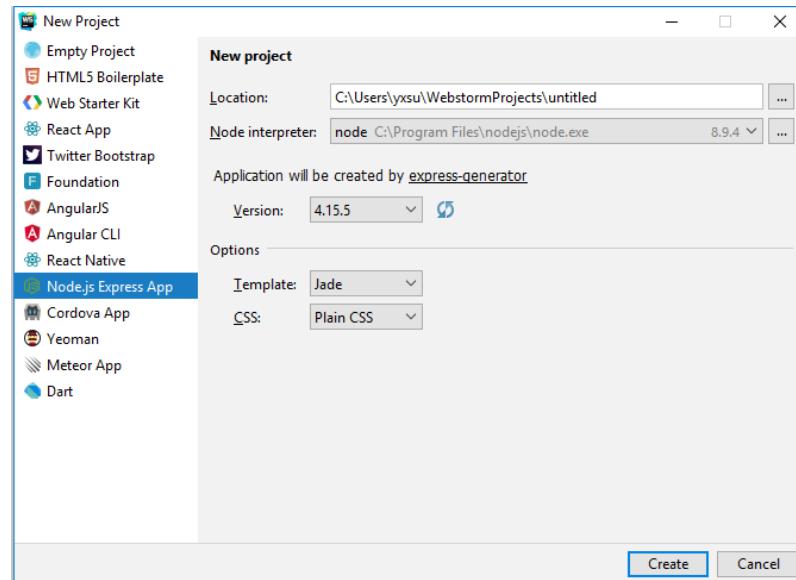
Route with Named Parameter

```
var express = require("express");
var http = require("http");
var app = express();
app.get("/products/:productId", function(req, res, next)
{
  res.send("Requested " + req.params.productId);
});
http.createServer(app).listen(8000);
```

Creating an Express Framework

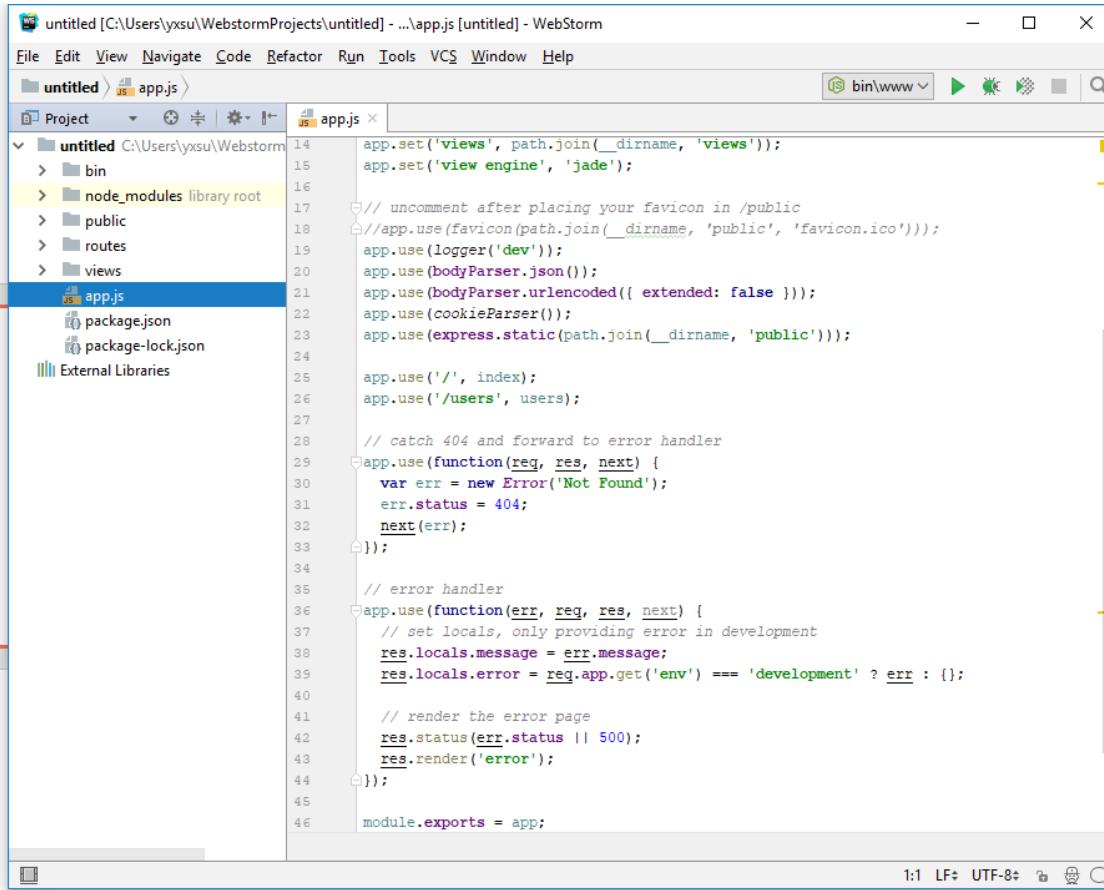
- Easiest way: WebStorm (demo)
- Or:
 - Installation:
 - `$npm install -g express`
 - Create:
 - `$ express testapp`
 - Install dependency:
 - `cd testapp && npm install`

WebStorm (IDE)



- <https://www.jetbrains.com/webstorm/>
- Apply for student account:
 - <https://www.jetbrains.com/student/>

WebStorm (IDE)



The screenshot shows the WebStorm IDE interface. The title bar reads "untitled [C:\Users\yxsu\WebstormProjects\untitled] - ...app.js [untitled] - WebStorm". The menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The toolbar has icons for bin\www, run, stop, and search. The left sidebar shows a project structure with a folder named "untitled" containing "bin", "node_modules" (library root), "public", "routes", "views", and "app.js". Below these are "package.json", "package-lock.json", and "External Libraries". The main editor window displays the contents of "app.js".

```
14 app.set('views', path.join(__dirname, 'views'));
15 app.set('view engine', 'jade');
16
17 // uncomment after placing your favicon in /public
18 //app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
19 app.use(logger('dev'));
20 app.use(bodyParser.json());
21 app.use(bodyParser.urlencoded({ extended: false }));
22 app.use(cookieParser());
23 app.use(express.static(path.join(__dirname, 'public')));
24
25 app.use('/', index);
26 app.use('/users', users);
27
28 // catch 404 and forward to error handler
29 app.use(function(req, res, next) {
30   var err = new Error('Not Found');
31   err.status = 404;
32   next(err);
33 });
34
35 // error handler
36 app.use(function(err, req, res, next) {
37   // set locals, only providing error in development
38   res.locals.message = err.message;
39   res.locals.error = req.app.get('env') === 'development' ? err : {};
40
41   // render the error page
42   res.status(err.status || 500);
43   res.render('error');
44 });
45
46 module.exports = app;
```

NodeJS, Django & Flask



JavaScript



Python

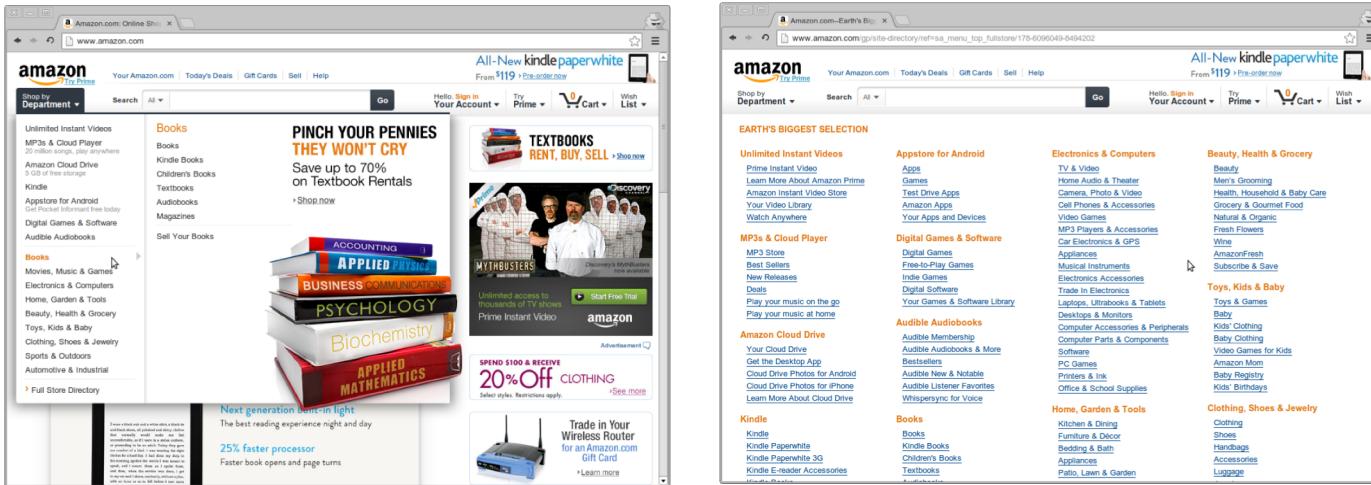


Python

	Node.js	Django
Pros	<ul style="list-style-type: none">• Availability of great libraries.• High performance.• Huge user community.• Handles concurrent requests easily.	<ul style="list-style-type: none">• Works fine with relational databases.• Easy to learn and very scalable.• Speedy development process.• Huge user community and great documentation.
Cons	<ul style="list-style-type: none">○ Asynchronous programming is difficult○ Not great with CPU intensive apps (single thread)○ Callbacks cause many nested callbacks.	<ul style="list-style-type: none">○ Not great for small-scale apps.○ A full understanding of the framework is needed.

Amazon

- Amazon is one of the major e-commerce player in the US
 - It starts as an online bookstore
 - It sells various kind of things nowadays



Amazon

- How a traditional e-commerce company end up selling their computing power?
 - Traffic to Amazon erupts significantly on Christmas/Black Friday (Think about Taobao on Nov. 11)
 - Amazon must build their computational infrastructure powerful enough to cope with the peak traffic
 - Most of the computing power are wasted during other time

AWS



- AWS stands for Amazon Web Services
 - AWS is one of the major cloud computing environment
 - Big names in startup are powered by AWS
 - Reddit, Pinterest, Flipboard, Foursquare
 - Bigger names use their own cloud
 - Microsoft, Facebook, Google, Alibaba, Tencent
 - What exactly is AWS
 - <http://www.youtube.com/watch?v=jOhbTAU4OPI>



AWS EC2

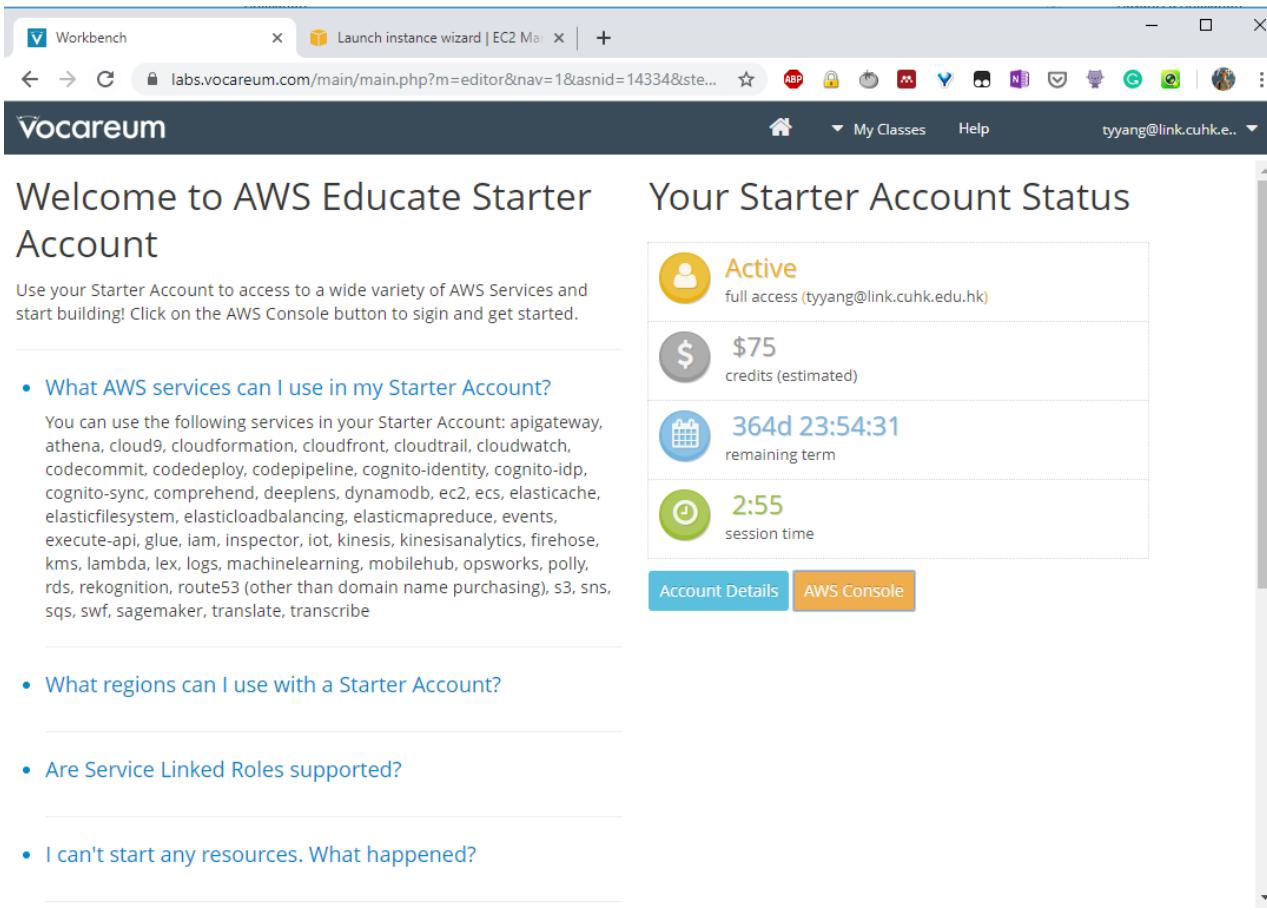
- Amazon EC2 stands for Elastic Computing Cloud
 - For this tutorial, think of it as a single virtual machine
- You are encouraged to deploy your project backend on EC2 to allow public access

Create AWS Account

- Create an account on AWS (If you don't have)
 - AWS Educate: **no credit card needed**
 - Students at AWS Educate member institutions will receive up to US\$75 of AWS credit per year in their AWS Educate Starter Account
 - https://www.awseducate.com/registration#APP_TYPE
 - AWS Personal: credit card needed.
 - New user are eligible for free access to micro instance for 12 months, 750 hours per month

Create Micro Instance on EC2

1. Login to your account and go to the AWS console



The screenshot shows a web browser window with the title "Vocaboreum". The URL in the address bar is "labs.vocareum.com/main/main.php?m=editor&nav=1&asnid=14334&ste...". The page displays the "Welcome to AWS Educate Starter Account" and "Your Starter Account Status" sections.

Welcome to AWS Educate Starter Account

Use your Starter Account to access to a wide variety of AWS Services and start building! Click on the AWS Console button to sign in and get started.

- [What AWS services can I use in my Starter Account?](#)
- [What regions can I use with a Starter Account?](#)
- [Are Service Linked Roles supported?](#)
- [I can't start any resources. What happened?](#)

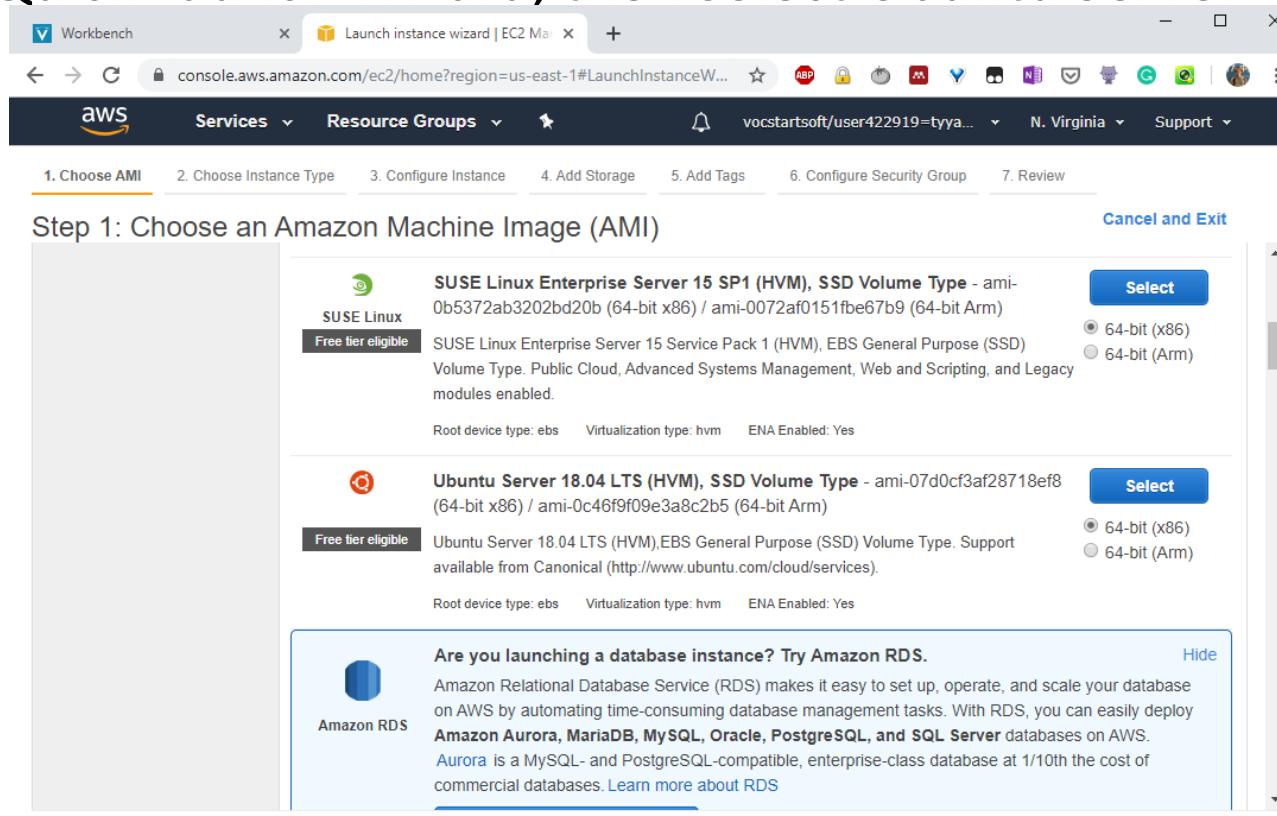
Your Starter Account Status

	Active	full access (tyyang@link.cuhk.edu.hk)
	\$75	credits (estimated)
	364d 23:54:31	remaining term
	2:55	session time

[Account Details](#) [AWS Console](#)

Create Micro Instance on EC2

2. Click the “Launch a virtual machine” to create an EC2 Instance
Select Quick Launch Wizard, then select Ubuntu Server 18.04



Create Micro Instance on EC2

For step 2 – 7 : use default option, then click launch

The screenshot shows the AWS EC2 instance creation wizard at Step 2: Choose an Instance Type. The navigation bar at the top includes links for Choose AMI, Choose Instance Type (which is underlined in orange), Configure Instance, Add Storage, Add Tags, Configure Security Group, and 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)

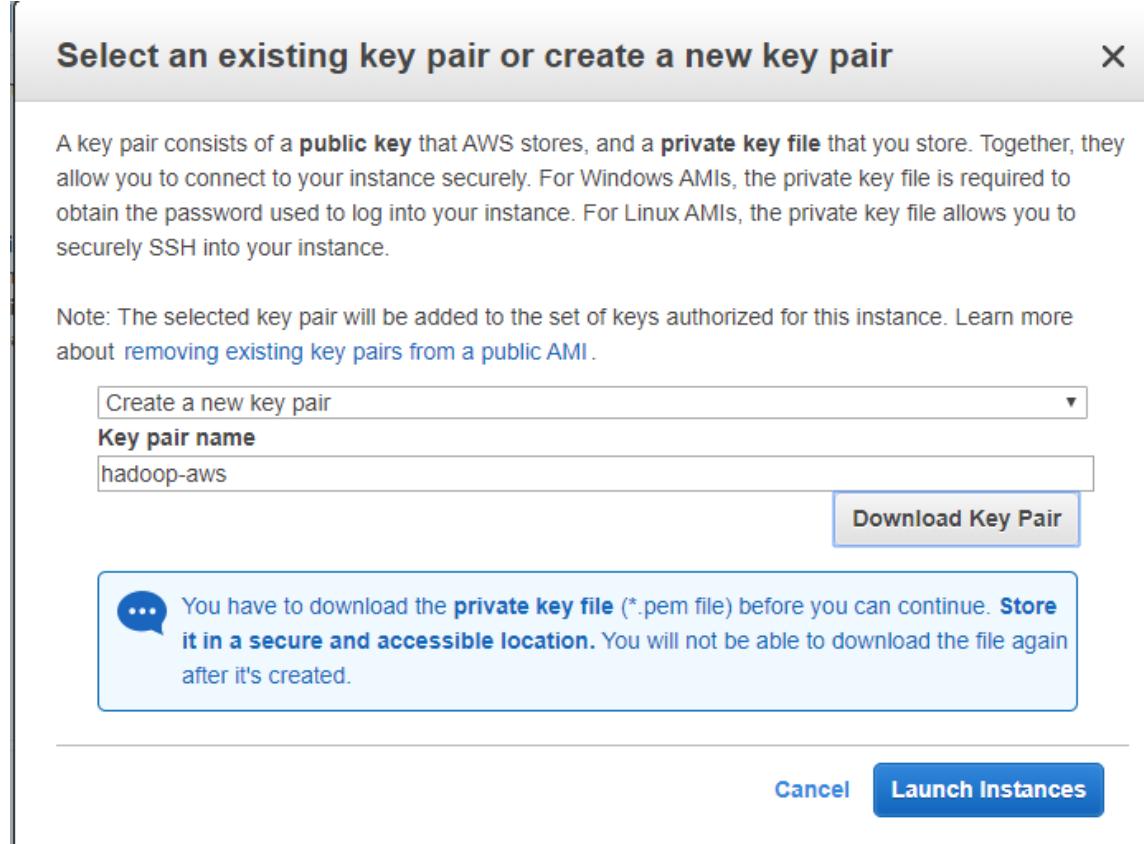
	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
<input type="checkbox"/>	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.xlarge	4	16	FRS only	-	Moderate	Yes

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

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Create Micro Instance on EC2

3. Create a new key pair (or select existing one) and **download it**.
Click Launch Instances and wait a few minutes until it is running.



Create Micro Instance on EC2

4. In your EC2 management console you will see the state of your VM

The screenshot shows the AWS EC2 Management Console interface. The left sidebar navigation bar includes links for EC2 Dashboard, Events, Tags, Reports, Limits, Instances (selected), Launch Templates, Spot Requests, Reserved Instances, Dedicated Hosts, Scheduled Instances, Capacity Reservations, Images (AMIs), and Elastic Block Store (Volumes, Snapshots, Lifecycle Manager). The main content area has tabs for 'Launch Instance', 'Connect', and 'Actions'. A search bar at the top of the main content area says 'Filter by tags and attributes or search by keyword'. Below it is a table with columns: Name, Instance ID, Instance Type, Availability Zone, Instance State, Status Checks, and Alarm. One row is visible: Name i-0bf9f3b15f49653cf, Instance ID i-0bf9f3b15f49653cf, Instance Type t2.micro, Availability Zone us-east-1b, Instance State running, Status Checks 2/2 checks ... (green checkmark), and Alarm None. At the bottom, a detailed view for the instance i-0bf9f3b15f49653cf shows Public DNS: ec2-3-95-185-118.compute-1.amazonaws.com, Description tab selected, Instance ID i-0bf9f3b15f49653cf, Public DNS (IPv4) ec2-3-95-185-118.compute-1.amazonaws.com, Instance state running, and IPv4 Public IP 3.95.185.118.

Connect to the instance

- First, find the public DNS of your instance

The screenshot shows the AWS EC2 Management console interface. On the left, there's a navigation sidebar with sections like EC2 Dashboard, Events, Tags, Reports, Limits, INSTANCES (with Instances selected), Launch Templates, Spot Requests, Reserved Instances, Dedicated Hosts, Scheduled Instances, Capacity Reservations, IMAGES (with AMIs selected), and ELASTIC BLOCK STORE (with Volumes selected). The main content area has tabs for Launch Instance, Connect, and Actions. Below that is a search bar and a table with columns for Name, Instance ID, Instance Type, Availability Zone, Instance State, and Status Checks. One row is visible, showing an instance with Instance ID i-0bf9f3b15f49653cf, Instance Type t2.micro, Availability Zone us-east-1b, Instance State running, and Status Checks 2/2 checked. At the bottom, there's a detailed view for the selected instance, showing fields for Description, Status Checks, Monitoring, and Tags. The Public DNS (IPv4) field is highlighted with an orange rectangle and contains the value ec2-3-95-185-118.compute-1.amazonaws.com. Below this, other details like Instance ID, Instance state, Instance type, and Elastic IPs are listed.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
i-0bf9f3b15f49653cf	t2.micro	us-east-1b	running	2/2 checked	

Instance: i-0bf9f3b15f49653cf Public DNS: ec2-3-95-185-118.compute-1.amazonaws.com

Description	Status Checks	Monitoring	Tags
Instance ID: i-0bf9f3b15f49653cf	Public DNS (IPv4): ec2-3-95-185-118.compute-1.amazonaws.com	IPv4 Public IP: 3.95.185.118	IPv6 IPs: -
Instance state: running	IPv6 Public IP: -	Private DNS: ip-172-31-83-	Elastic IPs: -
Instance type: t2.micro			
Elastic IPs: -			

Feedback English (US) © 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Connect to the instance

- If you are in Linux or Windows 10, you can connect to your instance with ssh

```
yangty@Terrill-MacBook ~ cd Downloads/aws-tut
yangty@Terrill-MacBook ~/Downloads/aws-tut ls
hadoop-aws.pem
yangty@Terrill-MacBook ~/Downloads/aws-tut ssh -i hadoop-aws.pem ubuntu@ec2-3-95-185-118.compute-1.amazonaws.com
```

- The key must be only readable to you
- <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html>

Connect to the instance

- If you are using Windows 8 or older version, you need an ssh client, putty is a great free ssh client.
- Follow the guide:
 - <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html>

Basic Linux Commands

- ls : list directory
- cd : change directory
- mkdir : create directory
- rm : remove file
- ps : show process status
- ssh: OpenSSH SSH client (remote login)
- wget: network downloader

Database Basics

- MySQL
- MONGODB

Installation and Connection of MySQL(Windows)

Download link:

<https://www.mysql.com/>

Switch to the downloaded folder:

```
cd C:\User\CSCI3100\Download\mysql-8.0.11\bin
```

Initialization(You will get a temporary password):

```
mysqld --initialize --console
```

Installation(Windows):

```
mysqld install
```

Start MySQL:

```
net start mysql
```

Connect MySQL:

```
mysql -h hostname -u username -p
```

Database Operation in MySQL

Create Database:

```
mysql> CREATE DATABASE CSCI3100;
```

Delete Database:

```
mysql> DROP DATABASE CSCI3100;
```

Select Database:

```
mysql> USE CSCI3100;
```

Table Operation in MySQL

Create Table:

```
mysql> CREATE TABLE student(
-> student_id INT NOT NULL,
-> name VARCHAR(100) NOT NULL,
-> gender VARCHAR(40) NOT NULL,
-> grade INT,
-> PRIMARY KEY (student_id )
-> )ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Delete Table:

```
mysql> DROP TABLE student;
```

Insert and Query Data in MySQL

Insert Data:

```
mysql> INSERT INTO student  
-> (student_id, name, gender, grade)  
-> VALUES  
-> (1155139458, "MIKE", "MALE", 98);
```

Query Data:

```
mysql> SELECT * from student;  
+-----+-----+-----+-----+  
| student_id | name | gender | grade |  
+-----+-----+-----+-----+  
| 1155139458 | MIKE | MALE | 98 |  
| 1155139462 | BOB | MALE | 89 |  
| 1155139478 | JANE | FEMALE | 89 |  
+-----+-----+-----+-----+  
3 rows in set (0.01 sec)
```

Common Clauses in MySQL

WHERE:

```
mysql> SELECT * from student WHERE grade<90;
+-----+-----+-----+-----+
| student_id | name   | gender | grade |
+-----+-----+-----+-----+
| 1155139462 | BOB    | MALE   | 89    |
| 1155139478 | JANE   | FEMALE | 89    |
+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

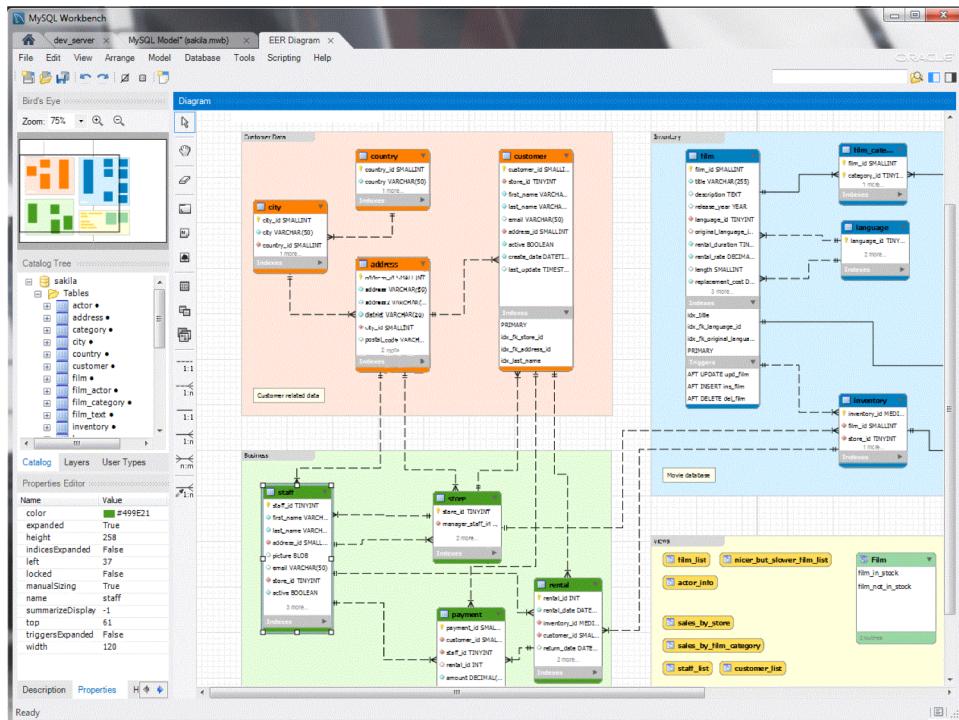
UPDATE:

```
mysql> UPDATE student SET grade=85 WHERE student_id= 1155139458;
```

DELETE:

```
mysql> DELETE FROM student WHERE student_id= 1155139458;
```

MySQL WorkBench (GUI for MySQL)



The screenshot shows the MySQL WorkBench interface with the following components:

- Query Editor:** Contains the following SQL query:

```

SELECT `actor`.`actor_id`,
       `actor`.`first_name`,
       `actor`.`last_name`,
       `actor`.`last_update`
  FROM `sakila`.`actor`;
    
```

The results pane shows the output for the first 10 rows:

actor_id	first_name	last_name	last_update
1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies	2006-02-15 12:45:49
2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China	2006-02-15 12:45:49
3	ADAPTATION HOLES	A Returning Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Ballo Factory	2006-02-15 12:45:49
4	AFFAIR PREJUDICE	A Fanciful Documentary of a Fisher And a Lumberjack who must Chase a Monkey in A Shark Tank	2006-02-15 12:45:49
5	AFRICAN EGG	A Fast-Paced Documentary of a Chef And a Dentist who must Pursue a Forensic Psychologist in	2006-02-15 12:45:49
6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China	2006-02-15 12:45:49
7	AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who must Discover a Butler in A Jet Boat	2006-02-15 12:45:49
8	AIRPORT POLLOCK	A Epic Tale of a Moose And a Girl who must Confront a Monkey in Ancient India	2006-02-15 12:45:49
9	ALABAMA DEVIL	A Thoughtful Panorama of a Database Administrator And a Mad Scientist who must Outgun a Mad So	2006-02-15 12:45:49

SQL Additions: Shows the full SELECT query with annotations explaining clauses like DISTINCT, ORDER BY, and GROUP BY.

SQL vs NoSQL Database

- SQL databases are known as relational databases, and have a **table-based** data structure, with a **strict, predefined schema** required.
- NoSQL databases, or non-relational databases, can be document based, graph databases, key-value pairs, or wide-column stores. NoSQL databases **don't require any predefined schema**, allowing you to work more freely with “unstructured data.”

SQL vs NoSQL Database

- Relational databases are vertically scalable, but usually more expensive, whereas the horizontal scaling nature of NoSQL databases is more cost efficient.
- Since NoSQL has looser restrictions, it can even be processed in memory, e.g. Redis

MongoDB

Advantage:

- Weak consistency, ensure the access speed
- Easier to retrieve data because its storage method
- Rich third party support
- Good performance

What makes it different from MySQL?

MongoDB	RDBMS (E.g. MySQL)
Document-oriented and non-relational database	Relational database
Document based	Row based
Field based	Column based
Collection based and key–value pair	Table based
Gives JavaScript client for querying	Doesn't give JavaScript for querying
Has dynamic schema and ideal for hierarchical data storage	Has predefined schema and not good for hierarchical data storage
100 times faster and horizontally scalable through sharding	By increasing RAM, vertical scaling can happen

Installation and connection of MongoDB(Windows)

Download Link:

<https://www.mongodb.com/>

Create the data folder:

```
c:>mkdir data  
c:>cd data  
c:\data>mkdir db
```

Run MongoDB server:

```
C:\mongodb\bin\mongod --dbpath c:\data\db
```

Connect to MongoDB:

```
C:\mongodb\bin\mongo.exe
```

Start MongoDB service:

```
net start MongoDB
```

Connection URL to MongoDB service:

```
mongodb://admin:123456@localhost/
```

Conceptual Analogy between MySQL and MongoDB

MySQL	MongoDB
database	database
table	collection
row	document
column	field
index	index
table joins	
primary key	primary key

Database Operation in MongoDB

Create or select the database:

```
>use CSCI3100
```

Delete current database:

```
>db.dropDatabase()
```

Collection Operation in MongoDB

Create the collection:

```
db.createCollection(student, options)
```

Option	Type	Description
capped	bool	If the size of the collection is fixed
autoIndexId	bool	If create the auto index
size	int	The size of the collection
max	int	The max size of the document in the collection

Delete the collection:

```
db.student.drop()
```

Document Operation in MongoDB

Insert the document:

```
>db.student.insert({student_id: 1155139458,  
    name: 'MIKE',  
    gender: 'MALE',  
    grade: 98  
})
```

Update the document:

```
>db.student.update({'student_id': 1155139458}, {$set: {'grade' : 85}})
```

Delete the document:

```
>db.student.remove({'student_id': 1155139458})
```

Query the document:

```
> db. student.find({'student_id': 1155139458}).pretty()
```

Robo 3T (GUI for MongoDB)

The screenshot shows the Robo 3T application window. On the left is a sidebar with a tree view of databases and collections:

- mongo_atlas_3.4 (4)
 - Replica Set (3 nodes)
 - System
 - db1
 - db2
 - Collections (2)
 - Accounts
 - Bills**

The "Bills" collection is selected and highlighted with a blue border.

In the main pane, there is a query bar at the top:

```
1 db.getCollection('Bills').find({})
```

Below the query bar is a results table titled "Bills". The table has three columns: "Key", "Value", and "Type". The "Value" column contains JSON documents, and the "Type" column indicates the type of each field. The results show 8 documents, each with fields: _id, Jan, Feb, Mar, OpLimit.

Key	Value	Type
(1) ObjectId("593972651efd2d98c2...")	{ 2 fields }	Object
_id	ObjectId("593972651efd2d98c...")	ObjectId
Jan	9999999.49999999	Decimal128
(2) ObjectId("593972661efd2d98c2...")	{ 2 fields }	Object
_id	ObjectId("593972661efd2d98c...")	ObjectId
Feb	1000000.55	Decimal128
(3) ObjectId("593972661efd2d98c2...")	{ 2 fields }	Object
_id	ObjectId("593972661efd2d98c...")	ObjectId
Mar	9000000	Decimal128
(4) ObjectId("593972661efd2d98c2...")	{ 2 fields }	Object
(5) ObjectId("593972661efd2d98c2...")	{ 2 fields }	Object
(6) ObjectId("593972671efd2d98c2...")	{ 2 fields }	Object
(7) ObjectId("593972671efd2d98c2...")	{ 2 fields }	Object
(8) ObjectId("593972671efd2d98c2...")	{ 2 fields }	Object
_id	ObjectId("593972671efd2d98c...")	ObjectId
OpLimit	1.4E+100	Decimal128

Two arrows point from the bottom of the image to specific elements in the table:

- An arrow points to the first row under the "Value" column, labeled "New icon".
- An arrow points to the last row under the "Type" column, labeled "New type name".

Database Basics

- DATABASE Operation in NODE.JS

Connect to MySQL with node.js

```
var mysql = require('mysql');
var connection = mysql.createConnection({
  host: 'localhost',
  user: 'username',
  password: 'password',
  database: 'CSCI3100'
});
connection.connect();
```

Query Data in MySQL with node.js

```
// connect to mysql first
var sql = 'SELECT * FROM student';
connection.query(sql, function (err, result) {
  if (err) {
    console.log('[SELECT ERROR] - ', err.message);
    return;
  }
  console.log(result);
});
connection.end();
```

Insert Data in MySQL with node.js

```
//connect to mysql at first
var addSql = 'INSERT INTO
student(student_id,name,gender,grade) VALUES(?, ?, ?, ?)';
var addSqlParams = [1155139468, 'MIKE', 'MALE', 98];
connection.query(addSql, addSqlParams, function (err, result)
{
    if (err) {
        console.log('[INSERT ERROR] - ', err.message);
        return;
    }
    console.log(result);
});
```

```
connection.end();
```

Update Data in MySQL with node.js

```
//connect to mysql at first
var modSql = 'UPDATE student SET grade = ? WHERE student_id = ?';
var modSqlParams = [85, 1155139458];
connection.query(modSql, modSqlParams, function (err, result) {
  if (err) {
    console.log('[UPDATE ERROR] - ', err.message);
    return;
  }
  console.log(result.affectedRows);
});
```

```
connection.end();
```

Delete Data in MySQL with node.js

```
//connect to mysql at first
var delSql = 'DELETE FROM student where
student_id=1155139458';
connection.query(delSql, function (err, result) {
  if (err) {
    console.log('[DELETE ERROR] - ', err.message);
    return;
  }
  console.log(result.affectedRows);
});
connection.end();
```

Connect to MongoDB with node.js

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://admin:123456@localhost/";

MongoClient.connect(url, { useNewUrlParser: true },
function (err, db) {
  if (err) {
    console.log('[CONNECT ERROR] - ', err.message);
    return;
  }
  console.log("Connection is successful!");
  db.close();
});
```

Query document in MongoDB with node.js

```
//set connection parameter at first
MongoClient.connect(url, { useUnifiedTopology: true }, function (err, db) {
  if (err) {
    console.log('[CONNECT ERROR] - ', err.message);
    return;
  }
  var dbo = db.db("CSCI3100");
  dbo.collection("student").find({ 'student_id': 1155139458
}).toArray(function (err, result) {
  if (err) {
    console.log('[QUERY ERROR] - ', err.message);
    return;
  }
  console.log(result);
  db.close();
});
});
```

Insert document in MongoDB with node.js

```
//set connection parameter at first
MongoClient.connect(url, { useUnifiedTopology: true }, function (err, db) {
    if (err) {
        console.log('[CONNECT ERROR] - ', err.message);
        return;
    }
    var dbo = db.db("CSCI3100");
    var myobj = { student_id: 1155139458, name: 'MIKE', gender: 'MALE', grade: 98 };
    dbo.collection("student").insertOne(myobj, function (err, res) {
        if (err) {
            console.log('[INSERT ERROR] - ', err.message);
            return;
        }
        console.log(res);
        db.close();
    });
});
```

Update document in MongoDB with node.js

```
//set connection parameter at first
MongoClient.connect(url, { useUnifiedTopology: true }, function (err, db) {
  if (err) {
    console.log('[CONNECT ERROR] - ', err.message);
    return;
  }
  var dbo = db.db("CSCI3100");
  var whereStr = { "student_id": 1155139458 };
  var updateStr = { $set: { "grade": 85 } };
  dbo.collection("student").update(whereStr, updateStr, function (err, res) {
    if (err) {
      console.log('[UPDATE ERROR] - ', err.message);
      return;
    }
    console.log(res);
    db.close();
  });
});
```

Delete document in MongoDB with node.js

```
//set connection parameter at first
MongoClient.connect(url, { useUnifiedTopology: true }, function (err, db) {
    if (err) {
        console.log('[CONNECT ERROR] - ', err.message);
        return;
    }
    var dbo = db.db("CSCI3100");
    var whereStr = { "student_id": 1155139488 };
    dbo.collection("student").deleteOne(whereStr, function (err, obj) {
        if (err) {
            console.log('[DELETE ERROR] - ', err.message);
            return;
        }
        console.log(obj);
        db.close();
    });
});
```

References

- [TOP 20 BEST NODEJS FRAMEWORKS FOR DEVELOPERS IN 2020](#)
- [Comparison of web frameworks](#)
- [Node.js vs. Django: Is JavaScript Better Than Python?](#)
- [Node.js Documents](#)
- [The SQL vs NoSQL Difference: MySQL vs MongoDB](#)
- httpbin.org (<https://httpbin.org/>)
- [Node.js Tutorial W3school](#)
- [How Browsers Work: Behind the scenes of modern web browsers](#)

References

- [Linux Command Line Cheat Sheet](#)
- [GNU/Linux Command-Line Tools Summary](#)
- [MySQL Tutorial for Beginners](#)
- [MongoDB Tutorials - The Official MongoDB Document \(Recommended\)](#)
- [MongoDB In 30 Minutes](#)
- [MySQL vs MongoDB? Which database is better?](#)
- [An Enterprise Architect's View - When to Use MongoDB](#)
- [MySQL WorkBench \(GUI for MySQL\)](#)
- [Robo3T \(GUI for MongoDB\)](#)

Thanks!