IERG4210 Web Programming and Security

Course Website: https://course.ie.cuhk.edu.hk/~ierg4210/
Live FB Feedback Group: https://fb.com/groups/ierg4210.2014spring/

Fast and Scalable Web & Database Servers

Lecture 6

Dr. Adonis Fung
phfung@ie.cuhk.edu.hk

Information Engineering, CUHK
Product Security Engineering, Yahoo!

Agenda

- Fast and Scalable Web Servers
  - Quick Introduction to Cloud
  - Architecture and Designs
- DB Storage Servers
  - Quick Intro. to DB Storage
    - Database v.s. Cache
  - Relational Databases (MySQL, SQLite)
    - Structured Query Language (SQL) Language
    - Example Usage thru Database Abstraction Layer
  - Quick intro. to in-memory cache (Redis)

Why Cloud?

- Uneven Utilizations by nature
  - Day of week and Hour of day: Web surfing
  - Season of year: Christmas e-gift cards
  - Adhoc usage: One-off computation jobs/testing

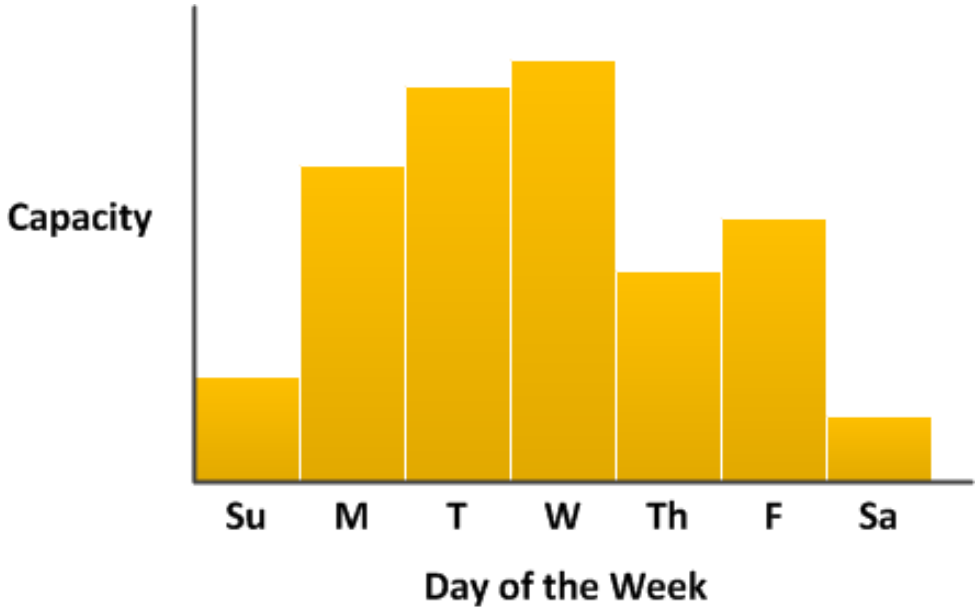
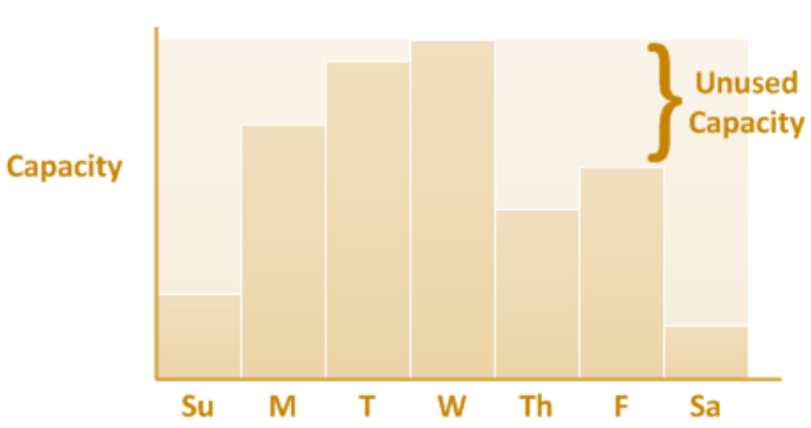
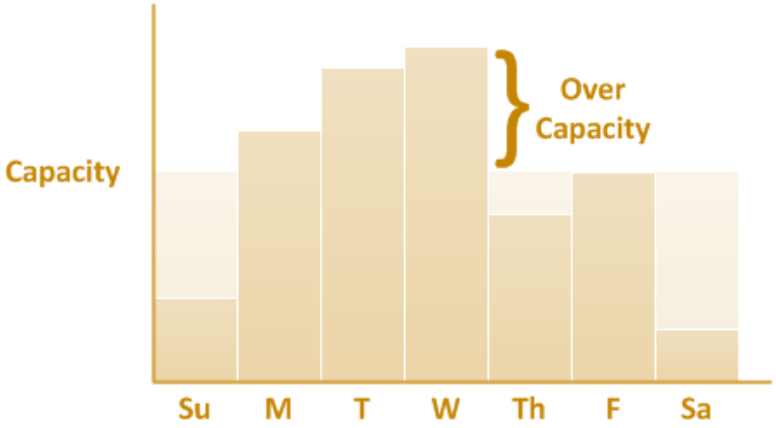


Image Ref: http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-dg.pdf

Cloud Benefits: on-demand + sharing





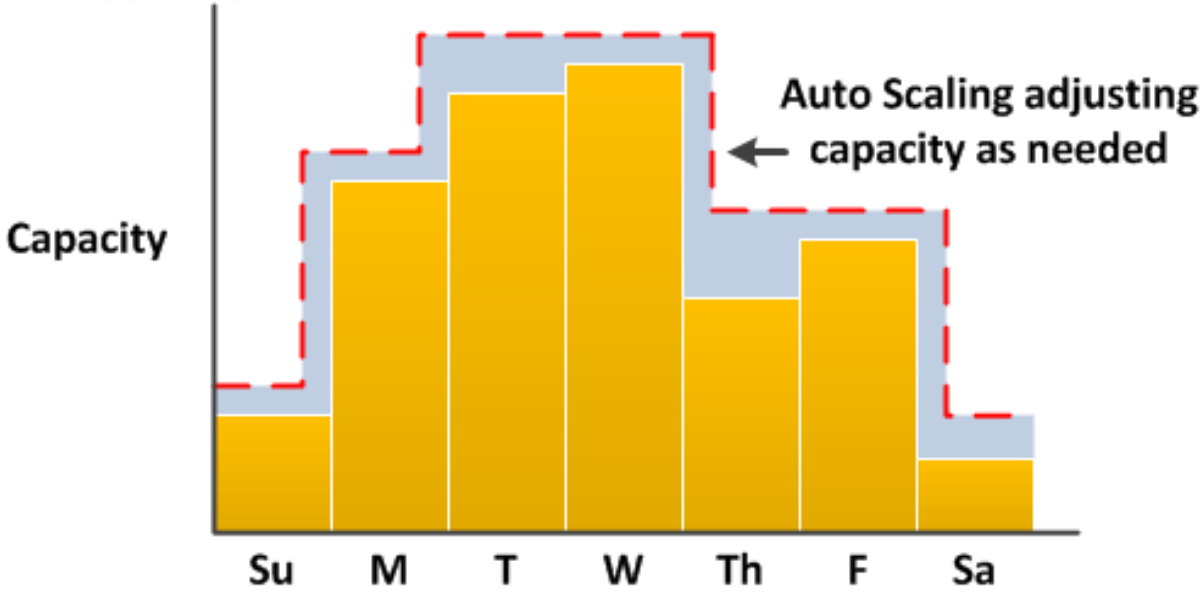


Image Ref: http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-dg.pdf

Cloud Classifications

- Cloud Service Models
  - Infrastructure-as-a-Service (IaaS)
  - Platform-as-a-Service (PaaS)
  - Software-as-a-Service (SaaS)

Suddenly, everything become X-as-a-Service (XaaS)

- What service models?
  - AWS EC2
  - AWS Elastic Beanstalk
  - Google Cloud Console
  - Google AppEngine

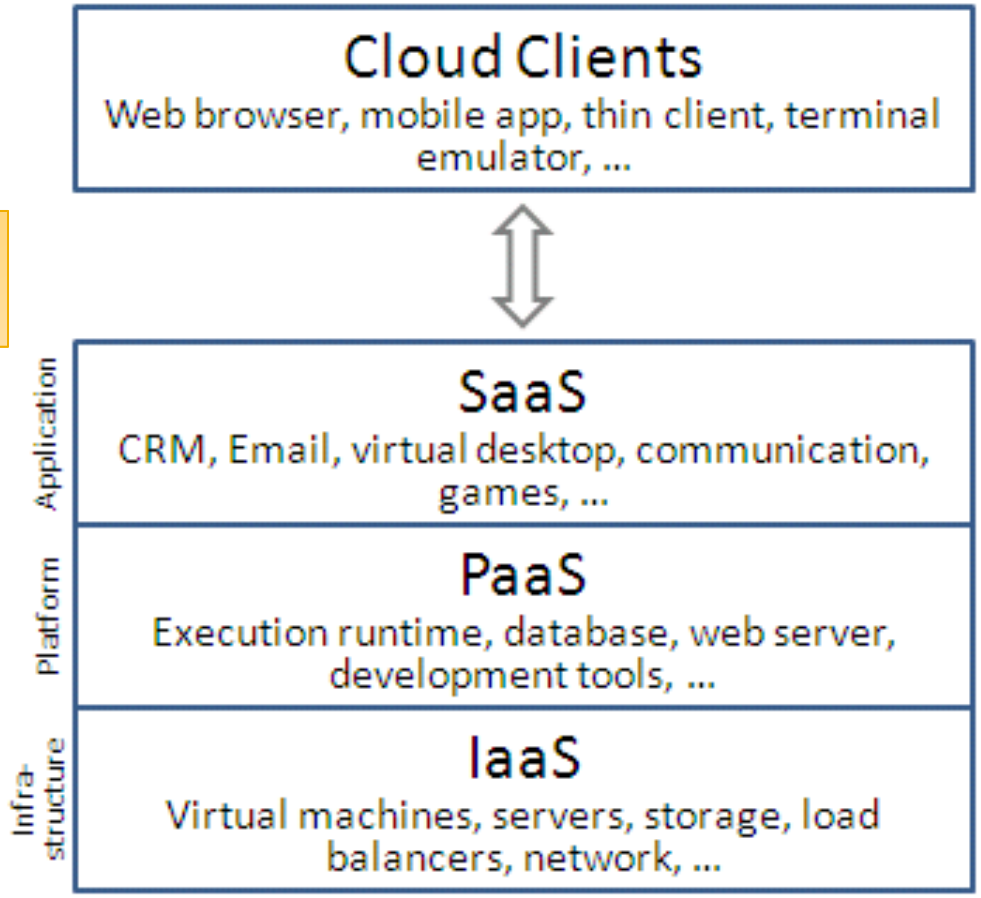
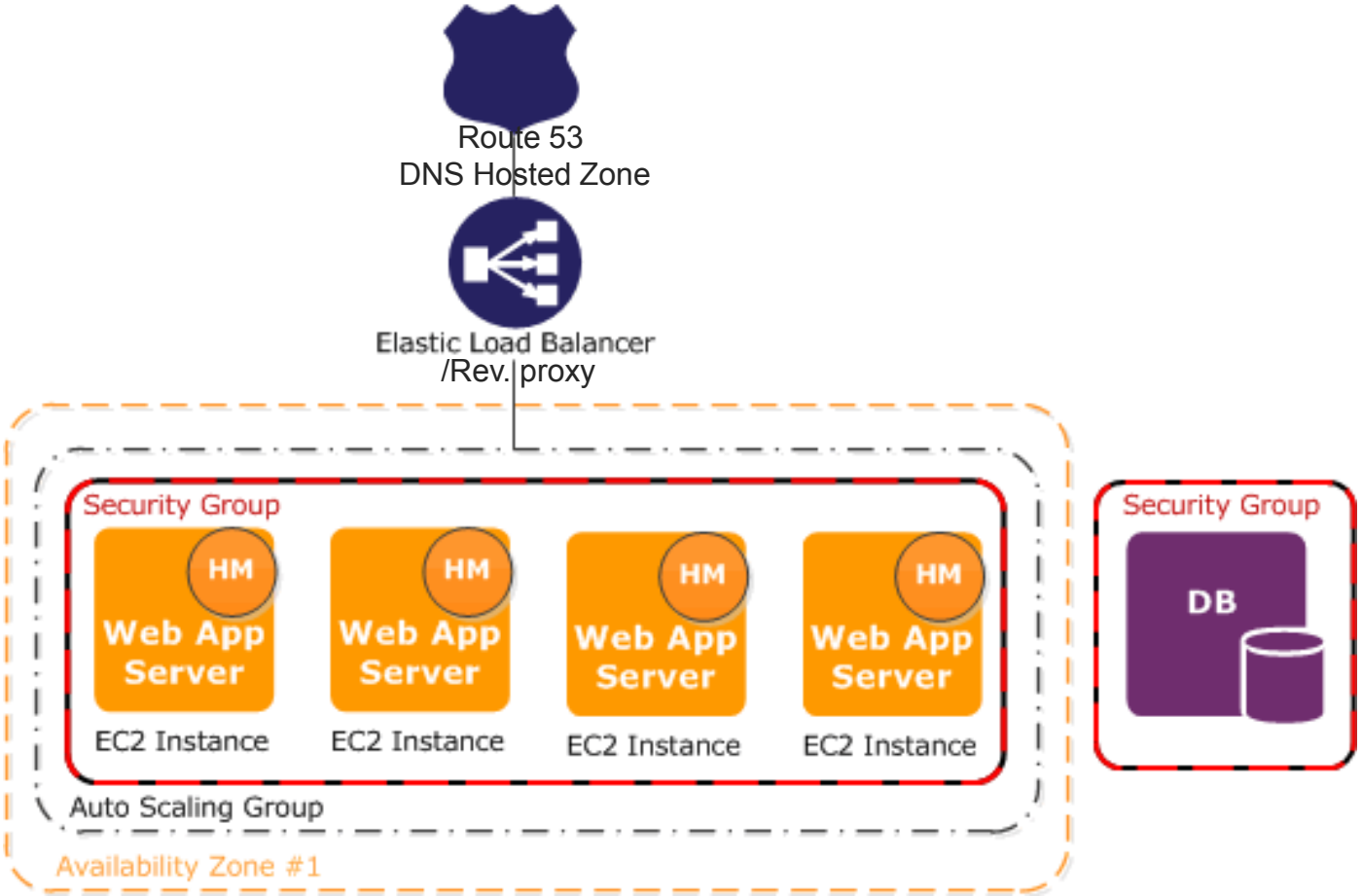


Image Ref: http://en.wikipedia.org/wiki/Cloud_computing#mediaviewer/File:Cloud_computing_layers.png

Fast and Scalable Web Server Platform

- Architecture of AWS Elastic Beanstalk: Web Server Tiers



Details and Image Ref: http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.concepts.architecture.html

Scale up v.s. Scale out

- Scale up (vertically): more resources for a single node
  - More expensive for supercomputer (less efficient)
- Scale out (horizontally): more nodes
  - A farm of cheaper instances well-networked (high thruput)
  - application distributable in idv. instance

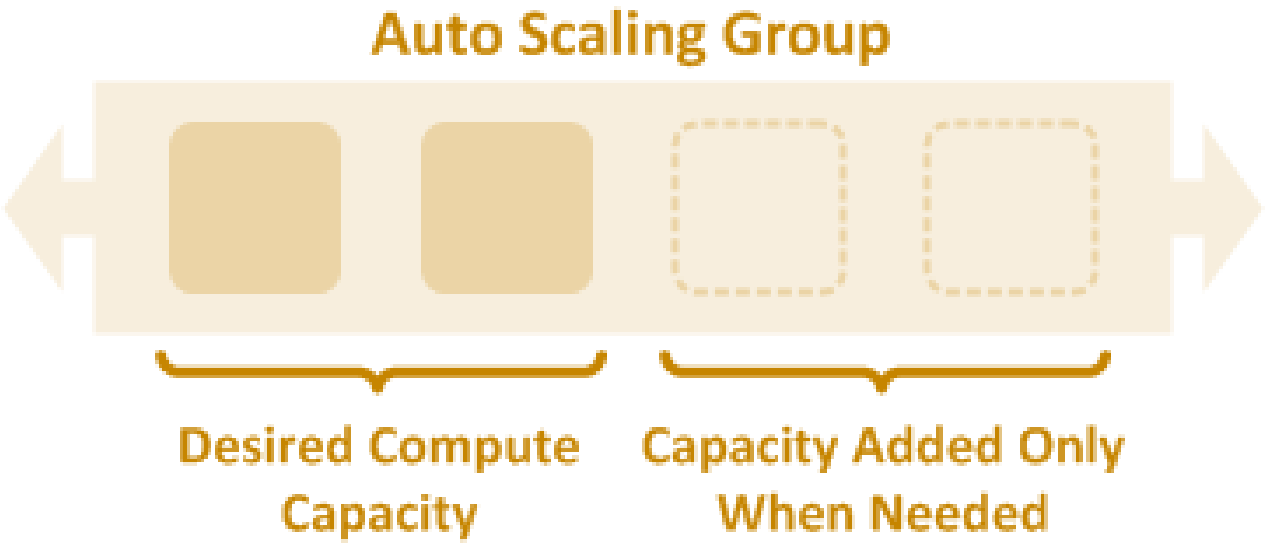


Image Ref: http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts.concepts.architecture.html

Design Considerations

- Scalability and Elasticity
  - Auto Scaling enables on demand instance/node creation/removal
  - Possibly based on metrics: CPU, memory, disk I/O, network I/O, etc…
- Fault Tolerance for Availability
  - Automated recovery for EC2 instances when some of them die
  - Automated backups for Storage
- Software Deployability
  - Easy to deploy; and create new environment to test out changes
  - Integrated with GIT for systematic versioning control

- that's why we chose AWS EB
  - plus we're awarded the education grant :)

DB STORAGE SERVERS

Introduction to DB Storage

- Relational Databases
  - Structured in tables : Slow but powerful
  - Hard to scale
  - Accessible through the Structured Query Language (SQL)
  - Often used as persistence storage
  - Examples: MySQL (free), MSSQL, Oracle, SQLite (free), etc
- NoSQL Databases
  - Unstructured as a tradeoff for speed
  - Easy to scale out, mostly query, async, inaccuracy tolerance by app
  - Accessible through API
  - Optimized for speed, thus often as In-memory Cache
  - Examples: Redis (free), Memcached (free), MongoDB, etc

Database: SQLite

- Public domain license (i.e. FREE!)
- Lightweight in design
  - Lightweight: multiple processes can read at the same time; however, only one process can make changes at any moment in time
  - Best for single-user apps (MobileApps/Simple WebApps)
- Supported by multi-platforms (e.g. Windows, Linux)
  - Pre-installed in AWS EC2
- Stores everything in a single file
  - Easy to embed, test, backup and transfer
- Simple access-right management
  - No user account management as in full-blown DBs like MySQL
  - Simply depends on the file access rights

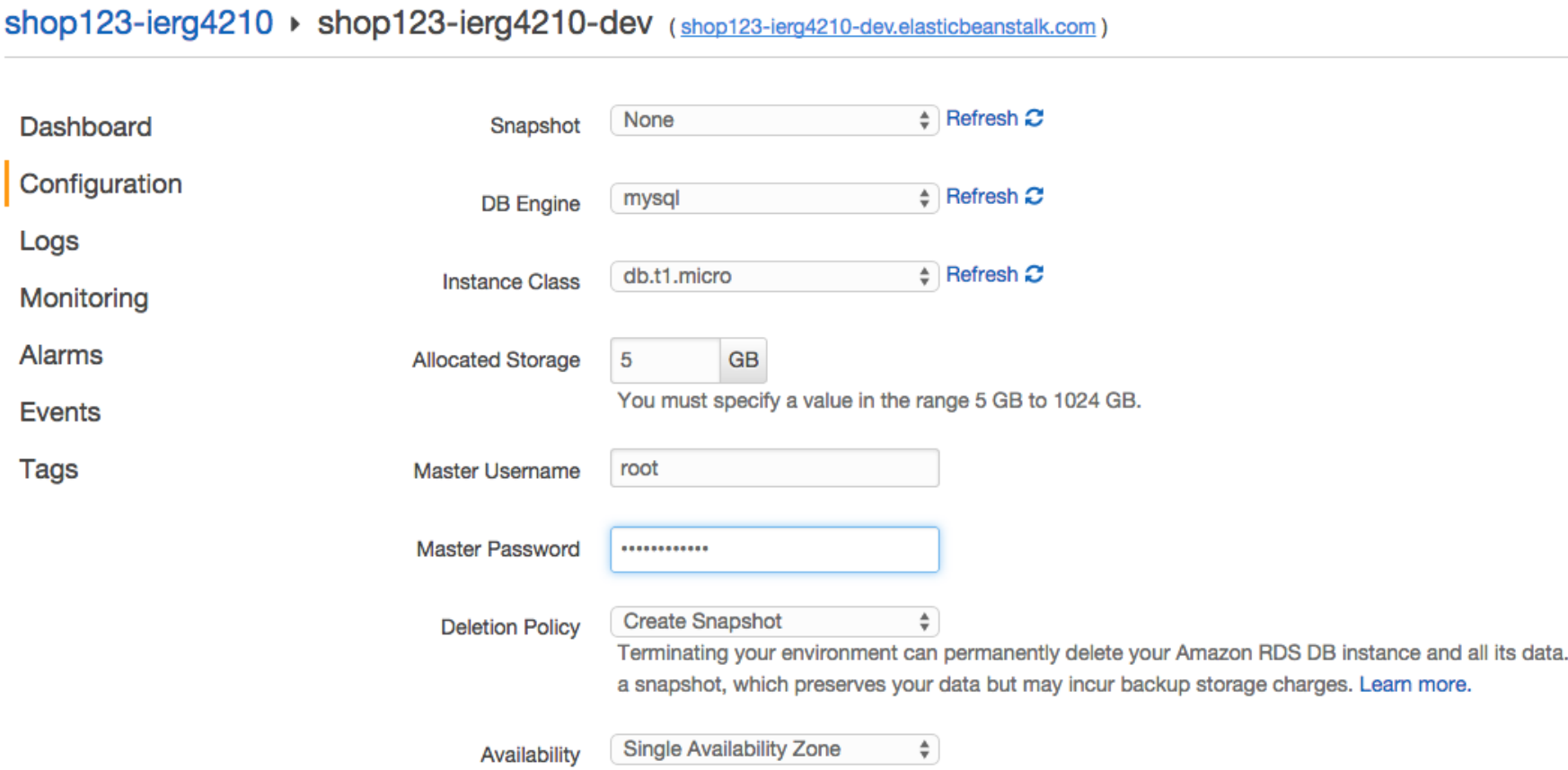
Database: MySQL

- Dual-licensing: GPL/FLOSS + proprietary
- Relational DB
  - Table structure
- Full-featured, accessible using SQL
  - But heavyweight, quite slow
  - Powerful as something for free
- Supported by multi-platforms (e.g. Windows, Linux)
  - Pre-installed in AWS EC2

MySQL favor

SQL LANGUAGE

Create a MySQL DB for EB

1. (local-env) $ eb console
2. Click Configuration -> Under Data Tier, Click Create DB
3. 

Connect to the DB using MySQL CLI

1. (local-env) $ eb ssh
2. $ sudo yum install mysql -y
3. $ mysql -u root -p -h <your-db>.rds.amazonaws.com
   - Enter your configured Master Password

- It fails when connecting directly to MySQL from local. Why?
  - The EC2 instances and MySQL are in the same Security Group, hence in the same Virtual Private Cloud (network)
  - From Security Group settings, expect only port 80 is opened
    - SSH is hosted at port 22, and is dynamically made accessible by eb ssh
    - MySQL is hosted at 3306, and thus cannot be accessed by public
  - Hence, access the DB thru EC2

MySQL: Create a User and DB

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 26
Server version: 5.5.40-log Source distribution

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> CREATE USER 'shopXX-admin' IDENTIFIED BY 'mypass';
mysql> CREATE DATABASE shopXX;
mysql> SHOW DATABASES;
mysql> GRANT ALL ON shopXX.* TO 'shopXX-admin';
mysql> exit;
```

MySQL: Login using New User and Pick new DB

- Re-login using the newly created user
  - $ mysql -u shopXX-admin -p -h <your-db>.rds.amazonaws.com
- Pick the newly created DB
  - mysql> USE shopXX;

- Considerations:
  - shopXX-admin is granted full access to DB called shopXX
    - You may like practicing the least privilege approach (e.g., SELECT)
    - To drop/delete the user:
      mysql> DROP USER shopXX-admin
  - Using shopXX-admin@localhost will fail because we'd support remote logins from the EC2

Create a Table (1/2)

- Creating a Table "categories"
  - Create the table

```
mysql> CREATE TABLE categories (
           catid INTEGER PRIMARY KEY AUTO_INCREMENT,
           name VARCHAR(512) NOT NULL
       ) ENGINE=INNODB;
```

    Note: Primary key is unique and auto-increment by default
    (i.e. incremented by 1 automatically for every new record)
  - To check what you have created

```
mysql> DESCRIBE categories;
```

  - To drop/delete the whole table and data

```
mysql> DROP TABLE categories;
```

  - To drop/delete all the data

```
mysql> TRUNCATE categories;
```

Create a Table (2/2)

- Creating a Table "products"
  - Create the table (simplified, add price/description type yourself)

```
mysql> CREATE TABLE products (
         pid INTEGER PRIMARY KEY AUTO_INCREMENT,
         catid INTEGER,
         name VARCHAR(512),
         price ________________,
         description _____________,
         FOREIGN KEY(catid)
           REFERENCES categories(catid)
       ) ENGINE=INNODB;
```

  Reference: Datatypes supported by MySQL

  - Create an index for catid – to make subsequent queries by catid faster

```
mysql> CREATE INDEX i1 ON products(catid);
```

INSERT

- Inserting some records to the newly created tables
  - To insert a record into categories

    mysql> `INSERT INTO categories VALUES (null, "Fruits");`

    Note: put null for the primary key to let it auto-increment
  - To insert 2 records into products

    mysql> `INSERT INTO products`
    `VALUES (null, 1, "Apple", "1.5");`

    mysql> `INSERT INTO products (catid, name, price)`
    `VALUES (1, "Banana", "1.5");`
  - Try to insert a product to an inexistent category:

    mysql> `INSERT INTO products (catid, name, price)`
    `VALUES (2, "Help", "999");`

    Error: constraint failed

    Note: This error is expected given that the foreign key setting
  - More on SQL INSERT: http://dev.mysql.com/doc/en/insert.html

SELECT

- Looking up records
  - To select all "fruits" in products (given fruits is of catid=1):
    mysql> `SELECT * FROM products WHERE catid = 1;`
  - To select only the name and price columns
    mysql> `SELECT name, price FROM products WHERE catid = 1;`
  - To select only 5 "fruits" in products:
    mysql> `SELECT * FROM products WHERE catid = 1 LIMIT 5;`
  - To select the 11-20th most expensive "fruits" in products: (Pagination?)

```
mysql> SELECT * FROM products
         WHERE catid = 1
         ORDER BY price DESC
         LIMIT 11, 10;
```

  - Recall: For those columns that are frequently queried, remember to create INDEX for performance sake (trading off space for speed)
  - More on SQL SELECT: http://dev.mysql.com/doc/en/select.html

UPDATE

- Updating a record
  - Setting a static value

```
mysql> UPDATE categories
       SET name = "Fresh Fruits"
       WHERE catid = 1;
```

  - Setting an expression (e.g. 10% increase in price)

```
mysql> UPDATE products
       SET price = price * 1.1
       WHERE pid = 2;
```

  - More on SQL UPDATE: http://dev.mysql.com/doc/en/update.html
- Note:
  - The WHERE conditions is the same as that of SELECT
  - So, when you are not sure about what records are affected
    - SELECT the records first, then replace it with UPDATE
    - Otherwise, you can kill all your data unintentionally

DELETE

- Deleting a record
  - Deleting the fruit category will result in an error
    mysql> `DELETE FROM categories WHERE catid = 1;`
    Reason: Given foreign key is ON, a cat with children can't be deleted
  - Deleting a product
    mysql> `DELETE FROM products WHERE pid = 2;`
  - The where conditions are again the same as that of SELECT
  - More on SQL DELETE: http://dev.mysql.com/doc/en/delete.html
- Deleting records requires extra attention!
  - Backup your database
  - Or SELECT what rows are affected before performing DELETE

Database Abstraction Layer

- A universal interface for accessing to different databases
  - Coding Consistency: Regardless of the DB, use the same set of code
  - Single Interface: Easy to switch database without code modifications

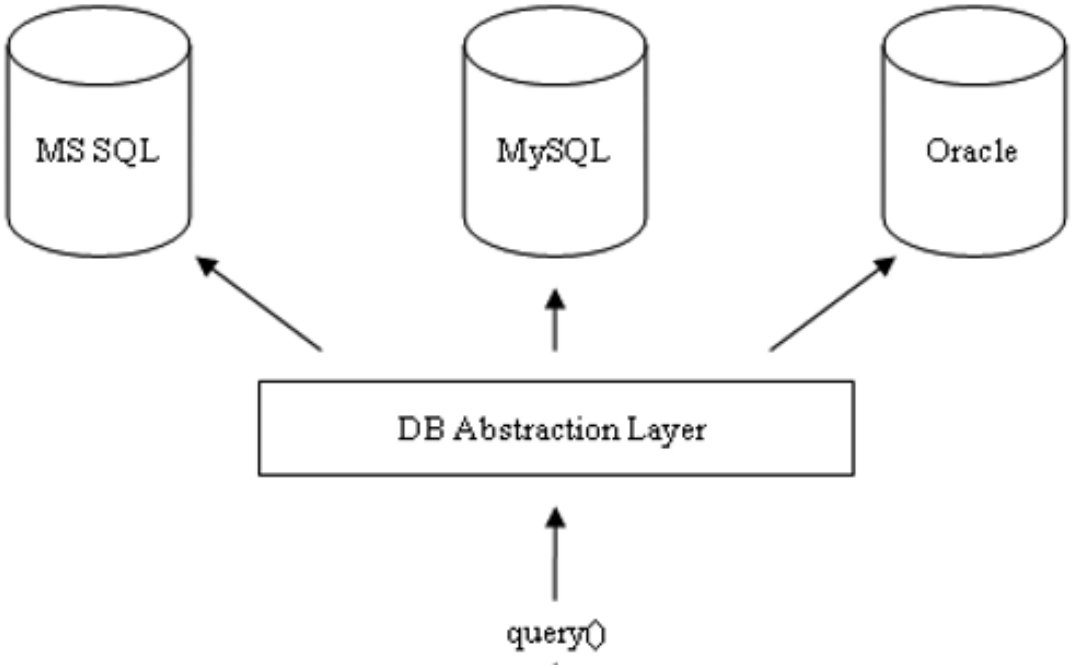


Image Ref: W. Jason Gilmore, Beginning PHP and MySQL From Novice to Professional, p.794, 2007

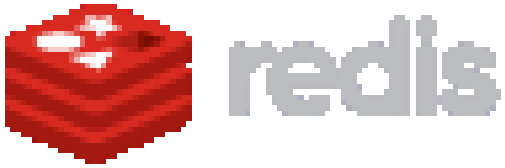
Example Usage

- Using Any-DB,

```
var anyDB = require('any-db');
var config = require('../shopXX-ierg4210.config.js');
var pool = anyDB.createPool(config.mysqlURI, {
                min: 2, max: 20
});

app.get('/', function (req, res) {
  // async fetch data from SQL, render page when ready
  pool.query('SELECT * FROM categories', function (error, results) {
    if (error) {
      console.error(error);
      return res.status(500).end();
    }
    res.render('home', {
      title: 'IERG4210 ShopXX',
      cat: results.rows
    });
  });
});
```

Redis



- Open-source NoSQL DB/Cache
  - In-memory key-value store (hence, very fast)
  - But also supports data structures such as sorted sets and lists

- Common use cases:
  - To serve queries, therefore GET requests
    e.g., cache your templates to prevent from re-rendering
    - When to expire? Expire on DB update?
  - To completely serve as a DB
    - Data loss when machine powers down (due to in memory)
    - Periodically backup data to persistent storage

- See Redis NPM, Redis.io, and AWS ElastiCache for details

Some Logistics…

- Assignment Phase 3 Released