

Project 5: Scaling Fabflix and Performance Tuning

Due Jun 11 by 11:45pm **Points** 100

▼ Task1: JDBC Connection Pooling

In this task, we will enable Fabflix with Connection Pooling.

Step 1: Enable JDBC Connection Pooling for Fabflix for all the servlets. Deploy the following [TomcatPooling example \(https://github.com/UCI-Chenli-teaching/cs122b-project5-TomcatPooling-example\)](https://github.com/UCI-Chenli-teaching/cs122b-project5-TomcatPooling-example). Note that the creation steps of mytestuser are different. MySQL 8.0 enables a more secure authentication by default but is less compatible with old drivers. To maintain compatibility, we use the traditional authentication method by adding "WITH mysql_native_password" when we create new test users in P5.

Step 2: Use Prepared Statements in all JDBC statements **involving user input**. You should have already done this in Project 3. If you haven't done so, please do it now, at least for Search functionalities, since Task 3 needs it. You can use this [tutorial \(http://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html\)](http://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html) on Prepared Statements.

- Prepared Statements are usually associated with one connection, which becomes tricky if you use a Connection Pool. The setting 'cachePrepStmts' can be set to true to cache the prepared statements.
- You can make this setting true by adding the flag to the JDBC connection URL in the /META-INF/context.xml like this -

```
url="jdbc:mysql://<IP>:<PORT>/moviedb?autoReconnect=true&allowPublicKeyRetrieval=true&useSSL=false&cachePrepStmts=true"
```

In the provided template of the [README.md \(https://canvas.eee.uci.edu/courses/54347/files/22162692/download\)](https://canvas.eee.uci.edu/courses/54347/files/22162692/download) file, explain briefly how and where (file name and file path on GitHub) you use the JDBC Connection Pooling and Prepared Statements in your code.

▼ Task2: MySQL Master-Slave Replication

This tutorial shows how to setup a MySQL cluster on AWS that includes a master and a slave. Reads can be sent to any of the two instances and writes must be sent to the master instance.

1. Setup two new AWS instances:

- Create two new AWS instances identical to your previous instance used to run Fabflix. We call these "instance 2" (as the master) and "instance 3" (as the slave). We assume their public IPs are 2.2.2.2 and 3.3.3.3, and their internal AWS IPs are 172.2.2.2 and 172.3.3.3, respectively.
- Add new inbound rules in their security group(s) to open their MySQL/Aurora type (port 3306) to each other using their AWS internal IP.

2. Set up the master instance.

- SSH to the master instance. Run the following command to install MySQL server.

```
master-shell> sudo apt-get update
master-shell> sudo apt-get install mysql-server
```

- Edit the `/etc/mysql/mysql.conf.d/mysqld.cnf` file and set the `bind-address` to **0.0.0.0** . Also, uncomment the lines of `server-id` and `log_bin` properties.
- Restart the MySQL service.

```
master-shell> sudo service mysql restart
```

- Login to the MySQL console. Create a new user and give it permission to replicate the database:

```
master-shell> sudo mysql -u root -p
master-mysql> create user 'repl'@'%' identified WITH mysql_native_password by 'slave66Pass$word';
master-mysql> grant replication slave on *.* to 'repl'@'%' ;
```

- Run the following command in the MySQL console to get the status of the master MySQL:

```
master-mysql> show master status;
```

Keep the values of the `File` and `Position` columns, as they are needed in the next step. For example, these values are `mysql-bin.000001` and `337`.

3. Set up the slave instance:

- SSH to the slave instance. Run the following commands to install MySQL server.

```
slave-shell> sudo apt-get update
slave-shell> sudo apt-get install mysql-server
```

- Edit the `/etc/mysql/mysql.conf.d/mysqld.cnf` file and set the `bind-address` to **0.0.0.0** . Also, uncomment the `server-id` property and use value 2. Do **not** uncomment the `log_bin` property.
- Restart the MySQL service.

```
slave-shell> sudo service mysql restart
```

- Login to the MySQL console and let the slave know about the master server:

```
slave-shell> sudo mysql -u root -p
slave-mysql> CHANGE MASTER TO MASTER_HOST='172.2.2.2', MASTER_USER='repl', MASTER_PASSWORD='slave66Pass$word', MASTER_LOG_FILE='mysql-bin.000001', MASTER_LOG_POS=337;
```

Note that the values used for `MASTER_LOG_FILE` and `MASTER_LOG_POS` are from the last step.

- Start the MySQL slave:

```
slave-mysql> start slave;
slave-mysql> show slave status;
```

It should show "Slave_IO_State: Waiting for master to send event".

4. Test if the synchronization works:

- On the master instance, create a database with a table with a tuple:

```
master-shell> sudo mysql -u root -p
master-mysql> create database pets;
master-mysql> use pets;
master-mysql> create table cats(name varchar(20));
master-mysql> insert into cats values ("fluffy");
```

- On the slave instance, check if the data has been propagated from the master:

```
slave-shell> sudo mysql -u root -p
slave-mysql> show databases;
slave-mysql> use pets;
slave-mysql> show tables;
slave-mysql> select * from cats;
```

We should see the database, table, and record on this instance propagated from the master. If so, congratulations!

5. Create moviedb on the master instance:

- Populate moviedb on the master instance using "create_table.sql" and "data.sql" (You have done this many times)
- On the slave instance, check if the data is automatically propagated:

```
slave-shell> sudo mysql -u root -p
slave-mysql> use moviedb; show tables;
```

Note: The propagation is only from the master to the slave, not vice versa. To test it, you can insert some records into a table in the slave database and check if it's propagated to the master instance. It should NOT!

Note: When you restart your AWS instances, the IP addresses of your instances could change. At this time, you need to reset the master address on the slave instance to reconnect them. However, note that MASTER_LOG_FILE and MASTER_LOG_POS could change; thus, you need to recheck the master status.

```
master-mysql> show master status;
slave-mysql> stop slave;
slave-mysql> CHANGE MASTER TO MASTER_HOST='172.2.2.2', MASTER_USER='repl', MASTER_PASSWORD='slave66Pass$word', MASTER_LOG_FILE='mysql-bin.000001', MASTER_LOG_POS=337;
slave-mysql> start slave;
slave-mysql> show slave status;
```

Note: It may be easy to create new AWS instances using an image of an existing instance. To do so, (1) select an existing instance and "Create Image" for it, and (2) Select the generated image under "IMAGES" -> "AMIs" and "Launch" instances using this image. If you use this approach, delete the file "auto.cnf" used by MySQL with the instance's auto-generated UUID.

```
shell> sudo \rm -rf /var/lib/mysql/auto.cnf
```

▼ Task3: Scaling Fabflix with a cluster of MySQL/Tomcat and a load balancer

In this example, we will scale up Fabflix to handle massive traffic using a MySQL and Tomcat cluster. Since the user would have only one entry point to Fabflix, we need a load balancer to balance the traffic from one entry point to multiple Fabflix instances.


Example:

Step 1 (master/slave): Setup Tomcat on each master/slave instance. (You should have done it many times.)

Step 2 (master/slave): On each master/slave instance, deploy the [TomcatPooling example \(https://github.com/UCI-Chenli-teaching/cs122b-project5-TomcatPooling-example\)](https://github.com/UCI-Chenli-teaching/cs122b-project5-TomcatPooling-example) to serve as an example for DB connection. Make the URL

http://PUBLIC_IP:8080/cs122b-project5-TomcatPooling-example work. You may need to modify the username/password and IP address

(use the internal AWS instance address). Also, modify the AWS security group setting for these two instances to allow remote access to their 8080 port.

Step 3 (master/slave): On each master/slave instance, deploy [Session example](https://github.com/UCI-Chenli-teaching/cs122b-project2-session-example)  (<https://github.com/UCI-Chenli-teaching/cs122b-project2-session-example>). Make the URL `http://PUBLIC_IP:8080/cs122b-project2-session-example/session?myname=Michael` work.

Step 4 (instance 1): On the instance that runs the original Fabflix instance (called "instance 1"), set up Apache and its proxy by doing the following:

- Install Apache2 and related modules:

```
instance1-shell> sudo apt-get install apache2
instance1-shell> sudo a2enmod proxy proxy_balancer proxy_http rewrite headers lbmethod_byrequests
instance1-shell> sudo service apache2 restart
```

Modify the security group to open port 80 of instance 1 to your IP address. Then, use your browser to open a URL using the public IP address of instance 1 (e.g., http://PUBLIC_IP:80 (http://public_ip/)), and you should see an "Apache2 Ubuntu Default Page". It means you have successfully set up Apache2 on this machine.

- Configure the Apache2 webserver to use its `proxy_balancer` module for sharing (i.e., redirecting) requests to the backend instances. To do it, edit the following configuration file:

```
instance1-shell> sudo vim /etc/apache2/sites-enabled/000-default.conf
```

- Create a load balancer proxy whose members are the backend instances. In particular, define a proxy on top of the file before the `<VirtualHost *:80>` tag.

```
<Proxy "balancer://TomcatPooling_balancer">
  BalancerMember "http://172.2.2.2:8080/cs122b-project5-TomcatPooling-example/"
  BalancerMember "http://172.3.3.3:8080/cs122b-project5-TomcatPooling-example/"
</Proxy>
```

Here we assume '172.2.2.2' and '172.3.3.3' are the private IP address of the master and slave instances, respectively.

- Add two new rules in the body of the VirtualHost tag.

```
ProxyPass /cs122b-project5-TomcatPooling-example balancer://TomcatPooling_balancer
ProxyPassReverse /cs122b-project5-TomcatPooling-example balancer://TomcatPooling_balancer
```

- Restart Apache:

```
instance1-shell> sudo service apache2 restart
```

- Modify the security group of the two backend instances to allow instance 1 (private ip) to access their 8080 port.

These settings will redirect HTTP requests to "instance1_IP/cs122b-project5-TomcatPooling-example" to one of the two backend instances. To test it, use a browser to point to `http://instance1_IP/cs122b-project5-TomcatPooling-example`. Be sure to open port 80 of instance 1 to your IP address. Next, check the Tomcat access log of the two backend instances. Note the log file path may vary depending on your Tomcat installation.

```
instance2-shell> sudo tail -f /var/lib/tomcat/logs/*
instance3-shell> sudo tail -f /var/lib/tomcat/logs/*
```

One of them should receive that request. Keep refreshing the page to send multiple requests, and check if the two backends are receiving the requests evenly.

Step 6 (instance 1): Configure the proxy on instance 1 to handle sessions properly. Since the current setting will send requests randomly to the backend, it will not pass cookies properly, causing sessions to fail. Instead, we want to make the session persist over several requests of the same client, i.e., to have a *sticky session*. To do it, read the [instructions](#) (http://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html), especially those under "Examples of a balancer configuration."

Here's a sample set for the `/etc/apache2/sites-enabled/000-default.conf` file for the "[Session example](#)" (<https://github.com/UCI-Chenli-teaching/cs122b-project2-session-example>) application:

```
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=BALANCER_ROUTE_CHANGED

<Proxy "balancer://Session_balancer">
    BalancerMember "http://172.2.2.2:8080/cs122b-project2-session-example" route=1
    BalancerMember "http://172.3.3.3:8080/cs122b-project2-session-example" route=2
    ProxySet stickysession=ROUTEID
</Proxy>
```

Also, do the following:

- Add two new rules in the body of the VirtualHost tag.

```
ProxyPass /cs122b-project2-session-example balancer://Session_balancer
```

```
ProxyPassReverse /cs122b-project2-session-example balancer://Session_balancer
```

- Restart Apache:

```
instance1-shell> sudo service apache2 restart
```

Test if it works by pointing to the URL http://instance1_IP/cs122b-project2-session-example/session?myname=Michael of instance 1. It should access one of the backend instances only.

Here's a sample Apache configuration file [000-default.conf \(https://canvas.eee.uci.edu/courses/54347/files/22932788/download\)](https://canvas.eee.uci.edu/courses/54347/files/22932788/download).

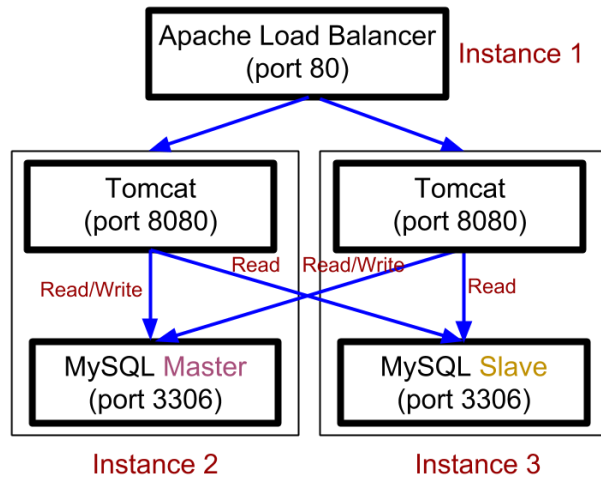
Main Task:

Sub-Task 1: This task will need three AWS instances (Load balancer, Master, Slave).

- Deploy your Fabflix application to the two backend instances (master/slave). Then, do MySQL master/slave replication.
- Configure the load balancer instance to enable load balancing, Connection Pooling, and sticky sessions.
- Make sure the write requests are sent to the master MySQL instance only, while the read requests should be sent to either the master or the slave MySQL instance. Accessing a remote MySQL server (e.g., master MySQL from Tomcat on the slave instance) needs configuration changes, as explained in the last part of the [setup script \(https://canvas.eee.uci.edu/courses/54347/files/22162696/download\)](https://canvas.eee.uci.edu/courses/54347/files/22162696/download).
- (Optional) Enabling the scaled version with HTTPS.
- **Explain in the README.md how to use Connection Pooling using two backend servers. Mention your configuration file name and path in your README.md.**

The following is the architecture diagram:





Notice that each Tomcat can talk to both MySQL instances in this architecture. For example, we can set up a cluster of MySQL and let each Tomcat talk to the cluster through another load balancer. If interested, you can read this [page \(http://mysqlhighavailability.com/setting-up-mysql-router/\)](http://mysqlhighavailability.com/setting-up-mysql-router/) on how to set it up.

Sub-Task 2: Google Cloud Platform (GCP): Install and configure the load balancer on a new Google Cloud instance. Just like the load balancer on AWS, the GCP load balancer must redirect all the requests to either of the master/slave instances hosted on AWS. In addition, the GCP load balancer needs to use the public IPs of the AWS (master/slave) instances and open port 8080 to the GCP load balancer.

You can get \$300 free credits for Google Cloud from the [free trial \(https://cloud.google.com/free/\)](https://cloud.google.com/free/). After that, start a computing engine (using Ubuntu 20.04) by following this [tutorial \(https://www.cloudbooklet.com/how-to-setup-ubuntu-20-04-on-google-cloud-compute-engine/\)](https://www.cloudbooklet.com/how-to-setup-ubuntu-20-04-on-google-cloud-compute-engine/), and repeat the same steps as you did to install the load balancer on AWS.

▼ Task4: Measuring the performance of Fabflix search feature

In this part, we will measure the performance of the keyword search feature you implemented in past projects. The measurement results described in subtasks 4.1 and 4.2 must be reported for both the single instance (i.e., the instance you prepared in earlier projects and

Task1) and the scaled version of Fabflix (from Tasks 2&3). Before running the test plans, you must load the original movie-data.sql and the movies from the XML parser.

Note: The URL to the single-instance version should be `http://INSTANCE1_PUBLIC_IP:8080/your-project-name`, while it should be `http://INSTANCE1_PUBLIC_IP:80/your-project-name` for the scaled version that you prepared in Task 2&3.

Task 4.1: Preparing the codebase for time measurement

Here, we are going to prepare to measure the following two statistical variables:

1. The average time it takes for the search servlet to run entirely for a query (called TS).
2. The average time spent on the JDBC parts per query (called TJ).

TS > TJ always.

Step 1: Insert the necessary time statements for measuring TS and TJ in the following sample. This is a helpful [article \(https://javajee.com/writing-to-a-file-from-a-servlet-in-a-java-ee-web-application\)](https://javajee.com/writing-to-a-file-from-a-servlet-in-a-java-ee-web-application) about how to write to a file in a servlet. You are required to measure and log the value of "search servlet total execution time" and "JDBC execution time" for every request served by the server (i.e., 1000 queries fired to the server will result in 1000 lines in the log file each line containing one sample value for calculating TS and TJ).

Particularly for TS samples, placing these log statements in a filter that wraps the search servlet is highly recommended.

```
// Time an event in a program to nanosecond precision
long startTime = System.nanoTime();
////////////////////////////////////
/// ** part to be measured ** ///
////////////////////////////////////
long endTime = System.nanoTime();
long elapsedTime = endTime - startTime; // elapsed time in nano seconds. Note: print the values in nanoseconds
```

Step 2: Write a script (log_processing.* in any language you prefer) to process the output log file of a query workload and calculate the TS and the TJ (i.e., by parsing the log statements and taking the average of all the samples). This script will be needed in preparing the table section in the report asked for in Task 4.3 below.

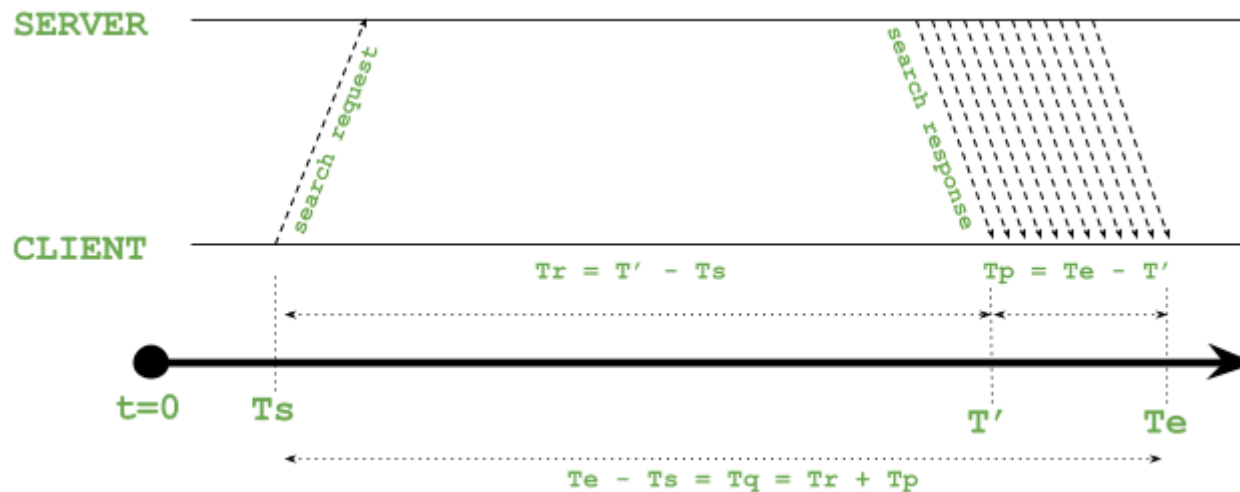
Submission-related note: In Task 4.3 below, you will run queries. Upload the log file created to Github and mention its location in your report. The usage of this script, which is expected to be found at the root of your .war file, must be explained in your README file.

Task 4.2: Preparing the test plan in Apache JMeter

In this part, you will use [Apache JMeter \(http://jmeter.apache.org/\)](http://jmeter.apache.org/) to measure the performance of the search feature of the Fabflix website. In particular, you must measure the **average query time** of the search feature using a set of queries based on the movie titles in this [file \(https://canvas.eee.uci.edu/courses/54347/files/22162697/download\)](https://canvas.eee.uci.edu/courses/54347/files/22162697/download). Assume the page size is 50, and we only want the first page of the results.

The following figure illustrates the round-trip time of a query from a client to the server and back to the client. The *query execution time* (i.e., "Tq") is the total time starting from when a search request is sent from the client (Ts) until the response is entirely received by the client (Te). It includes two major parts:

- (1) *response time* (Tr): the time it takes until the client hears the first bit of the response;
- (2) *payload time* (Tp): the time it takes for the response data to be downloaded by the client completely.



Step 1: Read this [reference \(http://jmeter.apache.org/usermanual/get-started.html\)](http://jmeter.apache.org/usermanual/get-started.html) to get an overview of JMeter. Read this [page \(http://jmeter.apache.org/usermanual/build-test-plan.html\)](http://jmeter.apache.org/usermanual/build-test-plan.html) to get familiar with JMeter basics.

Step 2: Download and install JMeter from this [link \(http://jmeter.apache.org/download_jmeter.cgi\)](http://jmeter.apache.org/download_jmeter.cgi).

Step 3: Make a JMeter test plan for the search feature of Fabflix, and then run the JMeter test from the local development against the remote AWS instance.

- JMeter test plan basics:
 - Refer to this JMeter manual [link \(http://jmeter.apache.org/usermanual/build-web-test-plan.html\)](http://jmeter.apache.org/usermanual/build-web-test-plan.html).
 - This [link \(https://www.digitaiocean.com/community/tutorials/how-to-use-apache-jmeter-to-perform-load-testing-on-a-web-server\)](https://www.digitaiocean.com/community/tutorials/how-to-use-apache-jmeter-to-perform-load-testing-on-a-web-server) is an example of performing load testing using JMeter.
- Plan requirement:
 - The plan must iteratively generate a proper HTTP or HTTPS search request for every movie title in the provided [query file \(https://canvas.eee.uci.edu/courses/54347/files/22162697/download\)](https://canvas.eee.uci.edu/courses/54347/files/22162697/download).
 - Here is a helpful [page \(http://ivetetecedor.com/how-to-use-a-csv-file-with-jmeter/\)](http://ivetetecedor.com/how-to-use-a-csv-file-with-jmeter/) about how to use a CSV file for JMeter.
- To test a website that uses a session, JMeter needs to be configured to use Cookie Manager to simulate the session. We provide a [sample \(https://canvas.eee.uci.edu/courses/54347/files/22162722?wrap=1\)](https://canvas.eee.uci.edu/courses/54347/files/22162722?wrap=1) JMeter test plan to help you start with JMeter Cookie Manager. It tests against the project2-login-cart-example we provided in P2.

When running JMeter to collect performance number, make sure to run it in the NON GUI mode. Here is an example command:

```
shell> [PATH]/jmeter -n -t sample.jmx
```

Task 4.3: Collecting the performance results

Run the tests for all the following settings to collect performance results. Remember to make the necessary changes to the JMeter test plan and the codebase for each case. Use the results to fill out the table sections in the provided [README.md \(https://canvas.eee.uci.edu/courses/54347/files/22162692/download\)](https://canvas.eee.uci.edu/courses/54347/files/22162692/download) file [_ \(https://canvas.eee.uci.edu/courses/54347/files/22162694/download\)](https://canvas.eee.uci.edu/courses/54347/files/22162694/download) as the measurement report. For each case, report the requested values in the corresponding columns, and write a short analysis in the last column. The image below is an example of what you should report in the second column called "Graph Results Screenshot."



Notes:

- In all cases, if not mentioned otherwise, your Fabflix codebase is assumed to use both the Prepared Statements and Connection Pooling optimization techniques.
- If more than one JMeter thread is to be used, each thread should start a new session in Tomcat (i.e., threads should not share a session id).

Single-instance cases (i.e., that are accessible via `http://INSTANCE1_PUBLIC_IP:8080/your-project-name`):

- Use HTTP, without using Connection Pooling, 10 threads in JMeter.
- Use HTTP, 1 thread in JMeter.
- Use HTTP, 10 threads in JMeter.
- Use HTTPS, 10 threads in JMeter.

Scaled-version cases (i.e., that are accessible via `http://INSTANCE1_PUBLIC_IP:80/your-project-name`):

- Use HTTP, without using Connection Pooling, 10 threads in JMeter.
- Use HTTP, 1 thread in JMeter.
- Use HTTP, 10 threads in JMeter.

▼ Rubric

General requirements: Same as project 4.

Basic part	Functionality	Details	Score
Task 1	Connection Pooling	Enable Connection Pooling for all servlets (both single and scaled versions).	10
		Answer questions from README.md (https://canvas.eee.uci.edu/courses/54347/files/22162692/download) template regarding Connection Pooling.	10
Task 2	Master/slave	Enable MySQL replication on the scaled version.	10
		Routing read/write queries correctly to Master/Slave MySQL.	5
		Answer questions from README.md (https://canvas.eee.uci.edu/courses/54347/files/22162692/download) template regarding Master/Slave.	5
Task 3	Load balancing	Master/slave AWS instances are set up and accessible with HTTP on port 8080, (optional) HTTPS on port 8443.	5
		AWS and GCP load balancers redirect traffic on port 80 to Master/Slave AWS instances.	10
		reCAPTCHA still works properly on website.	3
Task 4	Log files	Log files for all the 7 test cases with TS and TJ readings.	3
	Log Processing Script	Having the log_processing.* script and instructions of how to run it.	4
	JMeter TS/TJ Time Measurement Report	Fill in each cell of the table in the README.md (https://canvas.eee.uci.edu/courses/54347/files/22162692/download) file. There	35

		are 7 test cases and each case has 5 columns to be filled (7*5). Note that for the TS and TJ columns, we will be giving you marks only if we can locate the corresponding log files, with sufficient number of readings.	
Total			100

▼ Demonstration

In summary, we require two things for the demo:

1. A screen recording demo video **showing the JMeter test plan running against your instances.**
2. During the grading window, **leave 4 AWS instances and 1 GCP instance running (with web application deployed or load balancer configured)** for manual check.


Screen recording demo video

- **Please make sure your video is readable. Enlarge your font (terminal, browser) and make sure your video is not compressed.**
- You need a terminal connects to AWS, and a web browser open to tomcat manager page on AWS (<https://<AWS public IP>:8443/manager/html>).
- Use screen recording tools (e.g, QuickTime Player for macOS and Open Broadcaster Software for Windows, or other equivalents)
- Please keep the length of the video within **20** minutes. However, do **NOT** edit the video in any way. Otherwise it could be treated as cheating.
- Upload the video to Youtube or other public media hosting website, then include your video URL (publicly accessible) in [README.md](#) (<https://canvas.eee.uci.edu/courses/54347/files/22162692/download>), submit on GitHub.
- **Preparation before the demo:**

Before running the test plans, you must load the original movie-data.sql and the movies from the XML parser.

You should have 5 instances ready before the recording as below.

1. A single-version instance (the original Fabflix), opened on port **8080** for **HTTP** (redirect) and **8443** for **HTTPS**, with Connection Pooling enabled;

2. The Master-version **AWS** instance opened on port **8080** for **HTTP** with Connection Pooling enabled and master **MySQL**;
 3. The Slave-version **AWS** instance opened on port **8080** for **HTTP** with Connection Pooling enabled and slave **MySQL**;
 4. The **AWS** load balancer instance opened on port **80** for **HTTP**, redirecting traffic to the Master/Slave version instances;
 5. The **GCP** load balancer instance opened on port **80** for **HTTP**, redirecting traffic to the Master/Slave version instances.
- Demo on the local machine:
 1. Show your **AWS EC2** page and **GCP** Console Page showing your account and the running instances' IP.
 2. On your local machine, run 3 JMeter test plans as below, each for 5 minutes:
 - Single-instance version, test-case#3 using **HTTPS**, **10** threads in JMeter;
 - Scaled-instance version, test-case#2 using **HTTP**, **10** threads in JMeter, using the **AWS** load balancer.
 - Scaled-instance version, test-case#2 using **HTTP**, **10** threads in JMeter, using the **GCP** load balancer.
 - You can kill the JMeter for each test plan after 5 minutes limit.
 3. On your local machine, run the `log_processing.*` script against the sets of log files generated from the above 3 test plans; show the calculated TS/TJ time for each test plan.
 - Here is a [sample](https://youtu.be/HAUmobwm7io)  [.\(https://youtu.be/HAUmobwm7io\)](https://youtu.be/HAUmobwm7io) demo video.
-

▼ Submission

Same as Project 4, but please update README with the Project 5 content

In addition, make sure your submission contains the following:

- **img/** folder to contain all the 7 images of the JMeter screenshot of your test plans.
- **logs/** folder containing 7 SETS of log files each for a test plan.
- **README.md** (<https://canvas.eee.uci.edu/courses/54347/files/22162692/download>), from this template.