

UNIVERSITY OF
WATERLOO



SE464 Lab 1 Packet

Revised Fall 2023

Table of Contents

1 Learning Objectives.....	3
2 Background Information	3
3 Overview.....	4
3.1 Database	4
3.1.1 MySQL + RDS	4
3.1.2 DynamoDB.....	4
3.1.3 S3	4
3.2 Communication Framework.....	4
3.2.1 Express	5
3.2.2 gRPC.....	5
4 Important Disclaimers	5
5 AWS Account Setup	5
6 Lab Procedure	8
Testing Your System	11
Questions.....	11
Deliverables	12

1 Learning Objectives

After completing Lab 1, students will:

- Be able to set up a working basic web service with AWS which includes a database, a communication framework, and an EC2 instance.
- Understand the strengths, weaknesses, and trade-offs of working with different database types and different communication frameworks.
- Be able to combine different starter components into a system that works.
- Understand why the system built at the end of lab 1 is not production ready, and what can be added to it to make it production ready.

2 Background Information

The goal of this lab is to expose students to the fundamental components and design decisions involved in creating a modern software solution. One of the most common types of systems that exist in the world is the web system.

A **web system** is a software application whose content is accessed over the internet via the HTTP protocol through a web browser. In order to reliably deliver services to users, a combination of front and back-end components must work together seamlessly. The key components are:

1. **Front-End:** This is the in-browser UI that users interact with. Data is transmitted over the internet via HTTP in order to display content and communicate with the backend to submit user data.
2. **Servers:** Responsible for sending, receiving, and storing data. In a web system, there are often two types of servers used. In our case, we will simulate the communication between these two servers:
 - a. **Web Servers:** Responsible for serving web-based content to clients and handling and responding to HTTP requests from the clients.
 - b. **Application Servers:** Sit between the web servers and database and are responsible for transmitting data between the two. They contain logic for APIs that manipulate database contents.
3. **Database:** Used to manage and persist organized data that is required by the system. There are two major styles of databases:
 - a. **SQL Databases:** Use a rigid schema to define tables, columns, and relationships between data. SQL was introduced in CS 348.
 - b. **NoSQL Databases:** Do not have a rigid schema and are rather key-value stores that can handle both structured and unstructured data.
4. **Communication Protocol:** Allows for a client (web browser) and the web server to communicate over the internet. The most typical protocol for this is HTTP, which you learned / will learn more about in ECE 358. In software systems, this protocol is used in library functions or frameworks specifically crafted for facilitating communication between components.

However, a successful web system requires certain properties to be upheld in order for it to effectively serve its purpose no matter the demand from users. Namely, a web system must be:

- **Reliable & High-Performing:** The system should always do its job correctly without errors, and experience the least amount of downtime possible (i.e. be highly available).
- **Scalable:** The system should be able to handle various levels of user traffic as needed. This means servicing user requests in a timely manner (i.e. right away), and not letting infrastructure sit idle to waste money.
- **Secure:** The system should have robust security measures implemented in order to prevent attacks and ensure that sensitive data is protected.

No two web systems are the same. The components specified above exist in different forms and can be employed in different combinations to achieve the overarching goal of hosting a dynamic website that upholds the properties defining general success.

3 Overview

In this lab, your team will build a primitive web system which implements a very simplified online shopping application server. When you measure the performance of this application server, it will be similar to using a web server.

You have been provided several pre-built components, which you will complete, then combine in multiple different ways to create different infrastructure configurations. In particular, you will be exploring the particular databases and communication frameworks below.

3.1 Database

In this lab, you will explore working with both a relational (SQL) database and a K/V (NoSQL) database. In addition, you will explore working with a separate AWS service in conjunction with the database options when working with large files.

3.1.1 MySQL + RDS

MySQL is a relational database. RDS is an AWS service which makes it easy to set up, manage, and scale MySQL databases and other database services in the cloud. The documentation for MySQL hosted on RDS can be found [here](#).

3.1.2 DynamoDB

DynamoDB is a key-value NoSQL database. The AWS documentation for DynamoDB can be found [here](#).

3.1.3 S3

S3 is an object storage service. It has several use cases, from building cloud-native apps, to building data-lakes. The documentation can be found [here](#). In this lab, S3 will be used to store large files, such as images.

3.2 Communication Framework

Skeleton code will be provided for you. You will be working with Express (REST API) and gRPC with Node JS to allow for your system's components to communicate with each other, as well as to users over the internet.

3.2.1 Express

Express is a Node.js web app framework that makes it easy and simple to create a robust API. The Express documentation can be found [here](#).

3.2.2 gRPC

gRPC is a Remote Procedure Call (RPC) framework. The documentation for gRPC can be found [here](#).

4 Important Disclaimers

Completing the labs should not incur any costs. Everything can be done with the [AWS Free Tier](#). However, when you create your AWS account, AWS will ask you for a payment method. If your group is uncomfortable providing one of your credit cards, please reach out to Ahmed.

As you go through the lab manual, be very careful when setting up your infrastructure that you are choosing instance sizes that fit within the free tier. If not, you will be charged for the infrastructure you use. In the case that this does happen, you may be able to get your money back by contacting AWS customer support, but this is not guaranteed.

5 AWS Account Setup

Before getting started on the lab, your team needs an AWS account. Each team member should have an IAM role which has administrator access so every member can access every service required. Even if you already have an AWS account, it is recommended to create a new one because you will be exposing lab credentials to the course staff.

1. Select one team member to create an AWS account for your team. Follow the AWS documentation [Step 1: Set up an AWS account and create a User](#). The very first account you create will be for the root user. As the documentation explains, the root user should not be used unless you are executing an action that requires root access for security reasons. Make sure to enable MFA for your root user and store the password somewhere safe.
2. Go to the [IAM](#) service by searching for it in the search bar. Since we are going to create an **admin user** for every group member, we will create a **User group**. This will allow you to easily update and manage the permissions of all the admin users by editing the group.
 - a. Click User groups > Create group.
 - b. Give the group a meaningful name (ex. “admins”) and attach the policy called “AdministratorAccess” so that the admins have access to all AWS resources as shown below in Figure 1.
 - c. Finally, click the “Create group” button at the bottom of the page.

Name the group

User group name

Enter a meaningful name to identify this group.

Maximum 128 characters. Use alphanumeric and '+,=, @, -, _' characters.

Add users to the group - *Optional* (3) [Info](#)

An IAM user is an entity that you create in AWS to represent the person or application that uses it to interact with AWS. A user can belong to up to 10 groups.

<input type="checkbox"/>	User name ↗	Groups	Last activity	Creation time
<input type="checkbox"/>	jack-admin	0	21 hours ago	21 days ago
<input type="checkbox"/>	laura-test	0	3 hours ago	25 days ago
<input type="checkbox"/>	molly-admin	0	2 days ago	16 days ago

Attach permissions policies - *Optional* (Selected 1/856) [Info](#)

You can attach up to 10 policies to this user group. All the users in this group will have permissions that are defined in the selected policies.


<input type="checkbox"/>	Policy name ↗	Type	Description
<input checked="" type="checkbox"/>	 AdministratorAccess	AWS managed - job function	Provides full access to AWS services and resources.

Figure 1. Sample admin group configuration.

- Next, create an **admin user** for yourself in the IAM service.
 - Click Users > Add user.
 - For the User Details, specify the username and a custom password, and select the options as shown below in Figure 2.
 - For the user permissions, add the user to the previously created admin group, as shown in Figure 3.
 - Review the details and click “Create user”.
 - You should get an email which will ask for a password reset for your user. Do so, and add MFA for the user for extra security.
- Create an admin user for each groupmate following the same steps as for the admin user of the group member who also holds the root account (Step 3).

User details

User name

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

☒ Provide user access to the AWS Management Console - *optional*
 If you're providing console access to a person, it's a [best practice](#) to manage their access in IAM Identity Center.

Are you providing console access to a person?

User type

☐ Specify a user in Identity Center - Recommended
 We recommend that you use Identity Center to provide console access to a person. With Identity Center, you can centrally manage user access to their AWS accounts and cloud applications.

☒ I want to create an IAM user
 We recommend that you create IAM users only if you need to enable programmatic access through access keys, service-specific credentials for AWS CodeCommit or Amazon Keyspaces, or a backup credential for emergency account access.

Console password

☐ Autogenerated password
 You can view the password after you create the user.

☒ Custom password
 Enter a custom password for the user.

Some-password!

- Must be at least 8 characters long
- Must include at least three of the following mix of character types: uppercase letters (A-Z), lowercase letters (a-z), numbers (0-9), and symbols ! @ # \$ % ^ & * () _ + - (hyphen) = [] { }

☒ Show password

☒ Users must create a new password at next sign-in - Recommended
 Users automatically get the [IAMUserChangePassword](#) policy to allow them to change their own password.

If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. [Learn more](#)

Figure 2. Sample User details configuration.

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. [Learn more](#)

Permissions options

☒ Add user to group
 Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

☐ Copy permissions
 Copy all group memberships, attached managed policies, and inline policies from an existing user.

☐ Attach policies directly
 Attach a managed policy directly. In practice, we recommend attaching instead. Then, add the user to the

User groups (1/1)

Search

☒ Group name

Users

<input checked="" type="checkbox"/>	admins	3
-------------------------------------	--------	---

Figure 3. Adding a user to the admins group.

6 Lab Procedure

For this lab, you will be experimenting with 2 different types of databases and 2 different types of communication frameworks to build a web system. This procedure will guide you in building each of the components as well as putting them together.

It is suggested that your team of 4 breaks up into 2 groups of 2, where each group handles building 1 of the database components and 1 of the communication framework components. This way, you can work on components in parallel. Once all of the components are set up, the whole team can put them together in the 4 configurations and deploy.

Database

General setup:

1. Clone the repo [Lab Database Setup](#).
2. Download Python 3.x if you do not have it installed already, along with the dependencies needed to run the Python [main.py](#) script. We have provided a requirements.txt folder. To install all requirements according to this file, use `pip install -r requirements.txt`
3. Create S3 bucket by following the *S3 Bucket Creation* steps below and modify `s3_bucket_name` in [populates3table.py](#) and [populatedynamo.py](#) with the S3 bucket you create.
4. Uncomment the S3 population code in `main.py`.

S3 Bucket Creation (Unused in Lab 1, but good practice and used in lab 2)

1. Go to S3 service in the AWS UI.
2. Click the create bucket button.
3. Set the bucket's region to us-east-1
4. Leave the bucket's default settings, except make the bucket public by unchecking the "Block all public access" box.

Creating an Access Key

1. Click your account name at the top right corner, then select "Security Credentials". Create an access key for "Local code" with any name. This will be used later for your .env files.

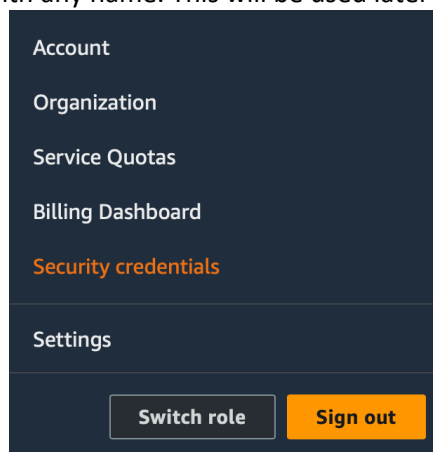


Figure 4. AWS popup menu with Security credentials selected.

Access keys (1)

Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. [Learn more](#)

Create access key

Figure 5. Section of Security credentials page allowing access key creation.

DynamoDB:

1. The database population scripts will create your DynamoDB tables automatically.

RDS:

1. Create an RDS with MySQL instance with the AWS console
2. Choose "Standard create" as the creation method
3. Choose MySQL > MySQL Community edition
4. Choose the Free Tier template
5. For deployment options, only deploy a single DB instance
6. In Settings, give your DB instance a name, and enter a master username and password. Also, uncheck the box so you're not managing master credentials in AWS secrets manager.
7. In Instance Configuration, choose db.t2.micro.
8. For storage, choose a General Purpose SSD with 20 GiB allocated storage, and don't enable storage autoscaling.
9. For connectivity, choose "Don't connect to an EC2 resource". You will do this manually later. Also choose IPV4, Default VPC and set "Public access" to Yes.
10. For database authentication, choose "Password authentication".
11. For the rest of the form, maintain the defaults.
12. Note the estimated monthly cost at the bottom of the form. It should be roughly 14.00 USD. Do not panic, this is presumably the estimated cost of the resource after the free tier ends. If you have a cost that is larger than this, it is possible you did not do the setup for the instance correctly and should review the form.
13. Configure the database locally:
14. Create a .env file in the repo root folder which includes the following keys.

```
# AWS RDS
RDS_HOSTNAME=<your rds endpoint>
RDS_USERNAME=<your rds username>
RDS_PASSWORD=<your rds password>
RDS_PORT=<your rds port>

# AWS DYNAMODB
AWS_ACCESS_KEY_ID=<your rds access key id>
AWS_SECRET_ACCESS_KEY=<your rds secret access key>
AWS_REGION=us-east-1
```

Populating DBs:

1. Run main.py script to generate the objects, store images in s3, and populate DynamoDB/SQL

Setting up Your EC2 Instance with Web Server

1. Fork the [lab skeleton](#) template repository. You will want to add your groupmates to this repository.
2. Create a .env file in the repo root which includes the following keys. This file should have the same contents as the file. the database setup portion above. This will need to be either committed to git or copied to the EC2 instance later:

```
# AWS RDS
RDS_HOSTNAME=<your rds endpoint>
RDS_USERNAME=<your rds username>
RDS_PASSWORD=<your rds password>
RDS_PORT=<your rds port>

# AWS DYNAMODB
AWS_ACCESS_KEY_ID=<your rds access key id>
AWS_SECRET_ACCESS_KEY=<your rds secret access key>
AWS_REGION=us-east-1
```

3. Complete the functions required to interface with the databases in `mysql_db.js` and `dynamo_db.js`. These functions are marked with TODOs. You may test the functionality locally before hosting the server on your EC2 instance.
4. IMPORTANT: read this entire step carefully. Log in to your AWS account, and initialize an EC2 instance, using all of the default options. While provisioning your EC2 instance, create a new key pair. Select the key-pair type based on your machine. The recommended format is .pem, for use with all Unix-based systems and git bash on Windows. Then, allow HTTP connections from the internet, as well as SSH connections from the internet. For more detailed instructions, reference [this medium article](#).
5. SSH into the new EC2 instance. You will need to download the .pem key file you created. You can do this by navigating to the EC2 “Instances” page, and selecting your instance, then selecting “Connect to Instance”. Navigate to the SSH tab, and download the key pair file. You may need to run `chmod 400 <KEY>.pem`
6. Install git and nodejs on the machine by using the following commands:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh |
bash
. ~/.nvm/nvm.sh
nvm install 16
```

```
sudo yum install git -y
git config --global user.name "Your Name"
git config --global user.email "Your Email"
```

```
ssh-keygen
cd .ssh
cat id_rsa.pub
```

7. Add the output of the last command to your Gitlab or Github as an ssh key. Note that this ssh key is required to clone the server onto your EC2 instance, and you may prevent your groupmates from accessing your GitLab by removing the SSH key after you are finished cloning and testing your code.
8. Clone your fork of the server repository onto the EC2 instance.
9. cd into the cloned repository and use npm i to install all required dependencies.
10. Use npm start <rest | grpc> <sql | dynamo> to start a certain configuration of the server.
11. To start multiple configurations of the server in parallel (you will need to do this to run the measurement script) press Ctrl+Z to pause the task, then use the **bg** command to resume the last-paused task in the background. This will let you start multiple applications in parallel.

Testing Your System

1. Clone the [measurement repository](#) onto your local machine.
2. cd to the directory and use npm i to install all dependencies.
3. Get the IP address of your EC2 instance from the AWS console. Provide this IP to main.js. Add http:// to the rest endpoint, but do not for the gRPC endpoint. Note that you must also specify the ports. By default, these are configured as 3000 for the REST endpoint, and 3001 for gRPC.
4. EC2 > Security Groups > [security group] -> Edit inbound rules
5. Create a new rule for All TCP, source: anywhere ipv4
6. Configure the number of iterations to run each test within main.js. To complete the report, use 3 iterations.
7. Use npm start to run the tests. Record your results.

Questions

Starter Component Combinations

- 1) Build the web system using the Express and RDS + MySQL components.
 - a) Test your code out with the front end and provide the runtimes for each of the test cases.
- 2) Build the web system using the Express and DynamoDB components.
 - a) Test your code out with the front end and provide the runtimes for each of the test cases.
- 3) Build the web system using the gRPC and RDS + MySQL components.
 - a) Test your code out with the front end and provide the runtimes for each of the test cases.
- 4) Build the web system using the gRPC and DynamoDB components.
 - a) Test your code out with the front end and provide the runtimes for each of the test cases.

Analysis & Reflection

1. Analyze the performance of the 4 starter component combinations. Describe any differences you observe between the performance of the combinations and briefly justify why each of these differences may be occurring. If you do not see significant performance differences, briefly justify why this may be, and predict and justify what would work best at a large scale.
2. Pick one of the 4 systems and draw an architecture diagram for it.

3. What are the advantages and disadvantages of working with gRPC and REST?
4. Describe the main differences between DynamoDB and MySQL in terms of data modeling, scalability, and querying abilities.
5. What are the advantages and disadvantages of working with DynamoDB and MySQL?
6. Discuss potential scalability issues with each database and suggest strategies to handle increased traffic and load. (Why is this system not production ready?)

Deliverables

Submit the following to a dropbox on LEARN:

1. The completed lab skeleton, in .zip format with proper documentation/comments. Insufficient documentation will result in marks lost.
2. A lab report with:
 1. Table of contents
 2. Answers to all the questions asked above, with citations if external sources were used.