UNIVERSITY OF
WATERLOO

SE464 Lab 2 Packet

Revised Fall 2023

Learning Objectives

After completing Lab 2, students will:

- Be able to set up load balancing for a web system.
- Understand how to make a system more reliable, scaleable, and robust.
- Have analyzed tradeoffs of using various load balancing methods.
- Understand the basics of application monitoring and observability.

Background Information

Lab 1 had your team build a very primitive web service, containing the minimum components to realize a very simple online shopping application. Typically, web services do not just have one **web server** and/or one **application server** when many users are expected to be using the service simultaneously. In order to be able to handle so much traffic and process and return the correct data quickly and reliably, multiple copies of these servers need to exist in the system.

In order to service user requests and distribute them evenly amongst the servers, a **load balancer** is typically employed. The load balancer is placed in front of the servers and directs and distributes traffic between users / clients and the servers. This has a primary advantage of making sure that all servers are utilized relatively equally, and by extension, that no server(s) end up being overwhelmed with traffic. Beyond this, implementing load balancing into a system allows for it to be more available, scaleable, and secure. To learn more about the benefits of load balancing and types of load balancing, see the AWS article *What Is Load Balancing?*

Another way to handle increased load in a system is parallelism. It is possible to multi-thread servers to handle requests on all CPU threads. Apart from writing our own multi-threaded server applications, this is easily achievable using a node process manager such as PM2. For more information, see the PM2 documentation.

In modern day applications, monitoring and observability have become crucial for building and maintaining robust, reliable, and performant systems. As software systems become increasingly complex and distributed, simply monitoring a console or keeping log files are insufficient in diagnosing performance issues and system errors.

Observability is the ability to gain insight into the internal state of an application, allowing developers to understand its behavior, diagnose issues, and optimize performance. It encompasses three key pillars: metrics, logs, and traces.

1. **Metrics** provide quantitative data about application state, including resource (CPU, memory) utilization, network calls, response times, and error rates. They are essential for tracking the health and performance of systems over time.
2. **Logs** allow us to record an application's activities and events, allowing teams to understand what happened and why. Logs are important for debugging, compliance, and auditing.

3. **Traces** provide end-to-end visibility into operations as they traverse a complex system, enabling the identification of bottlenecks and latency issues.

Overview

In this lab, your team will expand on the system you built in Lab 1 in order to improve it, specifically making it more robust, scalable, and reliable. This will in part be achieved by implementing load balancing in two different ways. First, your team will be introduced to basic application observability and monitoring through AWS CloudWatch, using its metrics dashboards and logging.

Second, your team will implement PM2, which can automatically multithread a node.js application. This will demonstrate the value of optimizing the performance of an application's backend; in this case, parallelizing it.

Third, your team will use AWS application load balancers to balance load across a second EC2 instance, which will be a duplicate of the first. This represents physically scaling your hardware systems to increase server capability.

Prerequisite

- Lab 1 completed

Procedure

Important note: The procedure for Lab 2 only uses the REST + SQL configuration of your work from Lab 1. **Deliverables for the lab report are highlighted in blue.**

Application Monitoring and Observability using AWS CloudWatch

Note: *Please read through and ensure that your metrics and logging usage are within the CloudWatch limits, as detailed here.*

1. In this lab, we will use the winston logging library. Install the packages by running the following command: `npm i winston winston-cloudwatch aws-sdk`
2. Create a new file `logger.js` at the root of your lab repository (lab-skeleton) by copying the contents of the file below:

```
const winston = require('winston');
const WinstonCloudWatch = require('winston-cloudwatch');

// Create a Winston logger
const logger = winston.createLogger({
    level: 'info', // Set the desired log level
    format: winston.format.json(),
    defaultMeta: { service: 'express-app' }, // Customize as needed
    transports: [
        // Output logs to the console for local development
        new winston.transports.Console(),

        // Send logs to CloudWatch Logs
        new WinstonCloudWatch({
            logGroupName: 'server-logs',
            logStreamName: 'server-logs',
            awsRegion: 'us-east-1', // Shouldn't have to replace this
            jsonMessage: true, // Ensure logs are formatted as JSON
        }),
    ],
});

module.exports = logger;
```

3. In the AWS Console, navigate to CloudWatch and go to Logs → Log Groups. Create a new log group with the default options **named 'server-logs' to match the contents of *logger.js***.

4. In the Log groups page, click on the log group you just created. Click on the Create log stream button near the bottom right of the page.

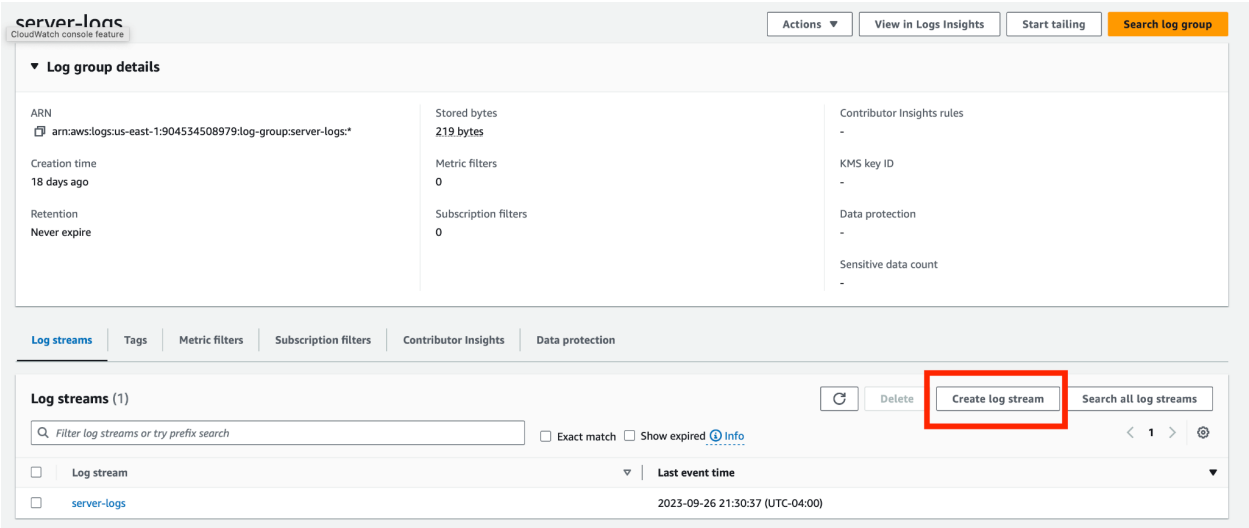

Figure 1. Location of the Create log stream button in the Log group page for the server-logs Log group.

5. Create a log stream **also named 'server-logs' to match the contents of *logger.js*.**
6. Replace your `console.log(...)` statements with `logger.info(...)`. You should be doing this in two files: restServer.js and mysql_db.js.
7. In controller.js, add appropriate logging statements for all your endpoints using `logger.info(...)`.
    a. At minimum, implement a log statement for an incoming request, as well as the total number of requests that your server has received. This log should include the query string/parameters that came with the request.
    b. **(OPTIONAL)**: You might notice that the REST starter code already includes the Morgan logger. Feel free to integrate this with Winston, as Morgan does a lot of the necessary logging automatically! Refer to this link for instructions.
8. Remember to import your logger in each file using it:
   `const logger = require("./logger");`
9. If you have done this work on your local machine, make sure to push it to your git repository, and pull the new code on your EC2 instance.
10. **Run the benchmarker unchanged from lab 1** and provide a screenshot of the logs in CloudWatch, similar to *figure 2* below. Try filtering your logs using a keyword (e.g."Database"). Include this screenshot in your lab report.

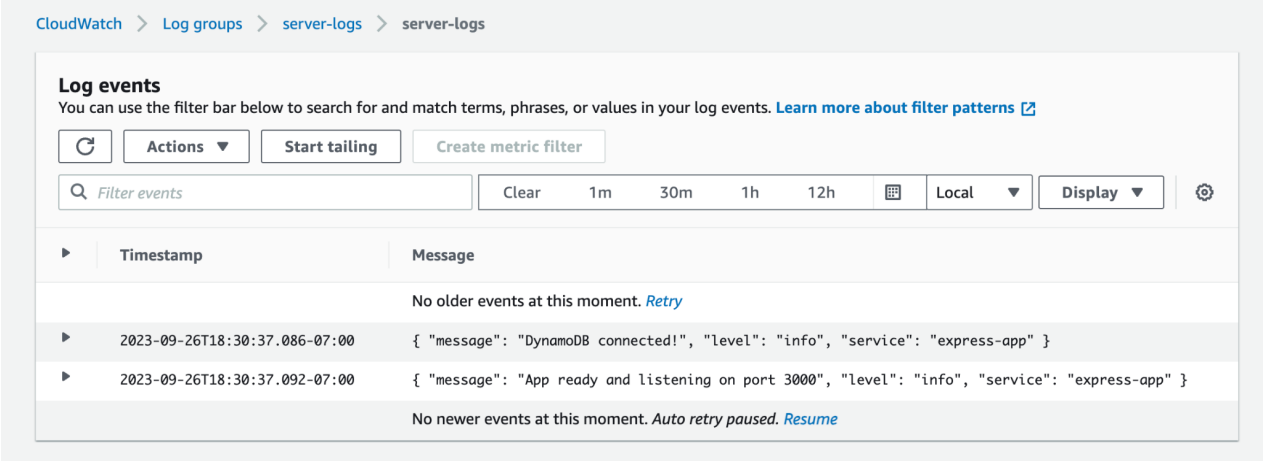

Figure 2. Example CloudWatch logs. These show DynamoDB logs but are just an example. Your completed lab will only log for REST/SQL.

11. Navigate to the CloudWatch homepage and click Create a default dashboard → Create Dashboard.
12. Add **line graphs** for EC2 CPU Utilization and EC2 Status Check Failed to your dashboard by clicking the + button in the top right corner of the page. When prompted, select *metrics* instead of *logs*.
13. Add a **stacked area graph** for EC2 Network In, Network Out.
14. Add a Number widget to display current RDS CPU utilization for Across All Databases.
15. (OPTIONAL) Feel free to add other metrics to the dashboard if you'd like! Another interesting option could be a number widget to monitor EC2 instance CPU utilization live.
16. Set the time range to 4 weeks and take a screenshot of your dashboard. See example below in *figure 3*.

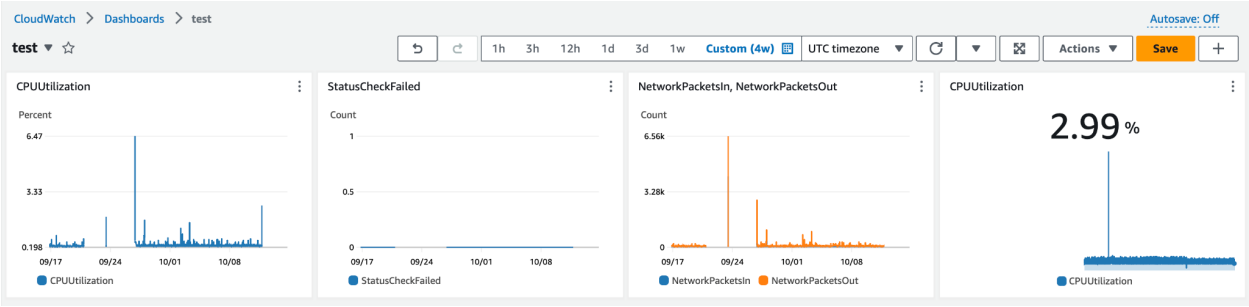

Figure 3. Example CloudWatch dashboard.

Multithreading Your Application Server with PM2

1. SSH into your EC2 instance from Lab 1.
2. Install PM2 globally using `npm i -g pm2`
3. Create a new file named `ecosystem.config.js` at the **root of your server repo** (lab-skeleton) and add the following contents:

```
module.exports = {
    apps : [{
        name   : "se464-lab",
        script : "./main.js",
        args   : "rest sql",
        instances: 2,
        exec_mode: "cluster",
        env: {
            port: 3000
        }
    }]
}
```

4. Start the server using `pm2 start ecosystem.config.js`
5. Monitor the logs of each thread of the server by using `pm2 monit`
6. Set the lab benchmarker to **100 iterations**, and **comment out the gRPC tests**. You should notice that the requests are distributed across both instances, and a marked improvement in performance over a non-multithreaded server.
7. Take screenshots of the `pm2 monit` terminal window to show requests coming in to both threads. Include these screenshots in your report.

PM2 Report Questions

1. Record the results of 100 iterations within the benchmarker **without** using PM2 multithreading.
2. Record the results of 100 iterations within the benchmarker **with** PM2 multithreading on 2 instances. On average, what is the % improvement?
3. Record the number of requests each thread receives. What method does PM2 use to balance incoming requests between threads and is this reflected in the results?
4. Explain the causes of some load-related issues using PM2 might solve. Additionally, provide an example of an issue that using PM2 would not solve.
5. Check and compare on CloudWatch how CPU utilization has/has not changed, and briefly explain why. Provide a screenshot of the CPU utilization as part of your explanation.

Load Balancing with AWS EC2 Load Balancer

1. Go to the EC2 service in the AWS console. Make sure the region is set to us-east-1 before moving forward.
    a. Follow the steps from Lab 1 section “Setting up your EC2 instance with a web server” to set up a second EC2 instance to mirror the one you set up in Lab 1, in the same availability zone as your other instance. Before moving on, make sure you have 2 EC2 instances up and running.
2. Click “Load balancers” in Resources and click the “Create load balancer” button.
3. Choose “Application Load Balancer”.
4. Choose Scheme to be Internet-facing, and the IP address type to be IPv4.
5. For the Network Mapping section, choose the default VPC and choose 2 availability zones based on where your EC2 instances are deployed. I.e. choose an availability zone to be the zone where you deployed your EC2 instances or else you will not be able to route traffic to them.
    - An **availability zone** is a location in an AWS region that is isolated from other availability zones in the region. This helps ensure that if one availability zone has a failure, the others are unaffected. Deploying in multiple availability zones hence results in higher availability and robustness. To read more about availability zones and AWS regions, see the associated AWS documentation.
6. For the Security groups section, choose the security group you made in Lab 1 to allow all traffic.
7. For Listeners and routing, set the protocol and port as HTTP and port 3000 and click create a new target group for the Default action.
    a. For the Basic configuration of the target group, select Instances, Protocol → HTTP and Port → 3000. Also select the default VPC and HTTP1 as the Protocol version.
    b. Leave the rest as is, and click Next at the bottom of the page.
    c. Select the EC2 instances you have created so the load balancer can route to them. Make sure to click the “include as pending below” button or else your target group will be empty. See Figure 4 for an example.

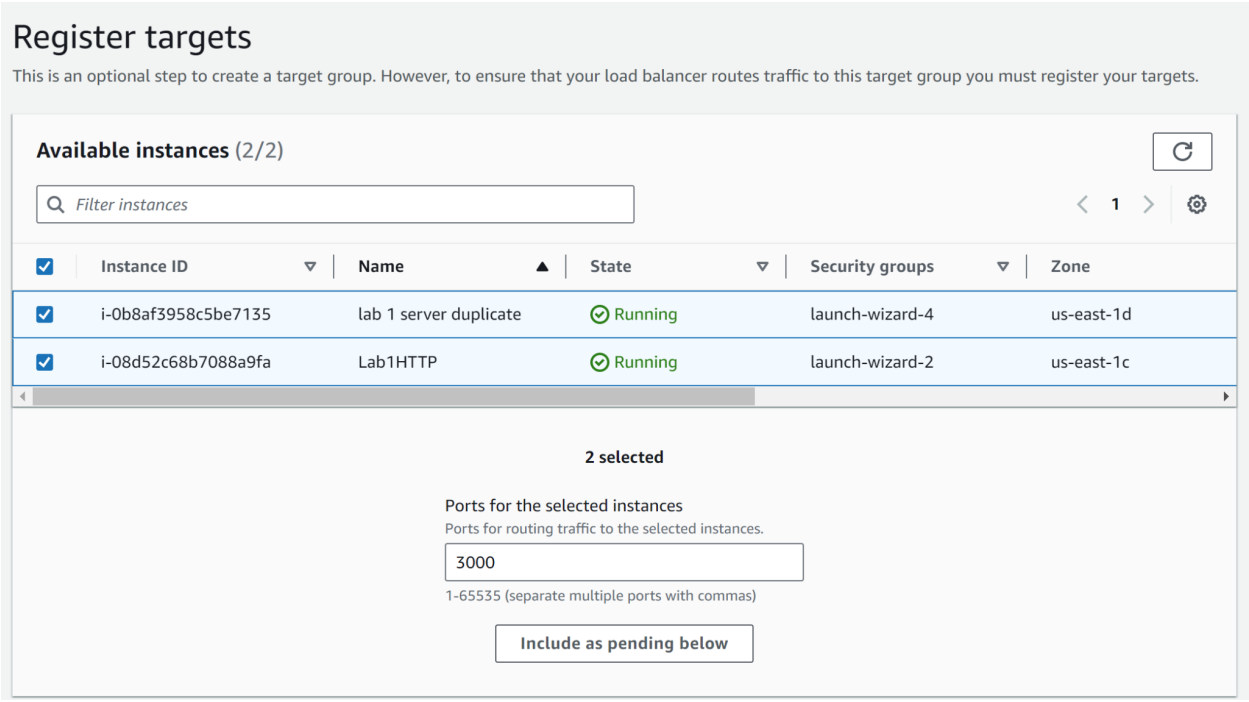

Figure 4. Registering targets for the load balancer.

d. Leave the rest as is and click Create target group.
e. Return to the Create load balancer page and click the refresh button to find and add the target group you just created. See Figure 5 below for an example.

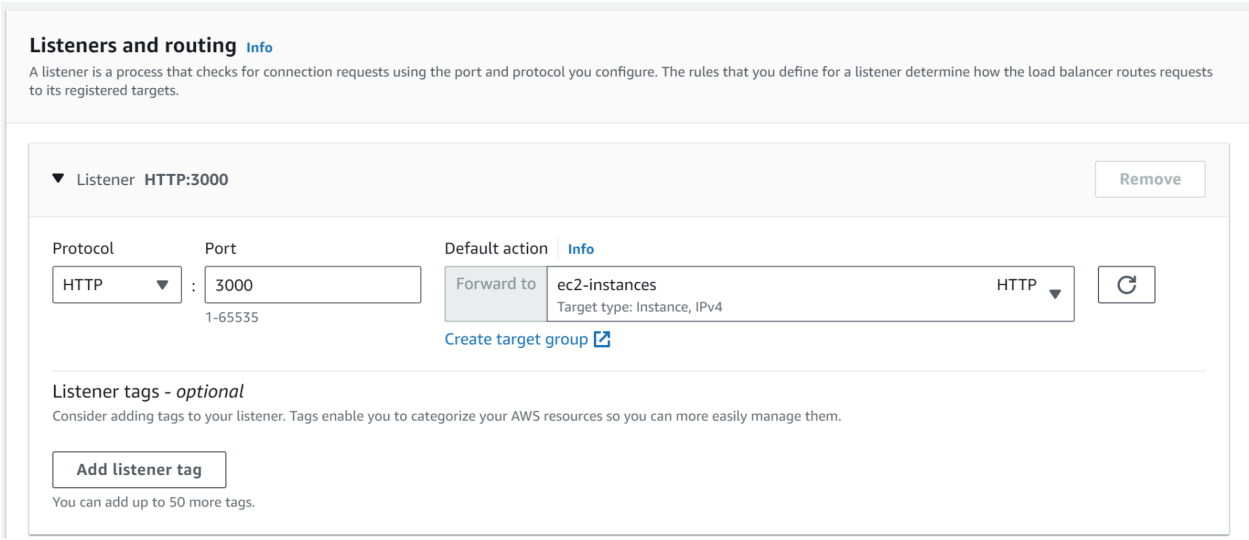

Figure 5. Setting up the created target group for the load balancer.

8. Review the summary, it should look like Figure 6 below.

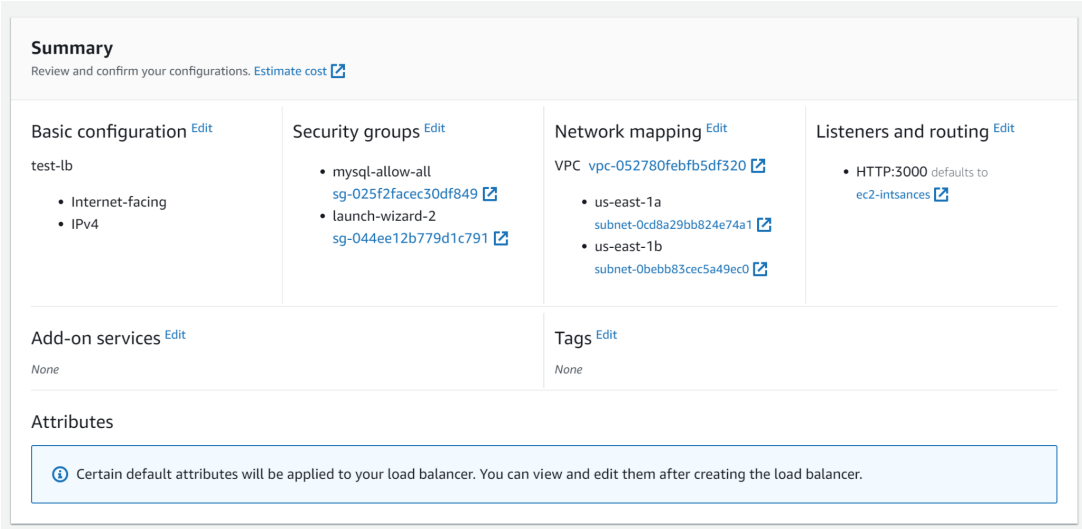

Figure 6. Sample load balancer configuration.

9. To find the public IP addresses of the load balancer in the 2 availability zones, on the left main sidebar click Network & Security > Network Interfaces.
    - The loadbalancer has 2 IP addresses since AWS deploys it in 2 availability zones. **Provide the benchmarker with the IP** that matches the availability zone where you deployed your EC2 instances when you run the test suite. If you do not change the IP of the benchmarker, you will be sending requests straight to a server and not through the load balancer.
10. SSH into both of your EC2 instances. **Start the REST servers WITHOUT the PM2 command above, instead run the same command you did in Lab 1 (`npm start rest sql`)**
11. Run the lab benchmarker from your local machine. Record your results. Include these results in your lab report.
12. Go back to your EC2 instances and stop the servers. Now, **run them WITH the PM2 command from above**.
13. Run the lab benchmarker from your local machine. Record your results. Include these results in your lab report.

Report Load Balancing Related Questions

1. Once you have confirmed that both EC2 instances are receiving requests and the system is working as expected, run the REST API test suite on the benchmarker. Record the results.
2. Compare the results you recorded in procedure steps 11 and 13 above. What was the performance difference?
3. How might you expect the performance to increase as the number of EC2 instances behind the load balancer and the traffic scales?

General Report Questions

1. Explain the difference between monitoring and observability, and describe the importance of both in large-scale, production-level applications.
2. Briefly describe what traces are and why they are important. Additionally, In a web system, provide an example of where traces might be valuable.
3. Provide three commonly used metrics and what main traits they indicate in a system (e.g. for performance/availability) and why these traits need to be monitored.
4. What are the different log levels within Winston? Briefly provide an example of what you might log at each level.
5. What would be some of the consequences of logging too much? What about logging too little?
6. Compare the performance between PM2 multithreading and the load balancer multiplexing multiple EC2 instances on their own (not in use at the same time). Either explain the difference, or provide some reasoning as to why the improvement is the same.
7. Which method would you use in a real-world setting for a web server application? In particular, consider what method you would use when you have a large number of users for your application vs. what you would use when your application has a small number of users. Also consider what you might use if you have limited hardware resources.

Deliverable Checklist:

- Lab report, in PDF format (**15 questions total, 10 marks each**)
- Screenshot of Log Stream in AWS CloudWatch
- Screenshot of AWS CloudWatch dashboard
- Screenshots of `pm2 monit` console showing requests incoming to each thread