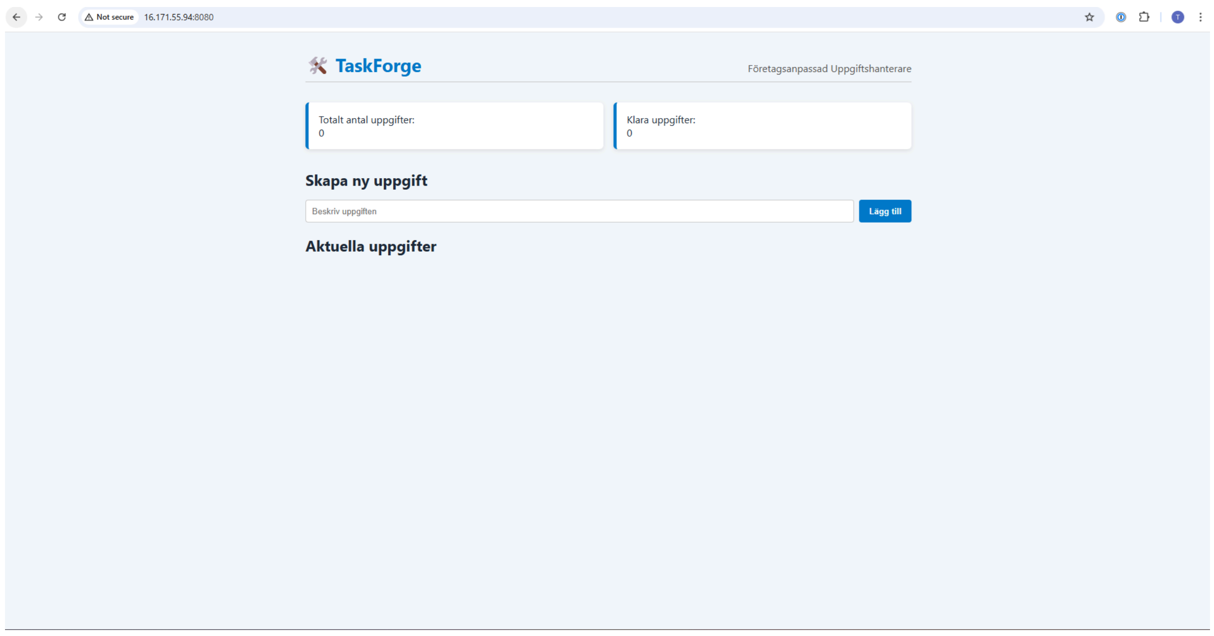
Project Documentation – Cloud-Based TodoApp for TaskForge AB (a imaginary company)

1. Introduction

This project is a cloud-based web application developed using **Spring Boot** and deployed on **Amazon Web Services (AWS)**. The application is a simple yet scalable todo list designed for the fictional company, TaskForge AB to help teams manage their tasks collaboratively. It's built with a focus on responsiveness, scalability and reliability.



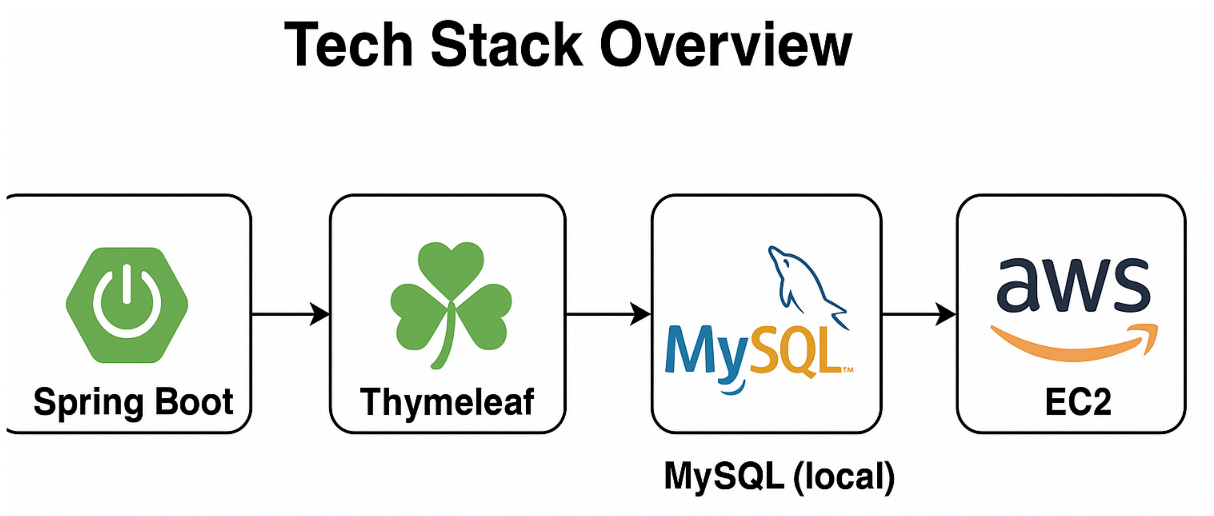
Totalt antal uppgifter: 0

Klara uppgifter: 0

Screenshot of the application running in the browser (the language used on the application is Swedish).

2. Technical Overview

Component	Technology	Purpose
Backend	Spring Boot	Handles logic and routing
UI	Thymeleaf	Server-side rendered dynamic HTML
Database	MySQL(local)	Stores todo tasks
Deployment	AWS EC2	Hosting the application
Service Manager	systemd	Ensures auto-restart on reboot
Load balancing	Application Load Balancer	Distributes incoming traffic
Autoscaling	Auto Scaling Group	Automatically adjusts instance count
Backup	AMI	To quickly launch identical instances



Practical explanation for a company

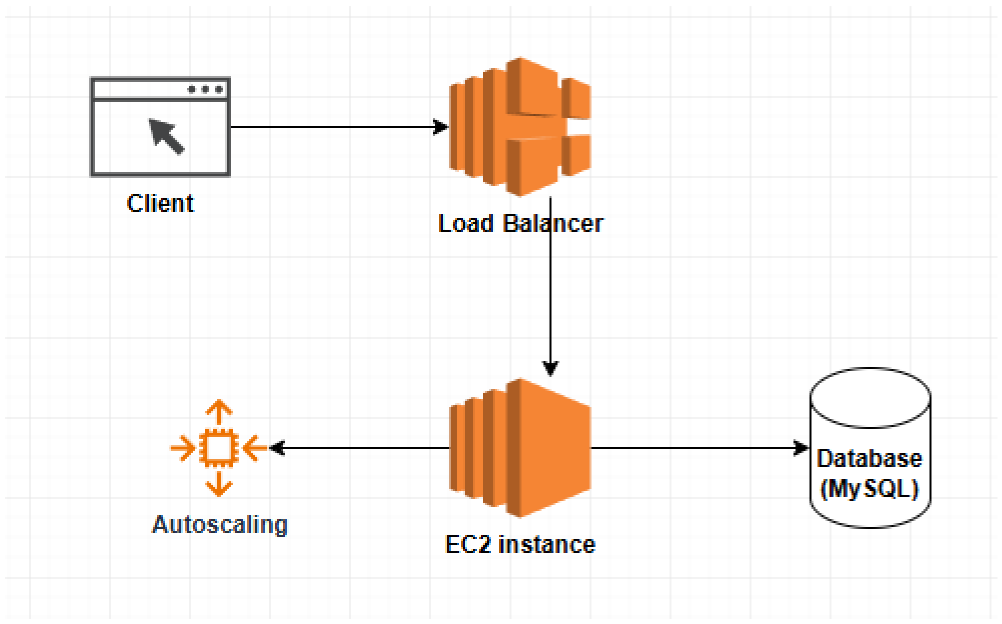
Imagine that your business need a internal app where employees can write daily tasks and where the system is simple and easy to maintain, this app makes sure:

- It automatically grows when many people use or try to access it at the same time.
- Users can access it with a single link and it always works.
- If the unfortunate thing happens and a server crashes another one takes over automatically so the system continues to run.
- All the information is stored and kept safe on the server.
- You never have to manually fix or start the servers/systems it does by itself.

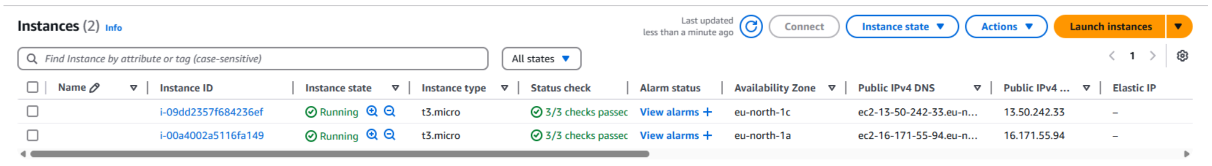
This setup is great for a small to medium sized business that wants a reliable and scalable tool without having to hire a big IT department.

3. AWS Infrastructure

The application runs inside an EC2 instance launched via an Auto Scaling Group. Incoming traffic is routed through an Application Load Balancer which forwards requests to a Target Group configured to communicate on port 8080 (standard port for Spring). The Spring Boot app runs as a background systemd service, MySQL runs locally on each EC2 instance.



Architecture diagram of the structure in AWS.

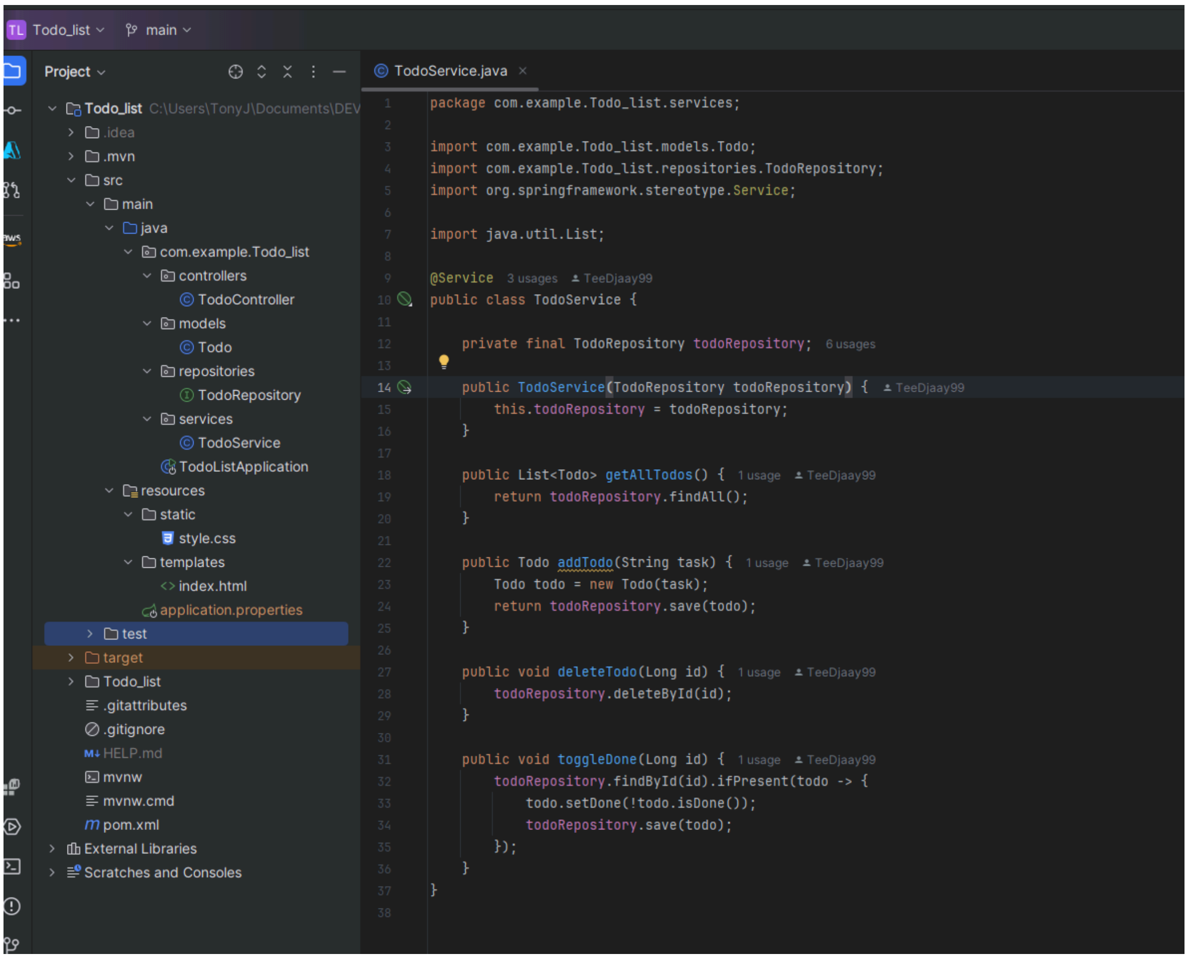

Instances (2) Info
Last updated less than a minute ago
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
1

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
	i-09dd2357f684236ef	Running	t3.micro	3/3 checks passec	View alarms +	eu-north-1c	ec2-13-50-242-33.eu-n...	13.50.242.33	–
	i-00a4002a5116fa149	Running	t3.micro	3/3 checks passec	View alarms +	eu-north-1a	ec2-16-171-55-94.eu-n...	16.171.55.94	–

The running instances because the Auto scaling is set to a minimum of 2 instances and a maximum of 3 instances.

4. Deployment Steps

1. The Spring Boot app was generated using Spring Initializr.
2. MySQL was installed on the EC2 instance and configured.
3. The .jar file was built using Maven.
4. File was uploaded to EC2 using **FileZilla(SFTP)**.
5. A **systemd service** was created for automatic startup on boot.
6. An **AMI** was created from the instance.
7. A **Launch Template** was created based on the AMI.
8. An **Auto Scaling Group** and **Application Load Balancer** were set up.



Todo_list · main

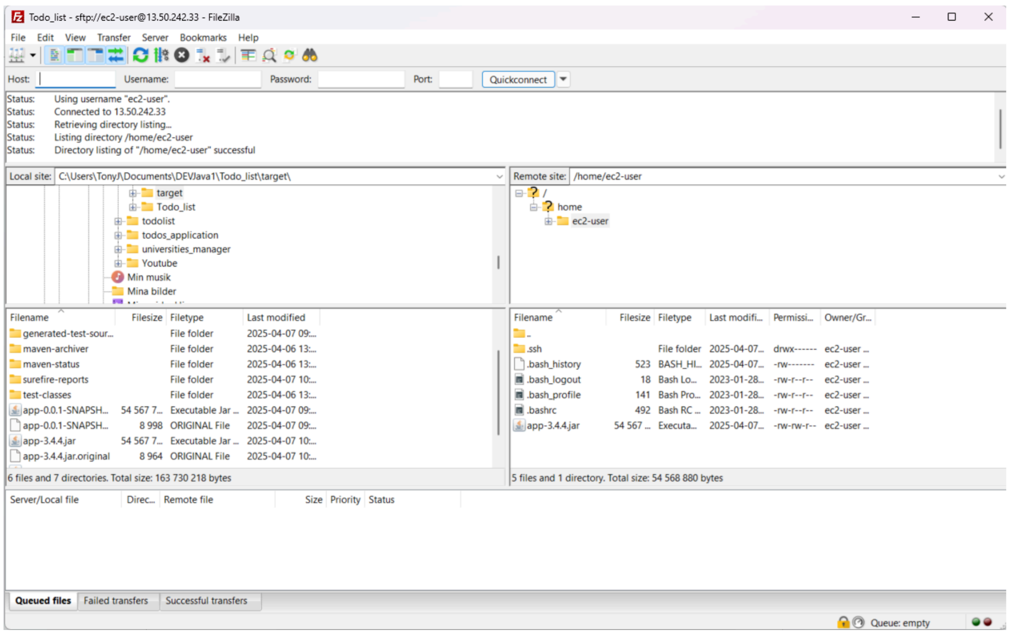
Project

```
Todo_list C:\Users\TonyJ\Documents\DEV
  .idea
  .mvn
  src
    main
      java
        com.example.Todo_list
          controllers
            TodoController
          models
            Todo
          repositories
            TodoRepository
          services
            TodoService
          TodoListApplication
      resources
        static
          style.css
        templates
          index.html
        application.properties
    test
  target
  Todo_list
  .gitattributes
  .gitignore
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
External Libraries
Scratches and Consoles
```

TodoService.java

```java
package com.example.Todo_list.services;

import com.example.Todo_list.models.Todo;
import com.example.Todo_list.repositories.TodoRepository;
import org.springframework.stereotype.Service;

import java.util.List;

@Service  3 usages  TeeDjaay99
public class TodoService {

    private final TodoRepository todoRepository;  6 usages

    public TodoService(TodoRepository todoRepository) {  TeeDjaay99
        this.todoRepository = todoRepository;
    }

    public List<Todo> getAllTodos() {  1 usage  TeeDjaay99
        return todoRepository.findAll();
    }

    public Todo addTodo(String task) {  1 usage  TeeDjaay99
        Todo todo = new Todo(task);
        return todoRepository.save(todo);
    }

    public void deleteTodo(Long id) {  1 usage  TeeDjaay99
        todoRepository.deleteById(id);
    }

    public void toggleDone(Long id) {  1 usage  TeeDjaay99
        todoRepository.findById(id).ifPresent(todo -> {
            todo.setDone(!todo.isDone());
            todoRepository.save(todo);
        });
    }
}
```

View of the project structure and the TodoService class.



Todo_list - sftp://ec2-user@13.50.242.33 - FileZilla

File Edit View Transfer Server Bookmarks Help

Host: Username: Password: Port: Quickconnect

```
Status:     Using username "ec2-user".
Status:     Connected to 13.50.242.33
Status:     Retrieving directory listing...
Status:     Listing directory /home/ec2-user
Status:     Directory listing of "/home/ec2-user" successful
```

Local site: C:\Users\TonyJ\Documents\DEVJava1\Todo_list\target\

```
target
Todo_list
todolist
todos_application
universities_manager
Youtube
Min musik
Mina bilder
```

Filename	Filesize	Filetype	Last modified
generated-test-sour...		File folder	2025-04-07 09:...
maven-archiver		File folder	2025-04-06 13:...
maven-status		File folder	2025-04-06 13:...
surefire-reports		File folder	2025-04-07 10:...
test-classes		File folder	2025-04-06 13:...
app-0.0.1-SNAPSH...	54 567 7...	Executable Jar ...	2025-04-07 09:...
app-0.0.1-SNAPSH...	8 998	ORIGINAL File	2025-04-07 09:...
app-3.4.4.jar	54 567 7...	Executable Jar ...	2025-04-07 10:...
app-3.4.4.jar.original	8 964	ORIGINAL File	2025-04-07 10:...

6 files and 7 directories. Total size: 163 730 218 bytes

Remote site: /home/ec2-user

```
/
  home
    ec2-user
```

Filename	Filesize	Filetype	Last modifi...	Permissi...	Owner/Gr...
..					
.ssh		File folder	2025-04-07...	drwx------	ec2-user ...
.bash_history	523	BASH_HI...	2025-04-07...	-rw-------	ec2-user ...
.bash_logout	18	Bash Lo...	2023-01-28...	-rw-r--r--	ec2-user ...
.bash_profile	141	Bash Pro...	2023-01-28...	-rw-r--r--	ec2-user ...
.bashrc	492	Bash RC ...	2023-01-28...	-rw-r--r--	ec2-user ...
app-3.4.4.jar	54 567 ...	Executa...	2025-04-07...	-rw-rw-r--	ec2-user ...

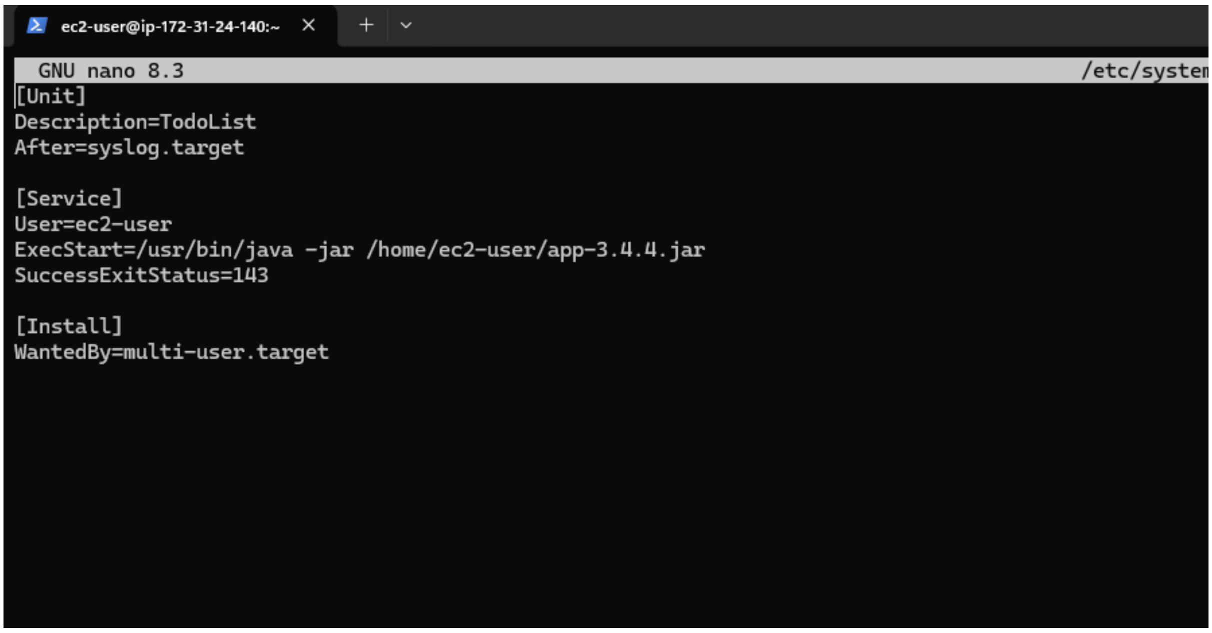
5 files and 1 directory. Total size: 54 568 880 bytes

Server/Local file	Direc...	Remote file	Size	Priority	Status

Queued files · Failed transfers · Successful transfers

Queue: empty

The FileZilla view of how i uploaded the .jar file.

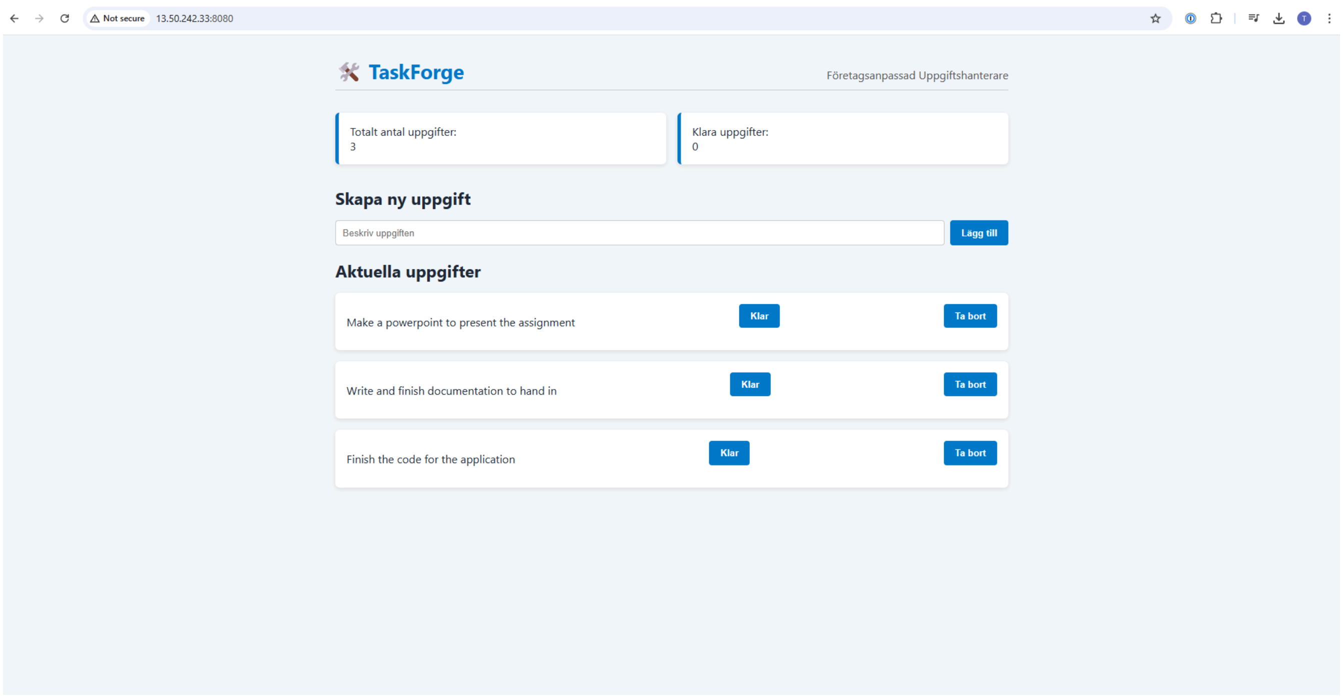

ec2-user@ip-172-31-24-140:~

```
  GNU nano 8.3                                                                    /etc/systen
[Unit]
Description=TodoList
After=syslog.target

[Service]
User=ec2-user
ExecStart=/usr/bin/java -jar /home/ec2-user/app-3.4.4.jar
SuccessExitStatus=143

[Install]
WantedBy=multi-user.target
```

Systemd config file.

5. Application Features

- Displays all tasks directly on the homepage.
- Ability to add new todos.
- Mark tasks as completed or uncompleted.
- Delete tasks.
- Shows statistics: totals tasks and completed tasks.
- Responsive UI with a clean design.



TaskForge

Företagsanpassad Uppgiftshanterare

Totalt antal uppgifter:
3

Klara uppgifter:
0

Skapa ny uppgift

Beskriv uppgiften

Aktuella uppgifter

Make a powerpoint to present the assignment

Write and finish documentation to hand in

Finish the code for the application

The main view of the application showing the todo form and listed tasks.

6. Future Improvements

To further develop this solution, i would consider:

- Migrating the database to **Amazon RDS** for central storage.
- Adding authentication and user management.
- Serving the app via HTTPS for security.
- Adding different language options (only available in Swedish for now).
- Monitoring performance with for example CloudWatch.

7. Learning & Reflection

This project provided hands-on experience in:

- Cloud-based backend development.
- Setting up infrastructure in AWS (a little overwhelming in the beginning with what service to use and keeping track of always terminating everything correctly when done so the billing doesn't skyrocket).
- Understanding how auto scaling and load balancing works.
- Connecting backend applications to databases.
- Learning and using the Terminal/Powershell.

This project helped me connect theoretical cloud concepts to a working, scalable solution.
I now feel a little more confident in building, deploying and maintaining a cloud environment on **AWS**.

Multi-Cloud Capability and Comparison (AWS, Azure, GCP)

In addition to **AWS** I also did some research on how this application could be deployed on **Google Cloud Platform (GCP)** and **Microsoft Azure.**

Google Cloud Platform (GCP)

Required services:

- Compute Engine (virtual machines)
- Cloud Load Balancer
- Cloud SQL (MySQL)
- Instance Groups + Autoscaler
- Cloud Storage

Pros:

- Great autoscaling capabilities.
- Good integration with Kubernetes/container integration.
- Good networking infrastructure.

Cons:

- Smaller community and ecosystem compared to AWS and Azure.
- Can be a little more difficult to learn especially for beginners.
- Less availability in some regions (the Nordics/Scandinavia) which can affect latency depending on where you are trying to access it from.

Microsoft Azure

Required services:

- Azure Virtual Machine - Azures answer for EC2
- Azure Load Balancer
- Azure Database for MySQL
- Azure Virtual Machine Scale Sets (for autoscaling)
- Azure Blob Storage - Like AWS S3

Pros:
- Great integration with other Microsoft services like Office 365, Teams etc..
- User friendly interface and easy to use (AWS is more terminal work sometimes).
- Good CI/CD (Continuous Integration/ Continuous Delivery) tools.

Cons:
- The costs can be a little higher than AWS in some regions.
- More GUI-driven (Graphical User Interface) and less terminal (can be good and can be bad sometimes).
- Less “hands-on” controls compared to AWS.

Summary for End users:

All three platforms AWS,Azure,GCP offer great performance and high availability which makes it that the end user has access to the service regardless of which platform it's deployed/running on.
The biggest difference is not noticed by the end user, it’s more of how the service is set up and managed “behind the scenes”.

AWS is best suited if you need or want full control over how the application is run and scaled.

Azure is the best choice for companies already using the Microsoft ecosystem, since it’s very compatible with for example Teams and Office 365.

GCP is the best for modern solutions that use containers and/or Kubernetes, a good choice for automated or advanced systems.

so the end user rarely notices what platform is being used but the choice what platform to use affects how simple and effectively the application can be developed, deployed and scaled.

Bonus: Attack Test with Burp Suite

Purpose:

The purpose of this test was to simulate a load test and to see if the application could handle a large amount of simultaneous requests of new todos.

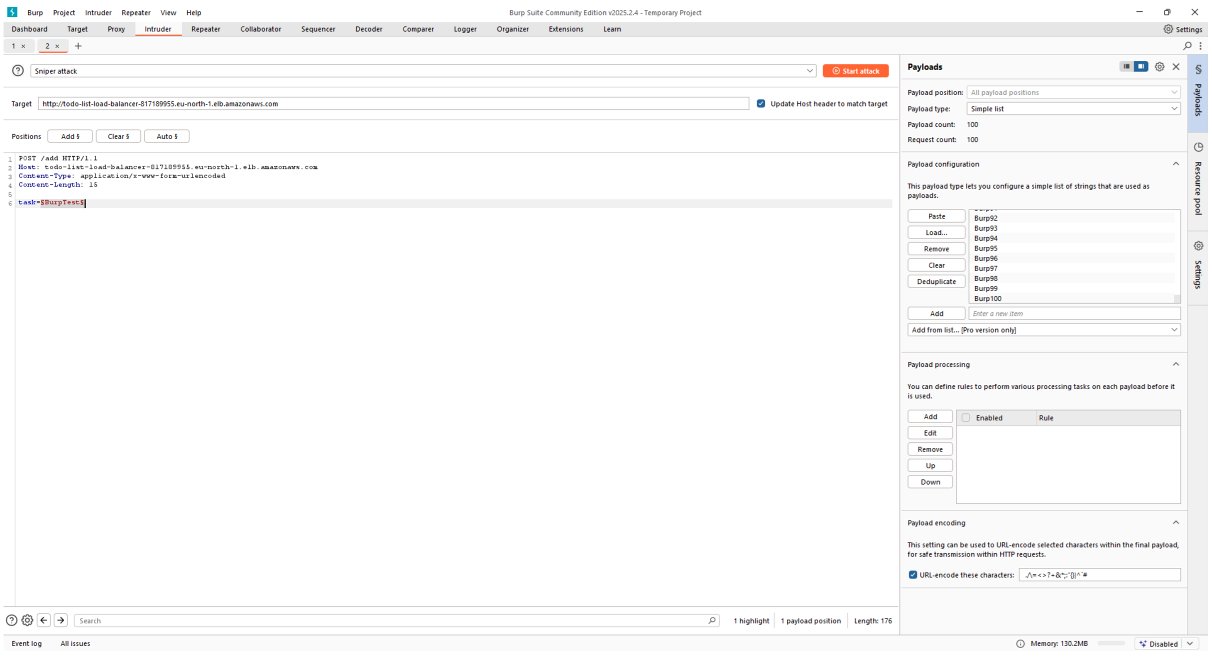
Tools:

- Burp Suite Community Edition
- Repeater and Intruder - modules
- Target: Spring Boot - app on a EC2 behind a AWS Load Balancer

Execution:

1. A POST-call with task=BurpTest was sent via Repeater and was verified to work.
2. The request was sent to Intruder with a payload-list of (Burp1 - Burp100).
3. The Server handled the 100 calls.
4. Status code 302 showed that every request was redirected correctly.

Screenshots:

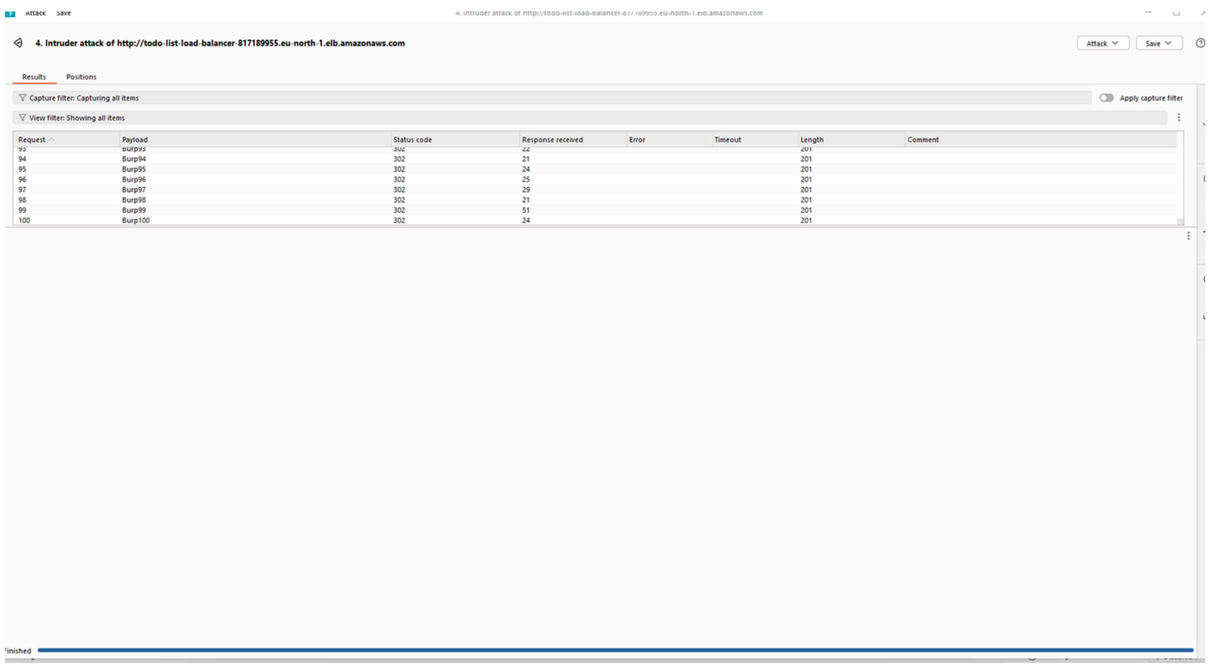

Sniper attack

Target: http://todo-list-load-balancer-817189955.eu-north-1.elb.amazonaws.com

```
POST /add HTTP/1.1
Host: todo-list-load-balancer-817189955.eu-north-1.elb.amazonaws.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

task=§BurpTest§
```

Payload position: All payload positions
Payload type: Simple list
Payload count: 100
Request count: 100

Image of the Burp Suite Intruder dashboard before the test.

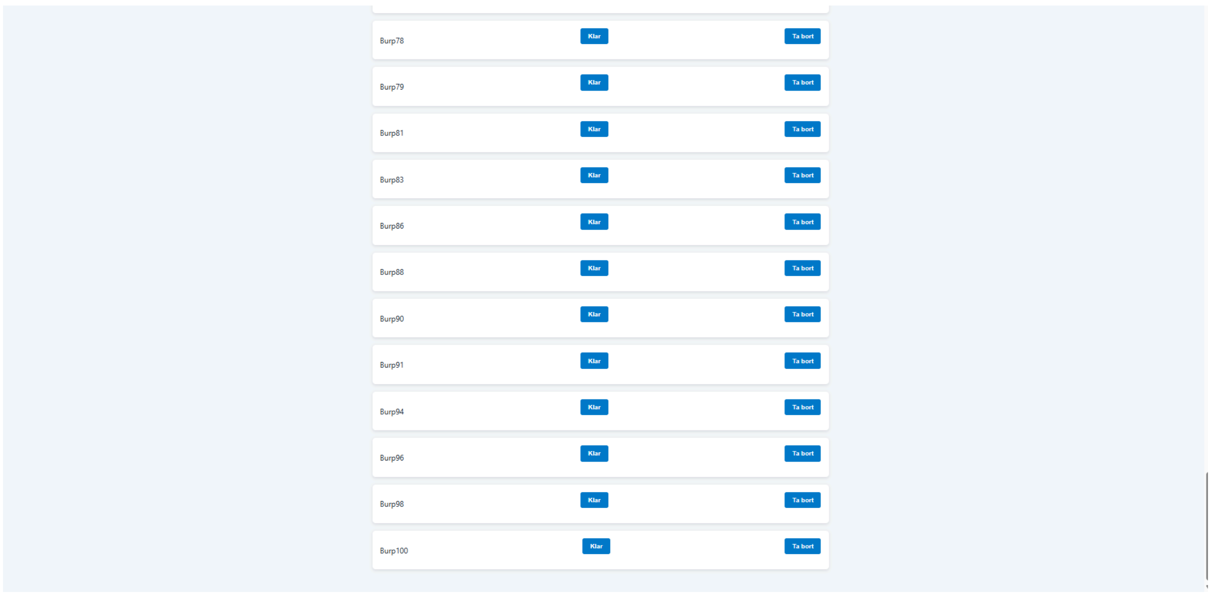

4. Intruder attack of http://todo-list-load-balancer-817189955.eu-north-1.elb.amazonaws.com

Capture filter: Capturing all items
View filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
93	Burp93	302	22			201	
94	Burp94	302	21			201	
95	Burp95	302	24			201	
96	Burp96	302	25			201	
97	Burp97	302	29			201	
98	Burp98	302	21			201	
99	Burp99	302	51			201	
100	Burp100	302	24			201	

Finished

A part of the requests and the 302 status code that shows it redirected the calls correctly.



Burp78	Klar	Ta bort
Burp79	Klar	Ta bort
Burp81	Klar	Ta bort
Burp83	Klar	Ta bort
Burp86	Klar	Ta bort
Burp88	Klar	Ta bort
Burp90	Klar	Ta bort
Burp91	Klar	Ta bort
Burp94	Klar	Ta bort
Burp96	Klar	Ta bort
Burp98	Klar	Ta bort
Burp100	Klar	Ta bort

Some of the 100 requests that were made by Burp Suite to the website.

Result:

- The application continued to answer normally and did not crash.
- All the calls were handled and saved to the database.
- No signs of overloading or crashing.

Burp suite worked fine to simulate a “high load” and the application showed good performance and **AWS Load Balancer + Auto Scaling** handled the traffic effectively. Maybe in the future i will try and write some kind of Bash/Python script that sends thousands of requests because after a little searching i came to the conclusion that Burp Suite Community Edition is good but a little limited for true overloading.

Project Title: Cloud-Based TodoApp for the fictionary company TaskForge AB
Created by: Tony Jokiranta
GitHub: https://github.com/TeeDjaay99/Todo_list
Contact: Tonyjokiranta@gmail.com
Date: 2025-04-10