

# CS554-Course Project

Rohan Vardekar  
UIC(650561030)

Project Link: [https://github.com/rvarde2/CS554\\_Project](https://github.com/rvarde2/CS554_Project)

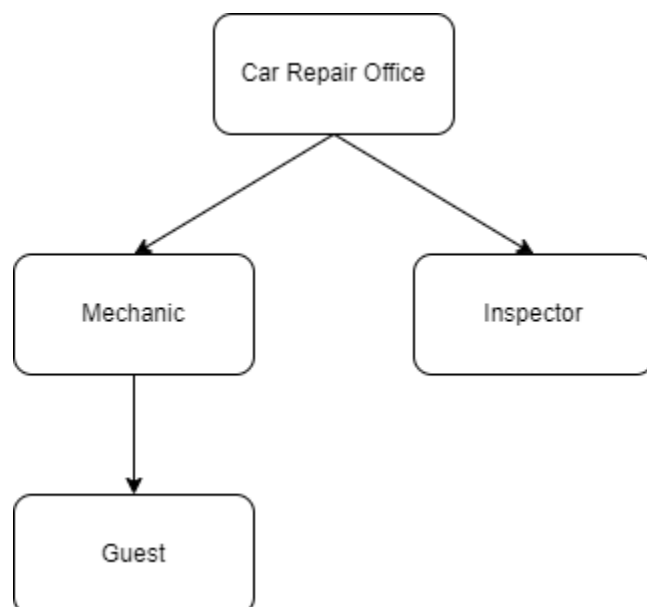
## Index:

- Introduction to Use Case
- Approach
- Components of the Project
- Call Flow Diagram
- Message Details
- Running the Project
- Telemetry Output
- Future Improvements
- Conclusion
- Additional Dependencies

## Introduction to Use Case

This project is designed to depict the scenario of the Car Repair Center. This center provides three types of repairs: Engine repair, Body repair, and Wheels repair. Along with the main Car Repair Office, there are Mechanics who will be fixing the car, and there are Inspectors who will be checking if the repair has been done properly. Now Guest will approach the Car Repair Office with a specific type of repair request for the first type along with some initial credit. The Car Repair Office will register the guest by adding him to the ledger and assign him a Mechanic which the Guest can approach directly. Usually Guest prefers his car being handled by the same Mechanic. But every time a Guest approaches the Mechanic for repair, the Mechanic should get approval from the Car Repair Office. The Car Repair Office will check if the Guest has a sufficient credit balance. If yes, then this request will be approved. The Car Repair Office will cut down the repair cost from available credits associated with that Guest from the ledger. The Car Repair Office will also suggest the Inspector to the Mechanic so that after completion of repair Mechanic can directly approach the Inspector. On completing an inspection, Inspector will give a green signal to the Mechanic so that he can hand over the Car to the Guest. Guests will go for a Test Drive and return after some amount of time with a new repair issue. For the sake of simplicity, each Guest will always come with the same type of Repair request initially mentioned. Now for the second time Guest can directly approach the Mechanic assigned to him. One Mechanic can handle multiple Guests one by one. The same goes for the Inspector. If either of them is busy, then requests will be added in Queue. Both repair work and inspection take a specific amount of time, based on the type of repair.

The hierarchy of the Car Repair System is as shown in the following diagram:



## Approach

The project is implemented in Java. Considering multiple individual entities within a system, the AKKA model is chosen for implementation. Lightbend AKKA library is used for this purpose. Car Repair Office, Mechanic, Inspector, and Guests are actors who interact with each other within a system. One of the goals of the project is to provide Telemetry Support so that the system can be monitored and tuned for optimal performance. Thus instead of starting from scratch, the Initial Template provided for the Coffee-House example provided by Lightbend is used to bypass the unnecessary complications in plugin integration. Compared to the Coffee-House example, Car-Repair implementation is easily scalable and flexible. Actor based model provides excellent modularity within a system. During development, the Car-Repair Office was implemented first, followed by Guest. After Successful Registration, Mechanics was implemented and assigned to the guest. Finally, the inspector was implemented. Guest requirements and information, repair cost, the time required for repair and inspection, no. of Mechanics and Inspectors can be modified via the Yaml file. These files will be read in the beginning. After a specified amount of duration, a snapshot of the ledger and assignments will be taken and stored in the YAML file.

## Components of the Project

**Repair:** Repair Interface is implemented by the Engine, Body, and Wheels repair class.

**CarRepairApp:** This is the entry point to the application. Actor system is created. Snapshot interval is read from the YAML file, snapshot of the ledger will be taken after every snapshot interval. Guest information is extracted from the YAML file. Create Guest request is sent to Car Repair Office.

**Car Repair:** This actor class acts as a Car Repair Office, it accepts Create Guest Message from Car Repair App and populates and registers the Guest. This class also populates Mechanics and Inspectors. It handles Approve Repair Request sent by Mechanic, checks if Guest has sufficient credit balance for the repair and replies back to the Mechanic. Repair rates are also read from the YAML file.

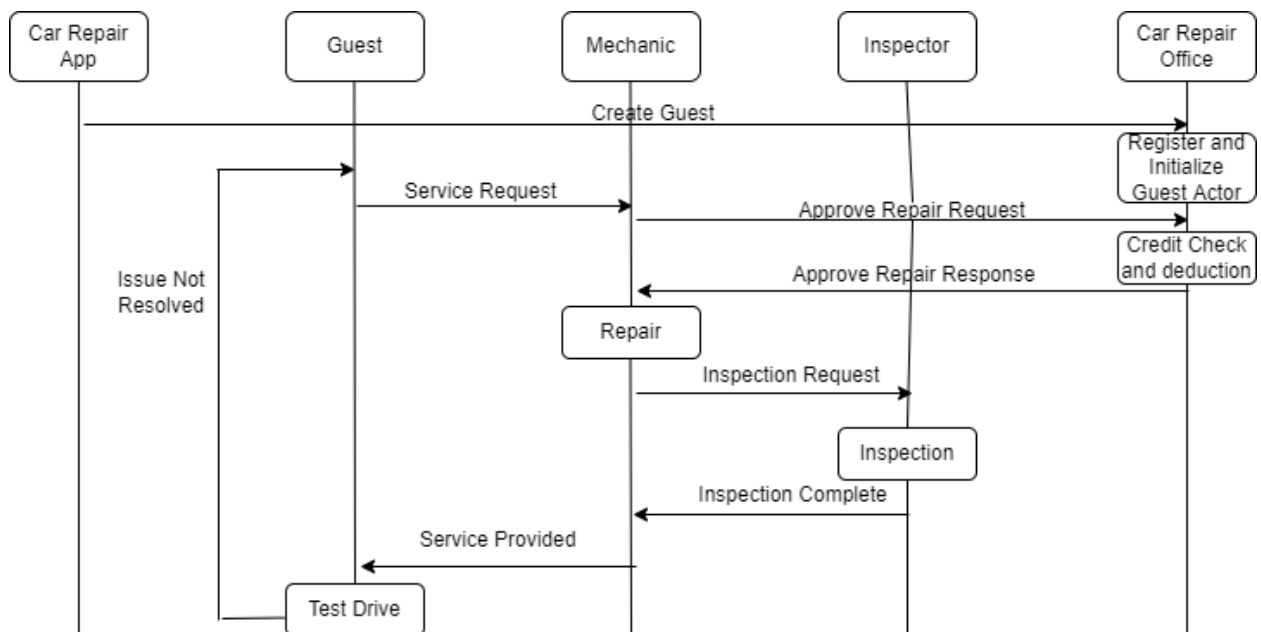
**Guest:** This actor class sends Service Request to the assigned Mechanic who was assigned during the registration phase. Guest will either receive Service Provided from Mechanic or his request will be ultimately denied by Car Repair Office due to insufficient

balance. On receiving Service Provided, the guest will go for a test drive and return after a certain amount of duration for the same repair.

**Mechanic:** Car Repair Office will populate a specific number of Mechanics based on the count provided in the YAML file. On receiving Service Request, Mechanic will send Approve Service Request to Car Repair Office. On receiving Approve Service Response, the Mechanic will repair the car for a duration dependent on the type of repair. On completing the repair, the Mechanic will send an Inspection Request to the Inspector suggested by Car Repair Office during Approve Repair Request. On receiving Inspection Complete from Inspector, Mechanic will send Service Provided to the Guest suggesting repair has been completed.

**Inspector:** Car Repair Office will populate a specific number of Inspectors based on the count provided in the YAML file. On receiving the Inspection Request, the Inspector will inspect the car for a duration dependent on the type of repair. On completing the inspection, the Inspector will reply back to the Mechanic with an Inspection Complete Message.

## Callflow Diagram



## Message Details

Message Name	From	To	Fields	Description
Create Guest	Car Repair Application	Car Repair Office	Type of Repair, Initial Credit, Test Drive Duration	Reads Guest Details from guestList.yml and asks to spawn guest actors. At this point Mechanic and Inspector are already spawned. While starting Guest, Mechanic will be assigned to Guest in roundrobin fashion.
Service Request	Guest	Mechanic	Type of Repair	Guest sending request to assigned Mechanic, Mechanic finds about Guest by looking at who has sent the message.
Approve Repair Request	Mechanic	Car Repair Office	Type of Repair, ActorRef Guest, Mechanic Id	Car Repair Office will check available credits and amount of credit required for repair before approving the request.
Approve Repair Response	Car Repair Office	Mechanic	Type of Repair, ActorRef Guest, ActorRef Inspector	Suggests that Guest has sufficient credits for repair, Credits for the repair will be deducted from the balance.
Inspection Request	Mechanic	Inspector	Type of Repair, ActoRef Guest	After completing repair, inspection is required
Inspection Complete	Inspector	Mechanic	Type of Repair, ActoRef Guest	After Inspection, we will allow Mechanic to handover car to the Guest
Service Provided	Mechanic	Guest	Type of Repair	Mechanic informing Guest that Repair is complete handover the Car

# Running the Project

**Input YAML files:** CS554\_Project\Car-Repair\application\src\main\resources

- carrepair.yml : snapshot duration, cost for various types of repair
- guestList.yml : guest details(name,initial credit, type of repair, test drive duration)
- inspector.yml : no. of inspectors, inspection duration for various types of repairs
- mechanic.yml : no. of mechanics, repair duration for various types of repairs

**Runtime Telemetry with grafana:** (Optional)

Make sure docker and docker-compose is installed. Now go to the CS554\_Project\Car-Repair\common\docker folder via command prompt or terminal, launch application by firing 'docker-compose up' command. Detail instruction on setup can be found here,

[https://academy.lightbend.com/courses/course-v1:lightbend+LAJ-P+v1/courseware/34de91591363473182383ef01cbd5aeb/fdab2e1cb8f84201b18f8562556a9b7a/2?activate\\_block\\_id=block-v1%3Alightbend%2BLAJ-P%2Bv1%2Btype%40vertical%2Bblock%4061bc52838c8a4ba5ba6517e19cd77c9b](https://academy.lightbend.com/courses/course-v1:lightbend+LAJ-P+v1/courseware/34de91591363473182383ef01cbd5aeb/fdab2e1cb8f84201b18f8562556a9b7a/2?activate_block_id=block-v1%3Alightbend%2BLAJ-P%2Bv1%2Btype%40vertical%2Bblock%4061bc52838c8a4ba5ba6517e19cd77c9b)

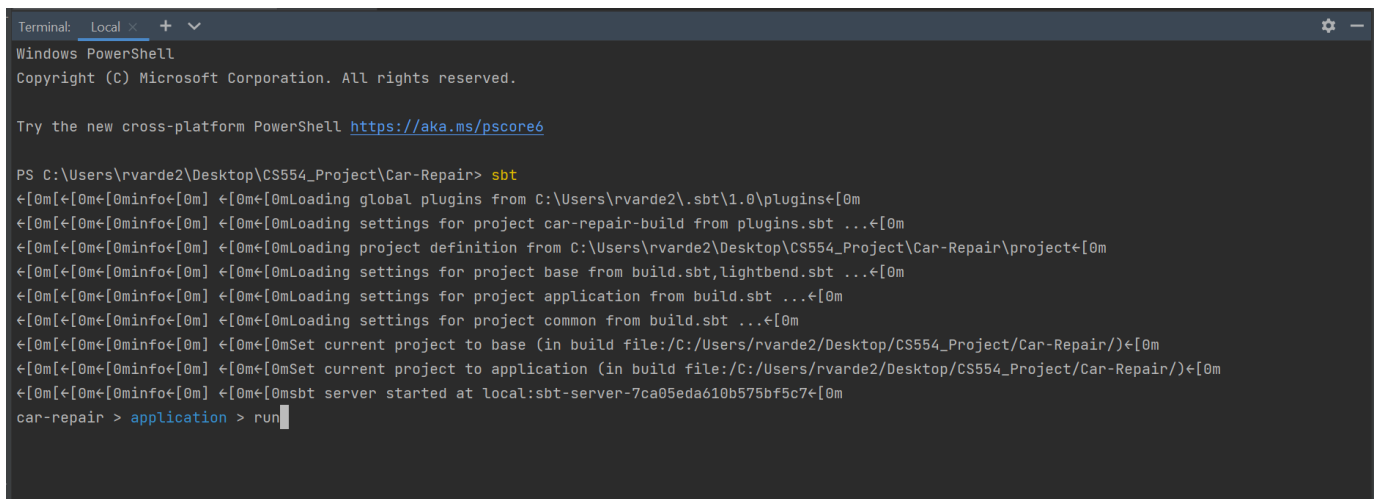
Alternatively, telemetry setup instructions can be found in repository. Once Docker Image is up and running, from the browser launch telemetry with <http://localhost:3000>

## Running the Project:

Make sure sbt is installed. Jdk 11.0.13 is used during development of the project. Using command prompt or terminal, go to the CS554\_Project\Car-Repair directory.

Launch sbt builder using command 'sbt'.

Launch application by typing 'run' command.



```
Terminal: Local + -
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\rvarde2\Desktop\CS554_Project\Car-Repair> sbt
[0m[0m[0minfo[0m] [0m[0mLoading global plugins from C:\Users\rvarde2\.sbt\1.0\plugins[0m
[0m[0m[0minfo[0m] [0m[0mLoading settings for project car-repair-build from plugins.sbt ...[0m
[0m[0m[0minfo[0m] [0m[0mLoading project definition from C:\Users\rvarde2\Desktop\CS554_Project\Car-Repair\project[0m
[0m[0m[0minfo[0m] [0m[0mLoading settings for project base from build.sbt,lightbend.sbt ...[0m
[0m[0m[0minfo[0m] [0m[0mLoading settings for project application from build.sbt ...[0m
[0m[0m[0minfo[0m] [0m[0mLoading settings for project common from build.sbt ...[0m
[0m[0m[0minfo[0m] [0m[0mSet current project to base (in build file:/C:/Users/rvarde2/Desktop/CS554_Project/Car-Repair/)[0m
[0m[0m[0minfo[0m] [0m[0mSet current project to application (in build file:/C:/Users/rvarde2/Desktop/CS554_Project/Car-Repair/)[0m
[0m[0m[0minfo[0m] [0m[0msbt server started at local:sbt-server-7ca05eda610b575bf5c7[0m
car-repair > application > run
```

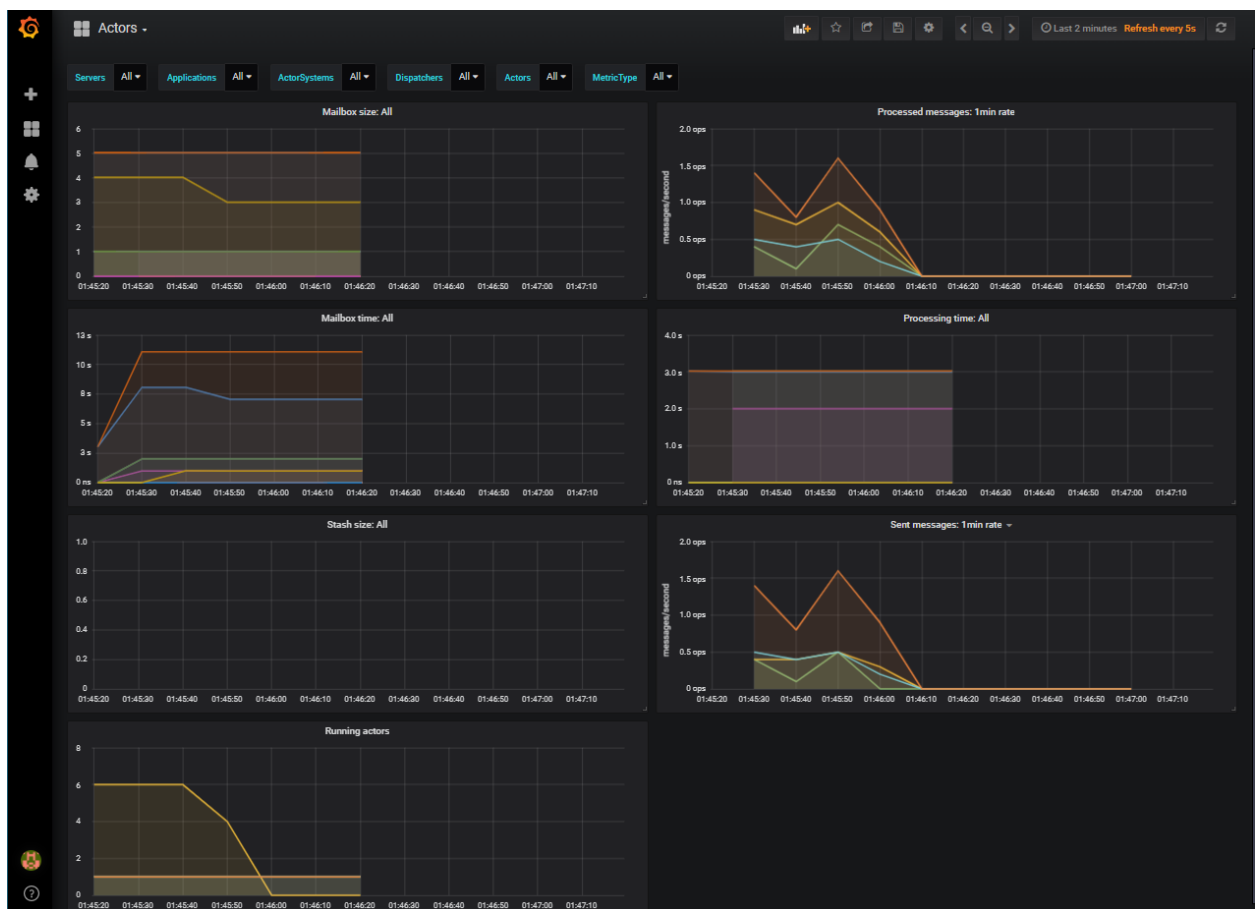
Output will be printed to the console. In addition to that logs will be collected in `CS554_Project\Car-Repair\application\car-repair.log`

Snapshots will be stored in `CS554_Project\Car-Repair\application\target\ledger.yml`

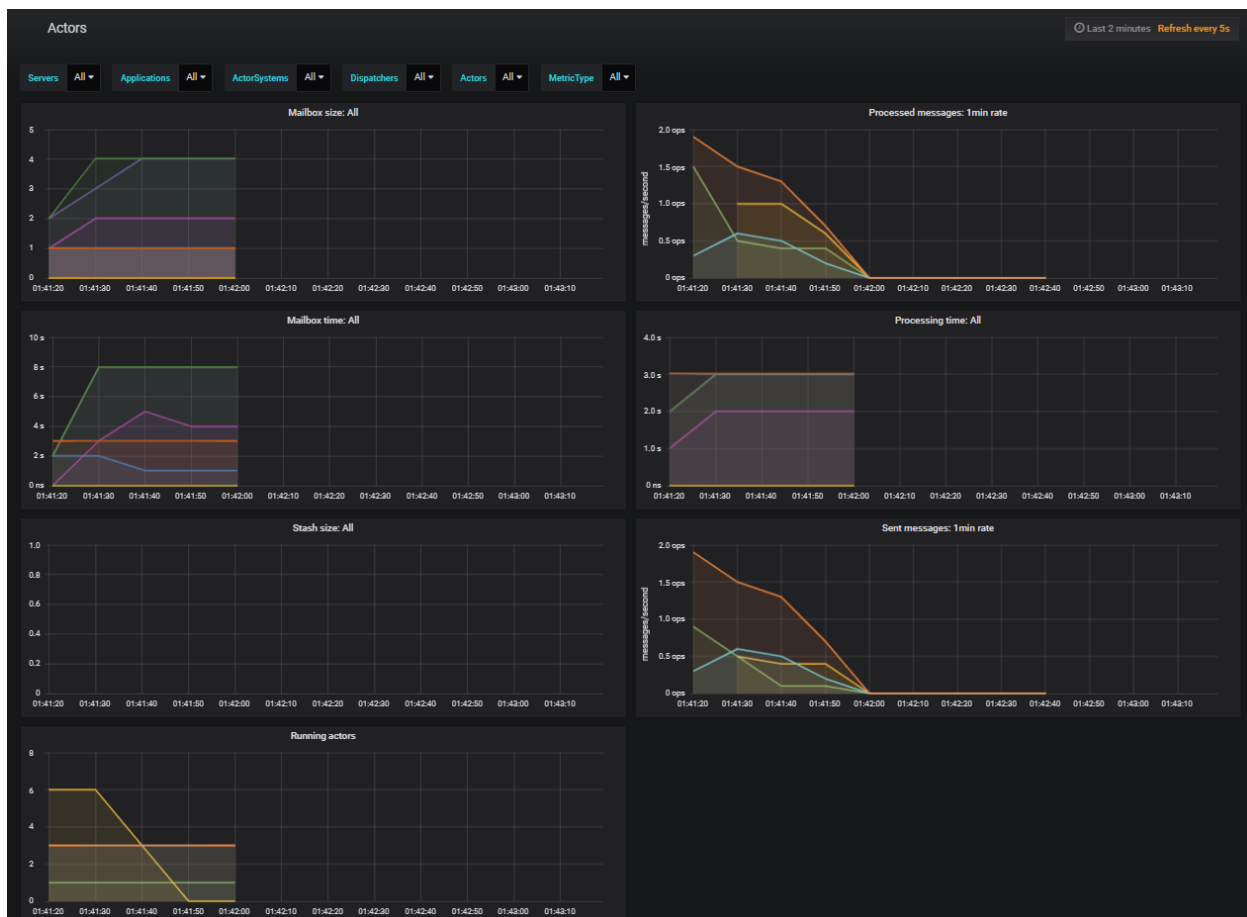
When all the requests from the Guest are denied due to insufficient balance, all the Guest actors will be terminated. To kill the application press Control+C in Powershell.

## Telemetry Output

1) 6 Guests, 1 Mechanic, 1 Inspector, Total Duration Required: 41 Seconds



2) 6 Guests, 3 Mechanic, 1 Inspector, Total Duration Required: 36 Seconds

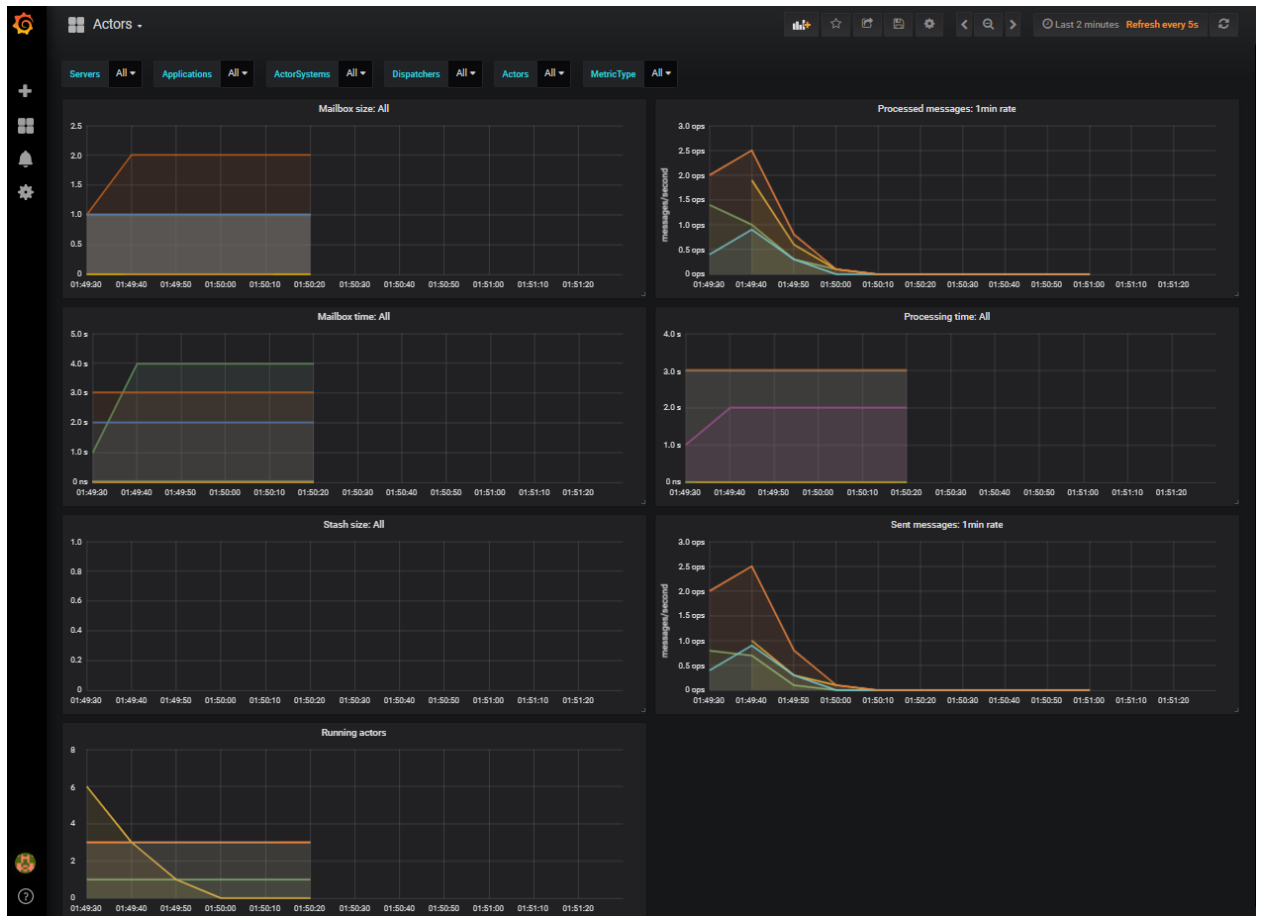




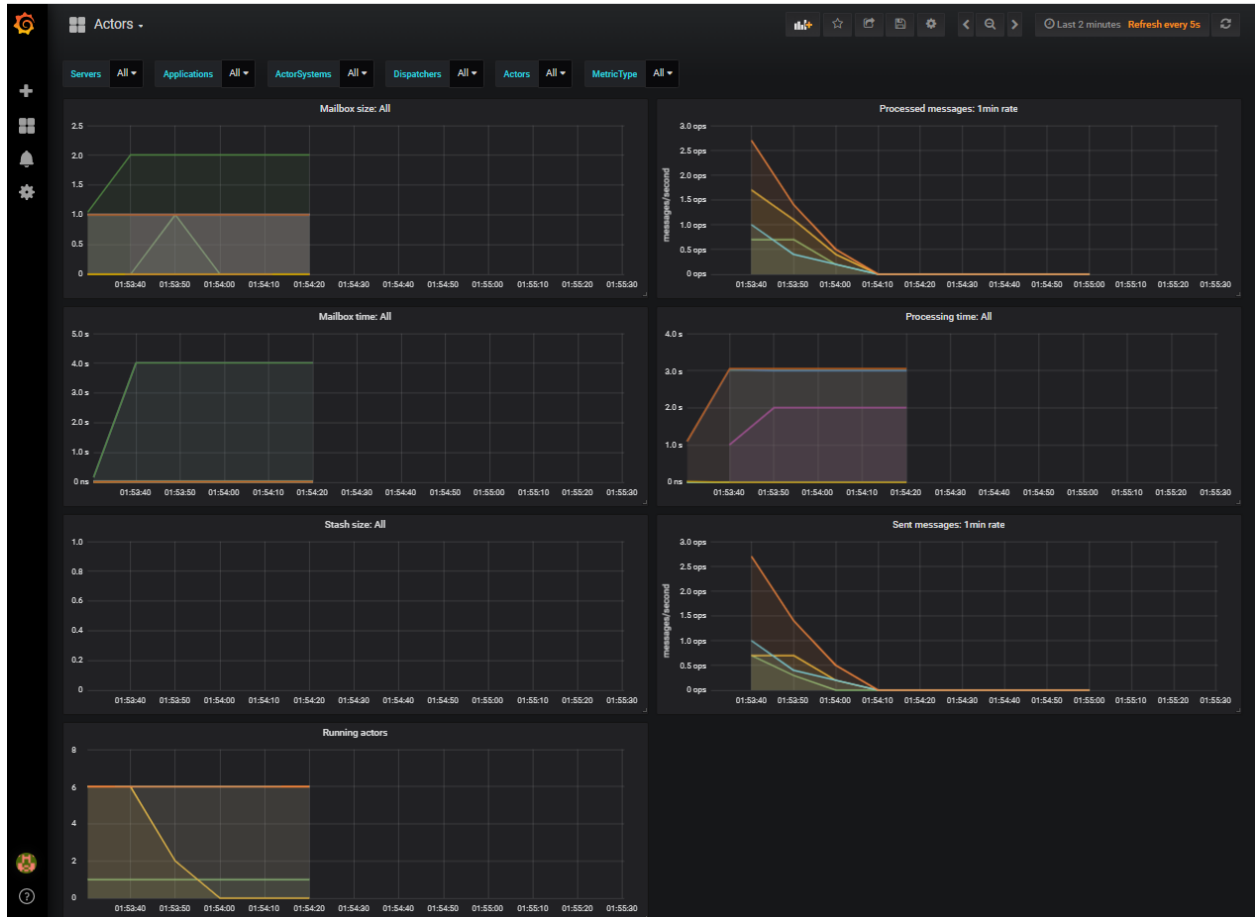
3) 6 Guests, 6 Mechanic, 1 Inspector, Total Duration Required: 37 Seconds



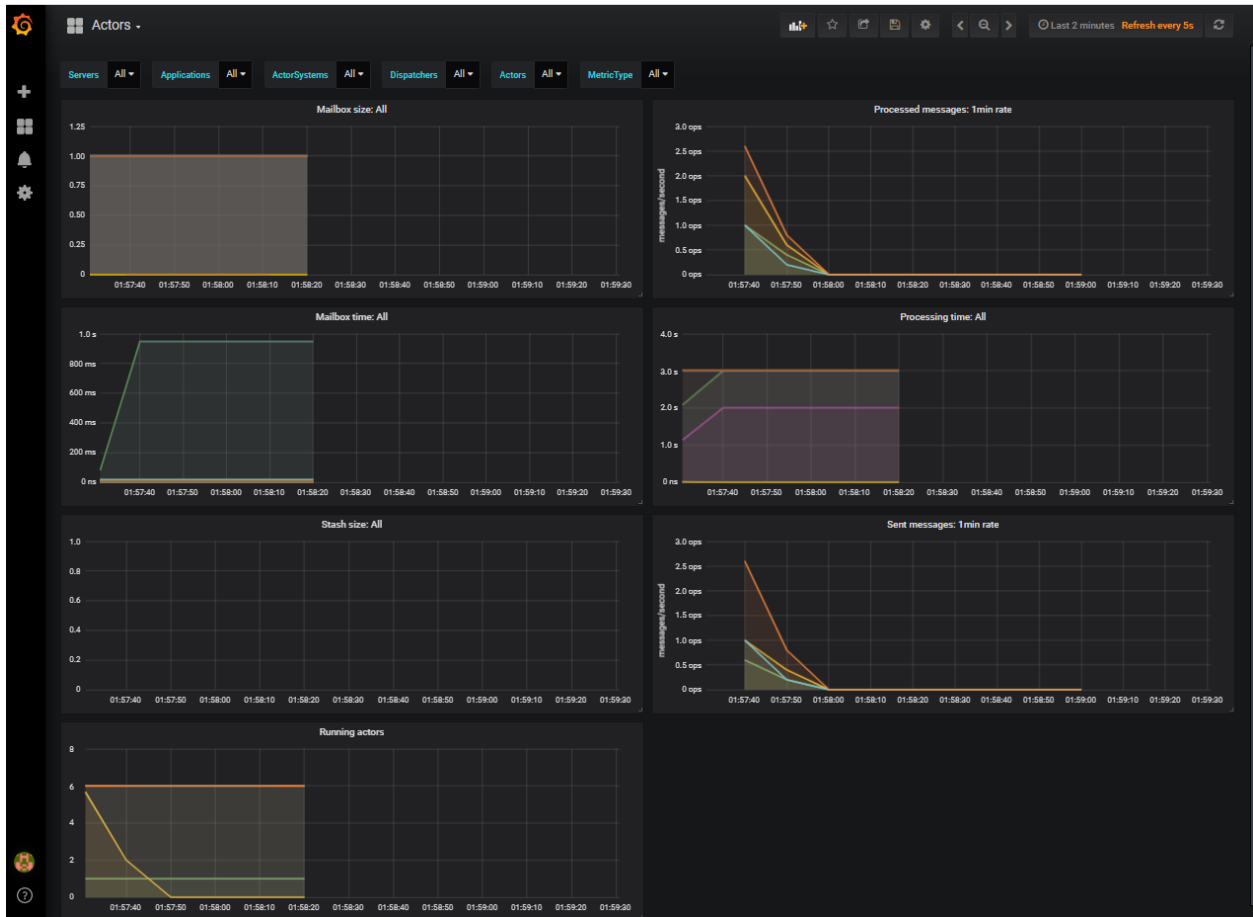
4) 6 Guests, 3 Mechanic, 3 Inspector, Total Duration Required: 27 Seconds



5) 6 Guests, 6 Mechanic, 3 Inspector, Total Duration Required: 28 Seconds



6) 6 Guests, 6 Mechanic, 6 Inspector, Total Duration Required: 24 Seconds



## **Future Improvements**

- 1) Instead of Guest directly going to Mechanic design can be changed such that Guest will directly go to the Car-Repair Office. This will eliminate delay and time wastage if Guest does not have sufficient credit and Mechanic is already busy with another Guest.
- 2) More flexibility can be introduced with Java Reflection.
- 3) Project can be deployed over the cloud.

## **Conclusion**

Actors will process and take action on only single message at a time. Minimizing either Mechanics or Inspectors essentially creates bottleneck and serializes the operation to great extent. This difference is clearly visible from Total Duration Required during above tests. With increase in available Mechanics and Inspector, throughput enhances. Note that after certain point increasing these numbers will not be beneficial if all of them are waiting and working at the same time. Ideal scenario is without for each other all of the actors complete their task.

## **Additional Dependencies**

- 1) Java jdk 11.0.13
- 2) Sbt
- 3) Lighbend AKKA library
- 4) Snake YAML for YAML Parsing
- 5) Docker and Docker Compose for Telemetry