ServiceDesk

This repository holds the demo implementation for service desk problem.

Abstract

A service desk platform where

- A company can register itself
- Company support agents (SA) for the company can
- - Login at the start of the day
    - - Logout at the end of the day
  - Users (Customers) should be able to raise tickets.
  - Tickets can either be accepted or rejected based on the availability of the agents to serve them.
  - Tickets will be assigned to the agents based on availability of the agent

Solution

APPROACH

- Single Login System based on RBAC(Role Based Access Control) should be implemented for better scalability, as login system in itself is a big thing.
  - The first scalable approach should be that the ticket accepting or rejecting should not depend on the number of agents as it can be a very large number.
- - For this, i will create an array of size 480(1-indexed)(slots)(as largest number is 480 minutes for bronze) and initialize the array with 0 on all places. then, start filling bronze tickets from min(timeRemaining,480) to 1, silver from min(timeRemaining,240) to 1 and gold from min(timeRemaining,120) to 1. I will check if the arr[i]< numberofagents, that means the ticket can be accepted for that slot. Also after each minute, the slot i will be updated with i+1 slot(because a minute has passed and that ith slot is no longer valid for ith minute now). And i will start solving the tickets from index 1, this will ensure all the tickets are solved in their given time.
  - - With this, the acceptance or rejectance of a ticket can be deduced in O(1) time complexity and not depend on the number of agents for a company.
- - Creating separate goroutines for each agent for the company, which will start solving the ticket from the starting position of the array and sleep for a minute then close the ticket.
- - mutex has been acquired at the proper places where it is required, as there will be a lot of goroutines creating all sorts of race conditions.

Challenges

- If the system scales to the point where buying a larger system is no longer feasible, and now has to be scaled horizontally-
- - This can happen as there is a map and for each company there is a struct which has an array of size 480, which counts for around 3.5KBs of memory, so for 10 lakh companies it will be 3.5 GBs of memory.

- There are a lot of agents for some company and so are the number of goroutines.
- As the primary driver of the system is the array which cannot be distributed, therefore a distributed caching system can be used for example redis. So, if a server goes down, then also we can extract the array and process the tickets and accept new tickets. This will increase network calls to redis.
  - The gateway can be implemented for a lot of reasons, security being the one and it can also store a map of companyid to servernumber on which it is hosted on, and if the ticket accpetance or rejection has to be handled synchronously then either http call or rpc call can be made to that server, and if the ticket acceptance or rejectance can be handled asynchronously, then the ticket request can be put in a kafka partition of a kafka topic according to the server number, and then a group of servers can belong to a specific consumer group in to consume from that kafka topic to ensure each ticket is processed once, but for this distributed caching is must.
- Also if a server dies, then an orchestrator must bring it up, and there should be a getServiceUp() Method which can be implemented easily if we are using distributed caching as it already has the latest state of the service which is essentially the state of the array.

Technology Used

- Everything is made in go (building,running with makefile)(default goal is help).

```
make
*    dev             to build the binary and run the code
*    build           to build the binary
```

- A lot of things are not completed,

> *like dependency injection of a NoSQL DB. implementing the redis save.*
> *gateway not implemented completely.*