GoLang Assignment

Task 1: (Score - 50)

Write a program that uses Go's concurrency features to print the numbers from 1 to 100 concurrently. The program should use a goroutine for each number, and the main goroutine should wait for all the goroutines to finish before exiting.

Task 2: (Score - 100)

Write a Go program that connects to a remote server via SSH, runs a specified ("uptime") command, and prints the command's output to the console. Parse the uptime command output to get details like server, status and users count. Final output to have JSON entries for all records in uptime.json file. The program should take the server's IP address, username, and password as command-line arguments. Additionally, the program should handle errors gracefully and be able to handle large outputs efficiently.

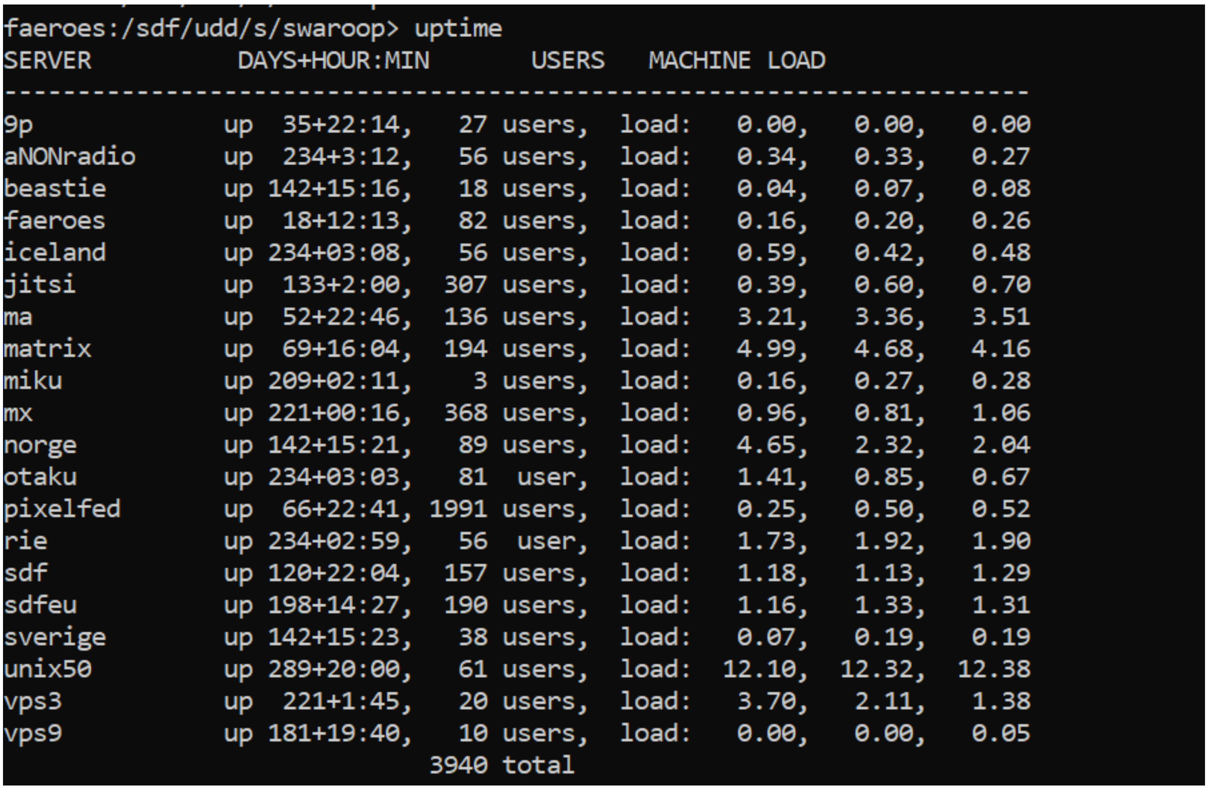
Instructions:

•The candidate would have to use Go's net and crypto libraries to establish the SSH connection and authenticate the user and write a function that takes the command-line arguments and use them to connect and authenticate to the remote server and run the command. Additionally, the candidate should handle errors and large outputs efficiently.

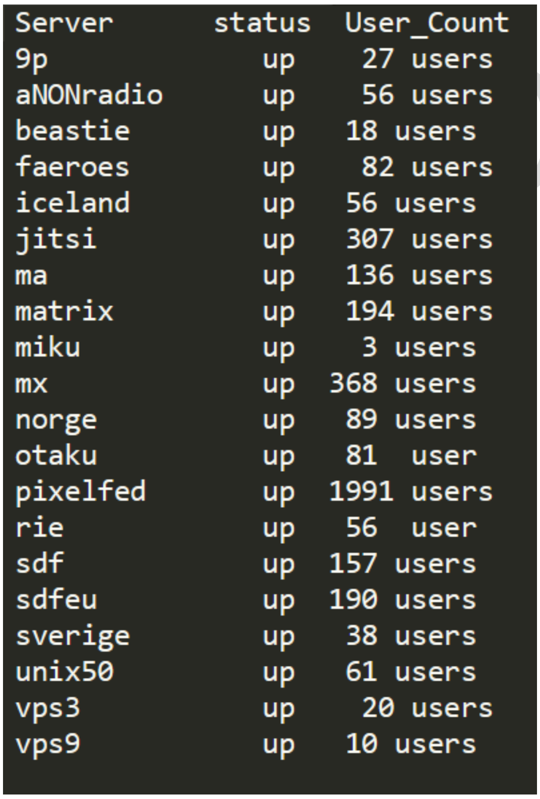
•Use following details for ssh hostname/IP = tty.sdf.org , username=Swaroop, password=cvTOaF06RjdArA, command to run=uptime

•Send uptime.json along with code files.

```
>ssh swaroop@tty.sdf.org
```

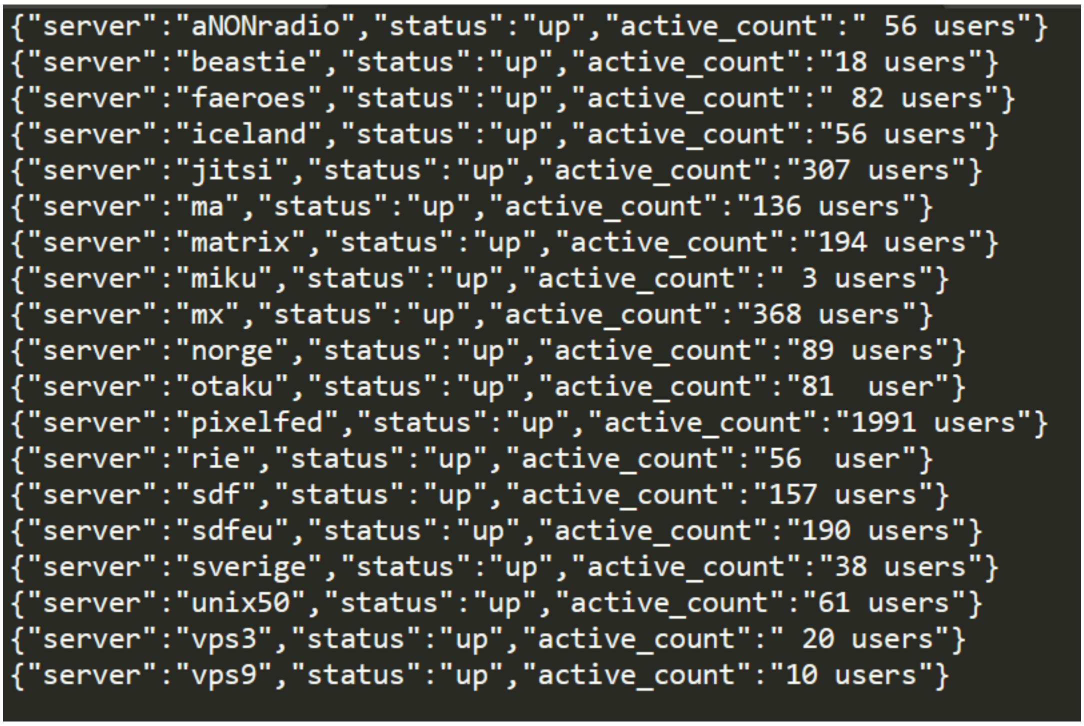


```
faeroes:/sdf/udd/s/swaroop> uptime
SERVER          DAYS+HOUR:MIN       USERS   MACHINE LOAD
----------------------------------------------------------------------
9p             up  35+22:14,   27 users,  load:  0.00,   0.00,   0.00
aNONradio      up  234+3:12,   56 users,  load:  0.34,   0.33,   0.27
beastie        up 142+15:16,   18 users,  load:  0.04,   0.07,   0.08
faeroes        up  18+12:13,   82 users,  load:  0.16,   0.20,   0.26
iceland        up 234+03:08,   56 users,  load:  0.59,   0.42,   0.48
jitsi          up  133+2:00,  307 users,  load:  0.39,   0.60,   0.70
ma             up  52+22:46,  136 users,  load:  3.21,   3.36,   3.51
matrix         up  69+16:04,  194 users,  load:  4.99,   4.68,   4.16
miku           up 209+02:11,    3 users,  load:  0.16,   0.27,   0.28
mx             up 221+00:16,  368 users,  load:  0.96,   0.81,   1.06
norge          up 142+15:21,   89 users,  load:  4.65,   2.32,   2.04
otaku          up 234+03:03,   81  user,  load:  1.41,   0.85,   0.67
pixelfed       up  66+22:41, 1991 users,  load:  0.25,   0.50,   0.52
rie            up 234+02:59,   56  user,  load:  1.73,   1.92,   1.90
sdf            up 120+22:04,  157 users,  load:  1.18,   1.13,   1.29
sdfeu          up 198+14:27,  190 users,  load:  1.16,   1.33,   1.31
sverige        up 142+15:23,   38 users,  load:  0.07,   0.19,   0.19
unix50         up 289+20:00,   61 users,  load: 12.10,  12.32,  12.38
vps3           up  221+1:45,   20 users,  load:  3.70,   2.11,   1.38
vps9           up 181+19:40,   10 users,  load:  0.00,   0.00,   0.05
                             3940 total
```



```
Server       status  User_Count
9p             up     27 users
aNONradio      up     56 users
beastie        up    18 users
faeroes        up     82 users
iceland        up    56 users
jitsi          up    307 users
ma             up    136 users
matrix         up    194 users
miku           up      3 users
mx             up   368 users
norge          up    89 users
otaku          up    81   user
pixelfed       up   1991 users
rie            up    56   user
sdf            up   157 users
sdfeu          up   190 users
sverige        up    38 users
unix50         up    61 users
vps3           up     20 users
vps9           up    10 users
```

Final output as below



```
{"server":"aNONradio","status":"up","active_count":" 56 users"}
{"server":"beastie","status":"up","active_count":"18 users"}
{"server":"faeroes","status":"up","active_count":" 82 users"}
{"server":"iceland","status":"up","active_count":"56 users"}
{"server":"jitsi","status":"up","active_count":"307 users"}
{"server":"ma","status":"up","active_count":"136 users"}
{"server":"matrix","status":"up","active_count":"194 users"}
{"server":"miku","status":"up","active_count":" 3 users"}
{"server":"mx","status":"up","active_count":"368 users"}
{"server":"norge","status":"up","active_count":"89 users"}
{"server":"otaku","status":"up","active_count":"81  user"}
{"server":"pixelfed","status":"up","active_count":"1991 users"}
{"server":"rie","status":"up","active_count":"56  user"}
{"server":"sdf","status":"up","active_count":"157 users"}
{"server":"sdfeu","status":"up","active_count":"190 users"}
{"server":"sverige","status":"up","active_count":"38 users"}
{"server":"unix50","status":"up","active_count":"61 users"}
{"server":"vps3","status":"up","active_count":" 20 users"}
{"server":"vps9","status":"up","active_count":"10 users"}
```

Task 3: (Score - 100)

Create a Go program that automates the execution of commands on multiple remote servers via SSH in parallel. The program should be able to read a list of servers and commands from a configuration file and execute the commands on all the servers simultaneously. The program should be able to handle authentication using password or key-based authentication. The program should be able to log the output of the commands executed on each server to a local file.

Instructions:

•Use following details for ssh hostname/IP = tty.sdf.org , username=swaroop, password=cvTOaF06RjdArA

•You should use Go's built-in golang.org/x/crypto/ssh package for connecting to the remote servers and executing the commands via SSH.

•You should not use any external libraries or frameworks.

•You should provide clear and concise comments in the code to explain your implementation.

•Once you have finished, please share your code and any instructions for running the program.

•Provide a short explanation of your solution, how you handle parallelism, how you handle authentication and any other notable points.

•Bonus points for providing an example of a configuration file that the program can use to execute commands on multiple servers.

Task 4: (Score - 100)

Create a RESTful API in Go that allows users to manage a list of items. The API should have the following endpoints:

GET /items - Retrieves a list of all items.

GET /items/{id} - Retrieves a specific item by its ID.

POST /items - Creates a new item.

PUT /items/{id} - Updates an existing item.

DELETE /items/{id} - Deletes an existing item.

The item object should have the following fields:

ID (unique identifier)

Name (string)

Description (string)

Price (float)

Instructions:

•The API should also include proper error handling and validation for each endpoint.

•You should use the "net/http" package and a router library like Gorilla Mux to handle routing.

•You can use an in-memory data storage such as slice to keep the data and use uuid package to generate unique id.

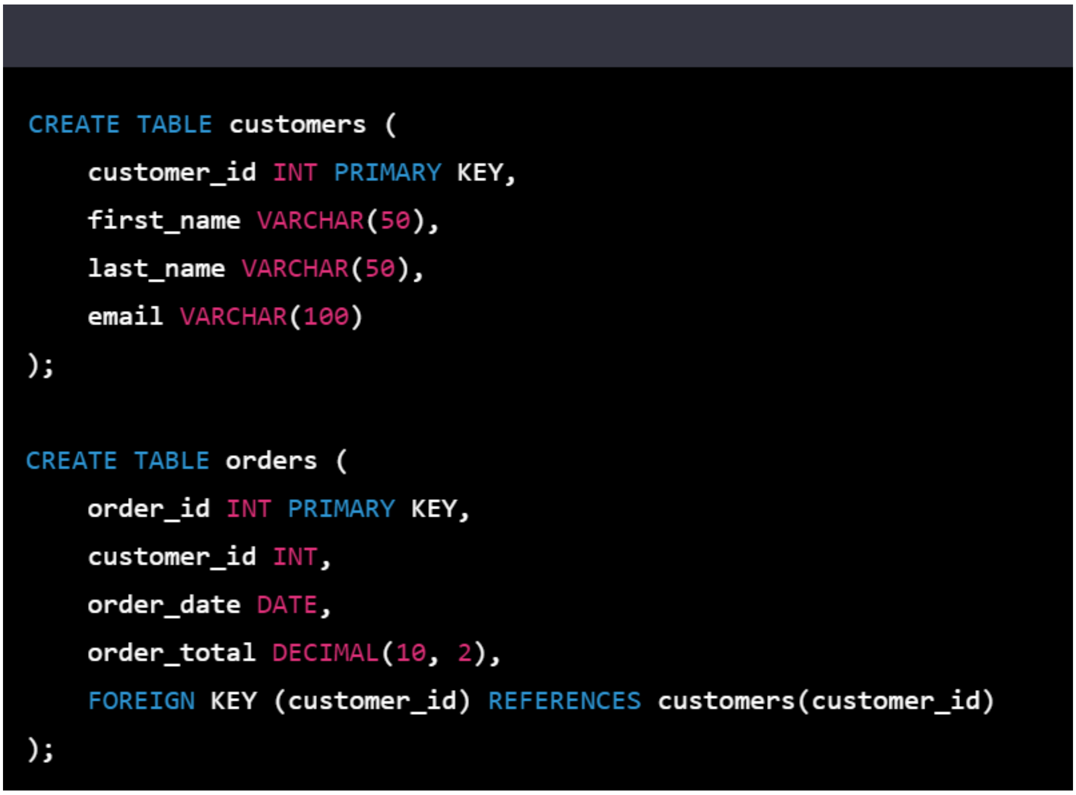
Task 5: (Score - 50)

Consider below SQL dataset

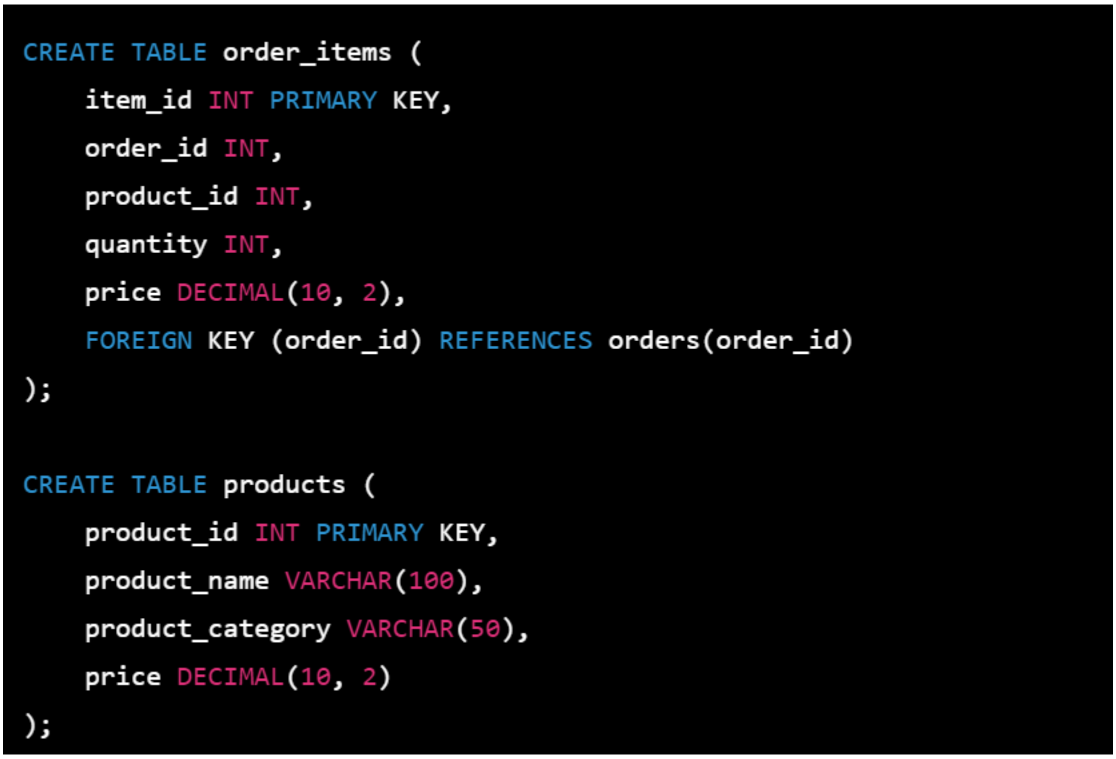
```
CREATE TABLE customers (
   customer_id INT PRIMARY KEY,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   email VARCHAR(100)
);

CREATE TABLE orders (
   order_id INT PRIMARY KEY,
   customer_id INT,
   order_date DATE,
   order_total DECIMAL(10, 2),
   FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

CREATE TABLE order_items (
   item_id INT PRIMARY KEY,
   order_id INT,
   product_id INT,
   quantity INT,
   price DECIMAL(10, 2),
   FOREIGN KEY (order_id) REFERENCES orders(order_id)
);

CREATE TABLE products (
   product_id INT PRIMARY KEY,
   product_name VARCHAR(100),
   product_category VARCHAR(50),
    price DECIMAL(10, 2)
);
```

Write a query that finds the total number of orders and total order value for each customer, grouped by customer, and ordered by total order value in descending order.

The query should also return the first name, last name, and email of the customer.



```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100)
);

CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    order_total DECIMAL(10, 2),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```



```
CREATE TABLE order_items (
    item_id INT PRIMARY KEY,
    order_id INT,
    product_id INT,
    quantity INT,
    price DECIMAL(10, 2),
    FOREIGN KEY (order_id) REFERENCES orders(order_id)
);

CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    product_category VARCHAR(50),
    price DECIMAL(10, 2)
);
```

Note: Share code and output screen for each step in a zipped folder