0. Prerequisites

0.1. Docker account
0.2. OpenShift account

1. Fixes

Story(rephrase): RollerCoaster operates in periodical timeline(standalone server run). And at the start point(by including client run), random number of users get in the roller coaster. Cycle will operate continuously unless there is no interruption on client or server.

Visible problems:

1.1. **Thread Safety**-> Slices.

The `rideQueue` slice is accessed concurrently from multiple goroutines. The `rideQueue` modifications are not always protected by a mutex, which could lead to race conditions.

- **Issue**: In `start()`, while looping over `rc.rideQueue`, you lock `rc.rideQueueMu` only for each rider being seated, not for the entire slice operation.
- **Fix**: The lock should cover both the slice length check and the operation that removes riders from the queue.

1.2. **Race Condition** -> `rc.ride`

The `rc.ride` slice (used to store seated riders) is being modified in the `seatRider()` method without proper locking around the append operation.

- **Fix**: Lock the `rideMu` before modifying `rc.ride`, and unlock once operation is complete. (🔒)

1.3. **Obsolete Imports:** You are importing `"math"` and `"math/rand"` but don't need `"math"`. You can use `rand.Intn()` directly, and there's no need for `math.Abs()` in this context.

1.4. **Randomisation:** In the client, you're using `rand.IntN()` without seeding it with a unique seed value. This can lead to the same sequence of random numbers each time the client runs.

- **Fix**: Use `rand.Seed(time.Now().UnixNano())` to ensure different random sequences for each client run.

1.5. **Handler:** http.Post

The client is sending POST requests without checking the response or handling errors, which could lead to issues like unhandled request failures or connection problems.

- **Fix**: Add error handling for the `http.Post` calls to ensure proper handling of failures.

1.6. **Indexing:** Car and Seat calculation:
You are using this calculation to assign a car and seat:

```
car := int(math.Abs(float64((i)/2))) + 1
carSeat := i % carCapacity
```

This calculation will sometimes assign riders incorrectly. For example, with `i = 0`, it assigns the rider to car 1, seat 0, which is fine, but the way it's structured seems overcomplicated.

- **Fix**: Simplify the seat assignment logic by dividing `i` by `carCapacity` to get the car number and using `i % carCapacity` for the seat number.

1.7. **Context Cancellation**:

The context in the `start()` method is not checked after the ride is done (`time.Sleep(10000 * time.Millisecond)`). This may cause the goroutine to block when the server is shutting down.

- **Fix**: Check `ctx.Done()` after the `time.Sleep` to allow early cancellation of the ride loop. 🪂__

Story complete:
- Queued riders are taking their rides, and extracted from Queue once their ride is finished.
- New riders are queued continuously, and appended to rideQueue without any interruption.
- Optimisation(possible): Create another queue for waiting riders to do sth else, though likely not necessary.

This should be much safer and less prone to concurrency issues.

2. Creating Deployment

2.1. Containerise application: Docker-files (🐳)

To avoid tedious networking in Docker I use OpenShift deployment. For now, I'll publish only the endpoint to public, since I can not share my OpenShift administrator credentials with you.

Building images:

```
docker build -t coaster-server .
```

```
docker build -t coaster-client .
```

Functional test images locally.

```
docker run -p 3000:3000 coaster-server
docker run coaster-client
```

2.2 Tagging images and pushing to docker-hub.

```
docker tag coaster-server tanermetin/coaster-server:latest
docker tag coaster-client tanermetin/coaster-client:latest
docker push tanermetin/coaster-server:latest
docker push tanermetin/coaster-client:latest
```

2.3 Making necessary configurations in OpenShift (🧉)

Pushed docker images is pulled by deployment files to create our service. Creating necessary configuration files for server, client, service and route, that helps to create pods, run service and expose public endpoint for us.

```
oc apply -f server/server-deployment.yaml
oc apply -f client/client-deployment.yaml
oc apply -f server/
oc apply -f route.yaml
```

2.3 Optional (get, destroy)

To delete only pods.

```
oc delete pods --all
oc delete all --all
```

To get basic backbone knowledge.

```
oc get pods
oc get deployments
oc get services
oc get routes
```

To delete all resources -CAREFUL WITH THIS ONE-.

```
oc delete all --all
```

3. Scaling Deployment

3.1. Multiple existing ways to scale application.

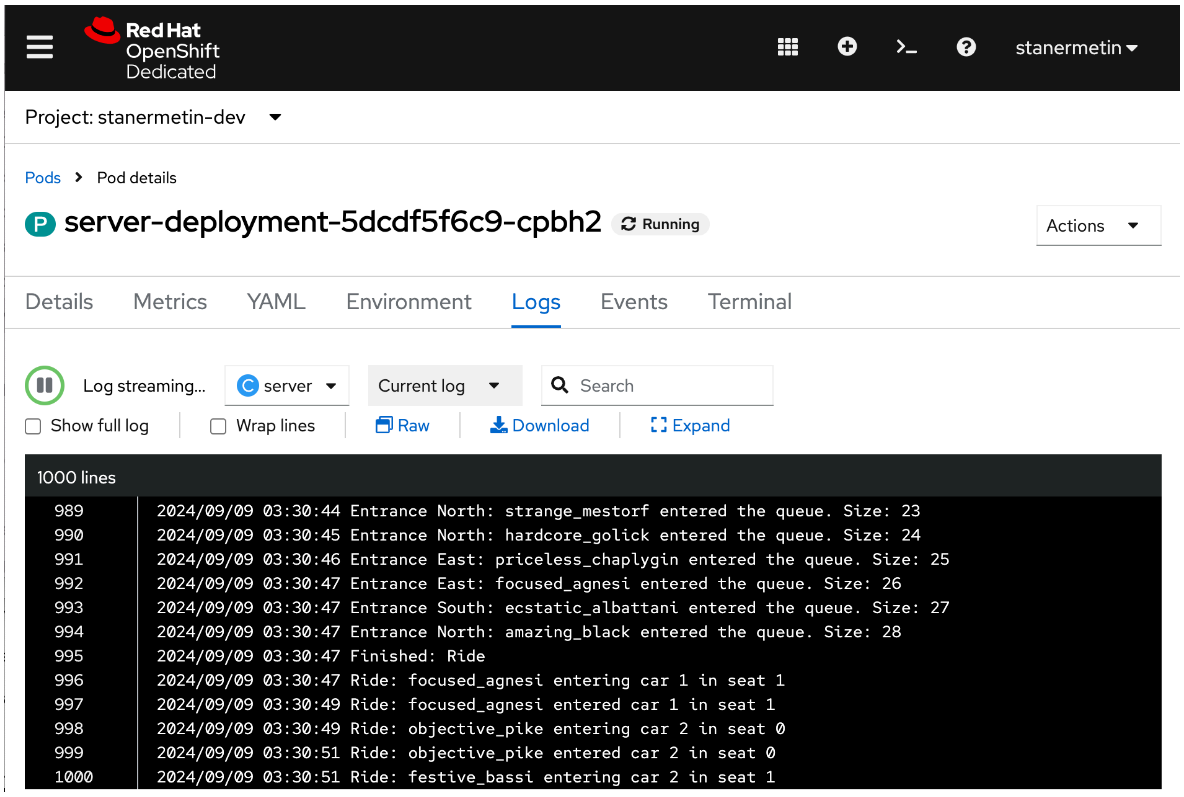
Option 3.1.1: Changing *replica* value in deployment files.

Option 3.1.2: Using the scalability options on OpenShift/GKE/AKS. Additionally add *LB*.

Server-Route:

```
https://server-route-stanermetin-dev.apps.sandbox-m2.ll9k.p1.openshiftapps.com
```

```
Client-Route:
https://client-route-stanermetin-dev.apps.sandbox-m2.ll9k.p1.openshiftapps.com
```



```
2024/09/09 03:30:44 Entrance North: strange_mestorf entered the queue. Size: 23
2024/09/09 03:30:45 Entrance North: hardcore_golick entered the queue. Size: 24
2024/09/09 03:30:46 Entrance East: priceless_chaplygin entered the queue. Size: 25
2024/09/09 03:30:47 Entrance East: focused_agnesi entered the queue. Size: 26
2024/09/09 03:30:47 Entrance South: ecstatic_albattani entered the queue. Size: 27
2024/09/09 03:30:47 Entrance North: amazing_black entered the queue. Size: 28
2024/09/09 03:30:47 Finished: Ride
2024/09/09 03:30:47 Ride: focused_agnesi entering car 1 in seat 1
2024/09/09 03:30:49 Ride: focused_agnesi entered car 1 in seat 1
2024/09/09 03:30:49 Ride: objective_pike entering car 2 in seat 0
2024/09/09 03:30:51 Ride: objective_pike entered car 2 in seat 0
2024/09/09 03:30:51 Ride: festive_bassi entering car 2 in seat 1
```

task duration: 4 hrs

Taner Metin