Table of Contents

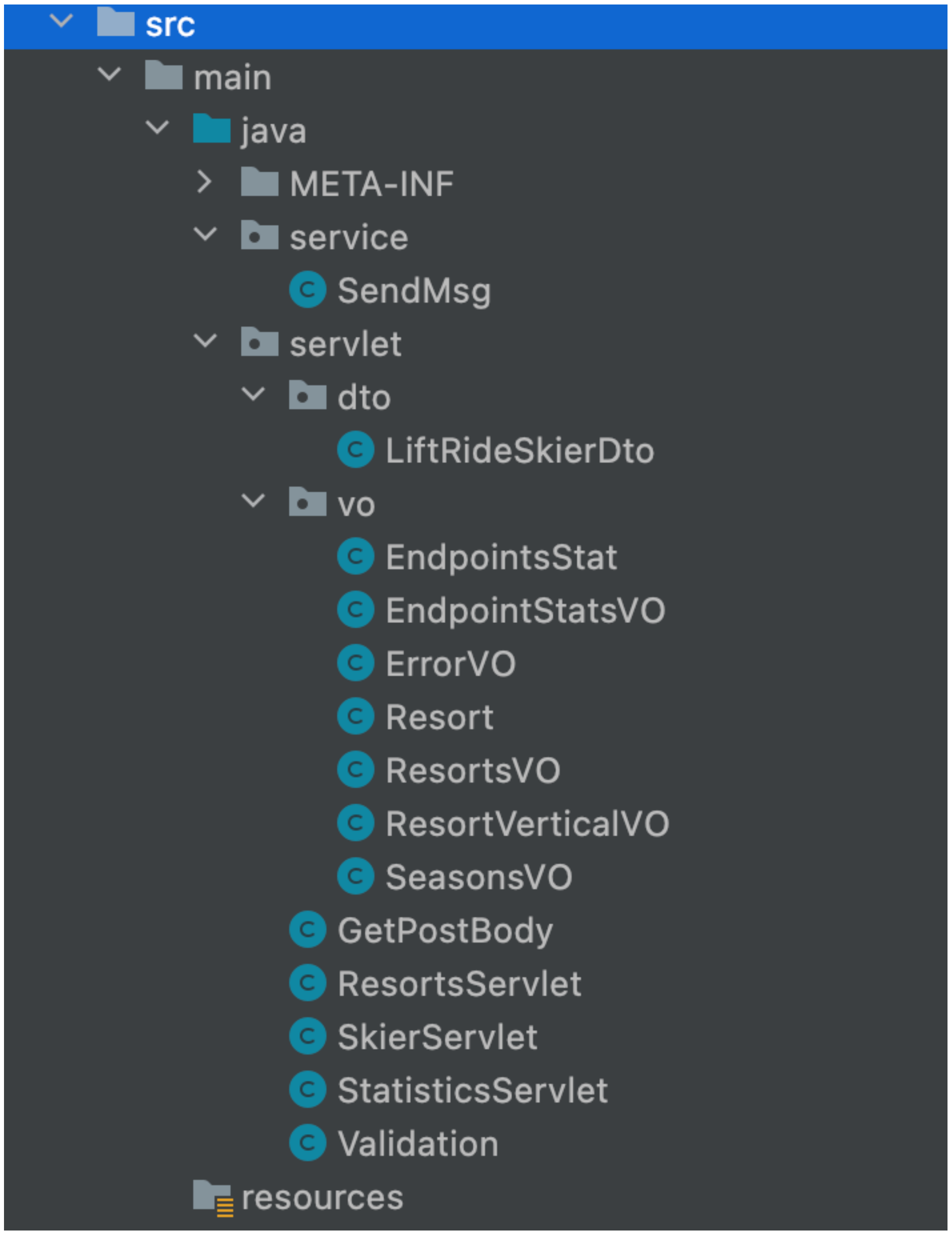
Git repo

https://github.com/Peihao-Zhu/cs6650

Description of design

Server / Producer

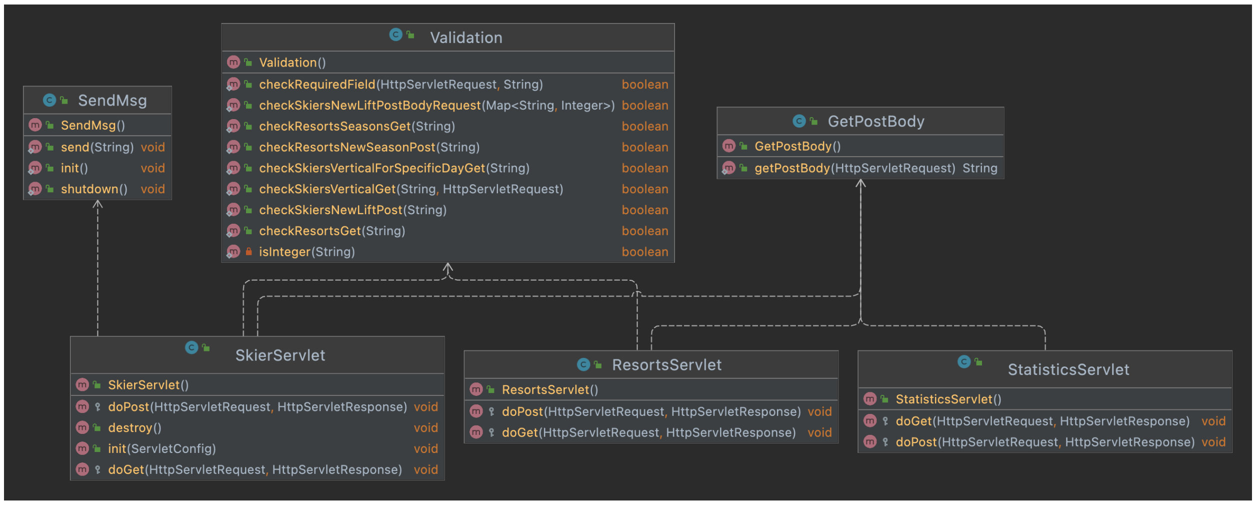
Below is my “Server” project directory structure



```
src
  main
    java
      META-INF
      service
        SendMsg
      servlet
        dto
          LiftRideSkierDto
        vo
          EndpointsStat
          EndpointStatsVO
          ErrorVO
          Resort
          ResortsVO
          ResortVerticalVO
          SeasonsVO
        GetPostBody
        ResortsServlet
        SkierServlet
        StatisticsServlet
        Validation
    resources
```

We can see there are two folders named **service** and **servlet** under folder **java**. The folder service is used for business service. And folder servlet acts like controller in MVC framework.

Below is the URL diagram for my server project.


SendMsg
SendMsg()
send(String) void
init() void
shutdown() void
Validation
Validation()
checkRequiredField(HttpServletRequest, String) boolean
checkSkiersNewLiftPostBodyRequest(Map<String, Integer>) boolean
checkResortsSeasonsGet(String) boolean
checkResortsNewSeasonPost(String) boolean
checkSkiersVerticalForSpecificDayGet(String) boolean
checkSkiersVerticalGet(String, HttpServletRequest) boolean
checkSkiersNewLiftPost(String) boolean
checkResortsGet(String) boolean
isInteger(String) boolean
GetPostBody
GetPostBody()
getPostBody(HttpServletRequest) String
SkierServlet
SkierServlet()
doPost(HttpServletRequest, HttpServletResponse) void
destroy() void
init(ServletConfig) void
doGet(HttpServletRequest, HttpServletResponse) void
ResortsServlet
ResortsServlet()
doPost(HttpServletRequest, HttpServletResponse) void
doGet(HttpServletRequest, HttpServletResponse) void
StatisticsServlet
StatisticsServlet()
doGet(HttpServletRequest, HttpServletResponse) void
doPost(HttpServletRequest, HttpServletResponse) void

SendMsg class:
SendMsg class is the rabbitmq producer. We use init() to initialize the connection factory and create specific connection as well as the channels; shutdown() method will be called when we shutdown the tomcat server, and all the channels and connection will be closed smoothly. send() mehod is the core part to implement the function of sending the message to rabbitmq server.

SkierServlet class:
SkierServlet class is a servlet we parse the requests related to skier API.

ResortServlet class:
ResortServlet class is a servlet we parse the requests related to resort API.

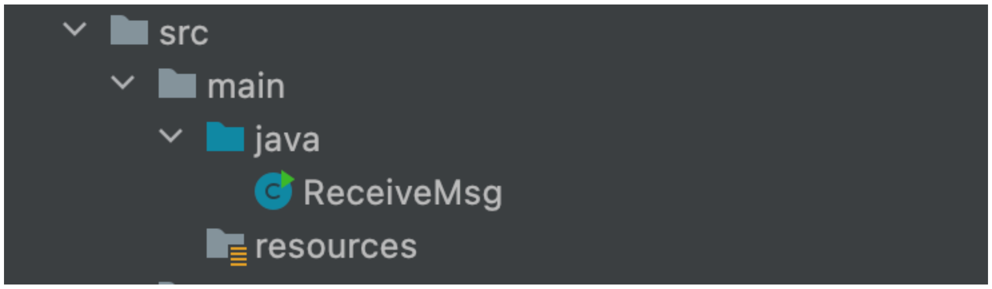
StatisticsServlet class:
StatisticsServlet class is a servlet we parse the requests related to statistics API.

Validation class:
Validation class is mainly used to validate the url and its corresponding parameters.

GetPostBody class:
GetPostBody class is a common class that we can get the post body with JSON type from a request, and return as a string type.

Consumer

Below is my “consumer” project directory structure



It’s really simple. The ReceiveMsg class has the function of subscribing a specific connection and consume the messages coming from the connection. Because we want to speed up the consuming process, so we let the class implement Runnable interface, and create multiple threads to concurrently consume messages from rabbitmq server.

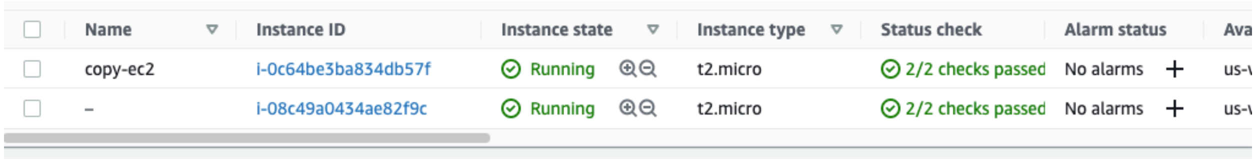
Test results

Load balancer

Before I go down the detailed result for different test cases, I will describe what I have done for server.

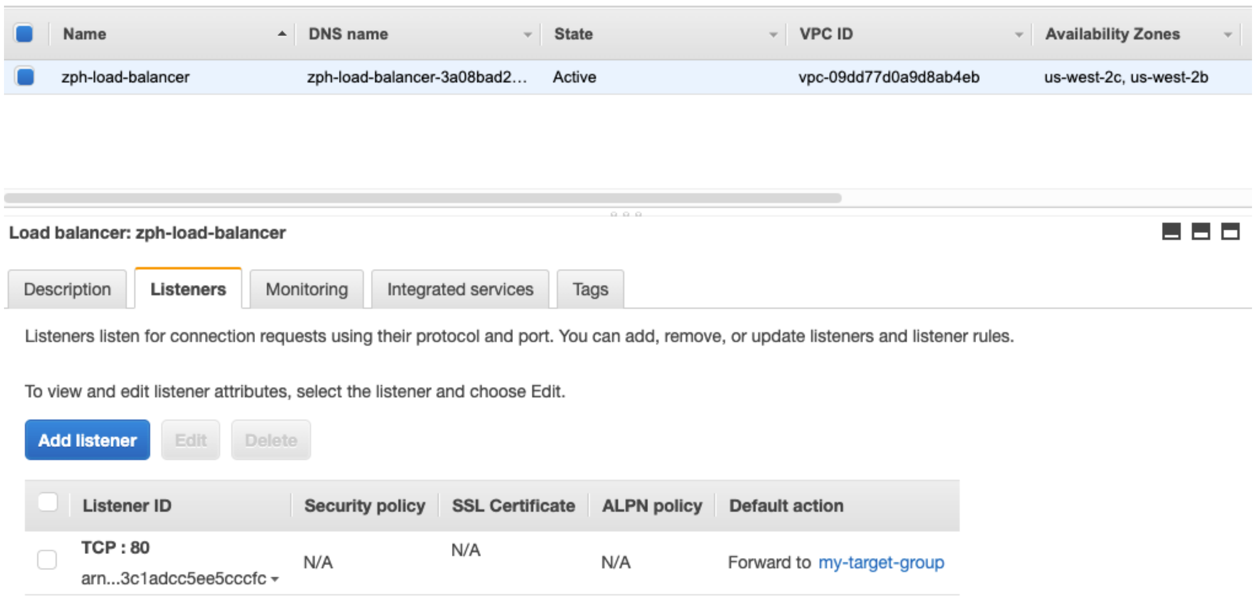
At the beginning I just use the singer ec2 instance to deploy my server, but as I increase the client threads, I can see the throughput seem grows slowly which means the server is overloaded with so many http requests. Then I introduce the AWS network load balancer which listens the tcp 80 port and forwards the request to my two instances with round-robin rule.

Below is the instances, first one is the new ec2 I create using the AMI of old ones. Second one is the original ec2 instance.



	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Ava
	copy-ec2	i-0c64be3ba834db57f	Running	t2.micro	2/2 checks passed	No alarms	us-\
	–	i-08c49a0434ae82f9c	Running	t2.micro	2/2 checks passed	No alarms	us-\

Below is the load balancer I configured



	Name	DNS name	State	VPC ID	Availability Zones
■	zph-load-balancer	zph-load-balancer-3a08bad2...	Active	vpc-09dd77d0a9d8ab4eb	us-west-2c, us-west-2b

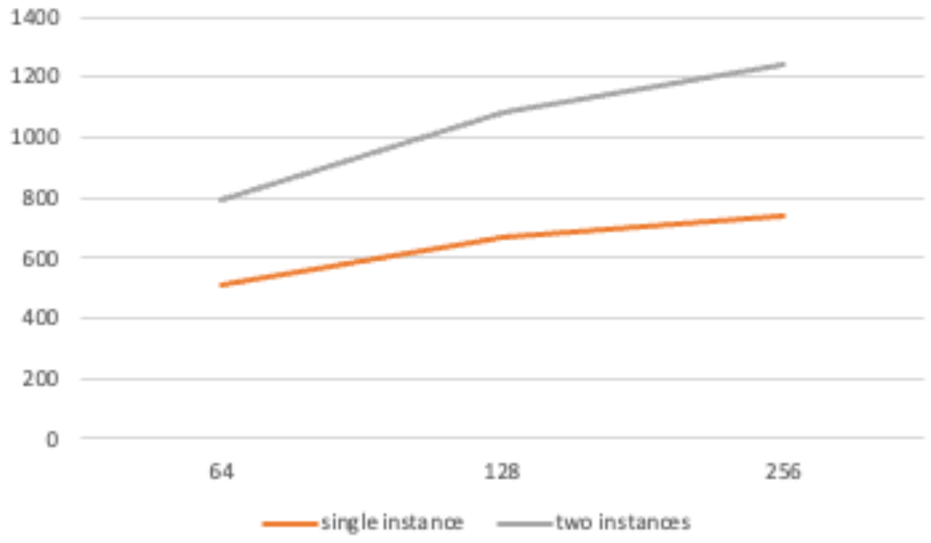
Load balancer: zph-load-balancer

Listeners listen for connection requests using their protocol and port. You can add, remove, or update listeners and listener rules.

To view and edit listener attributes, select the listener and choose Edit.

	Listener ID	Security policy	SSL Certificate	ALPN policy	Default action
☐	TCP : 80 arn...3c1adcc5ee5cccfc	N/A	N/A	N/A	Forward to my-target-group

I draw the chart to illustrate the different throughput for two different server architectures as we increase the client threads.



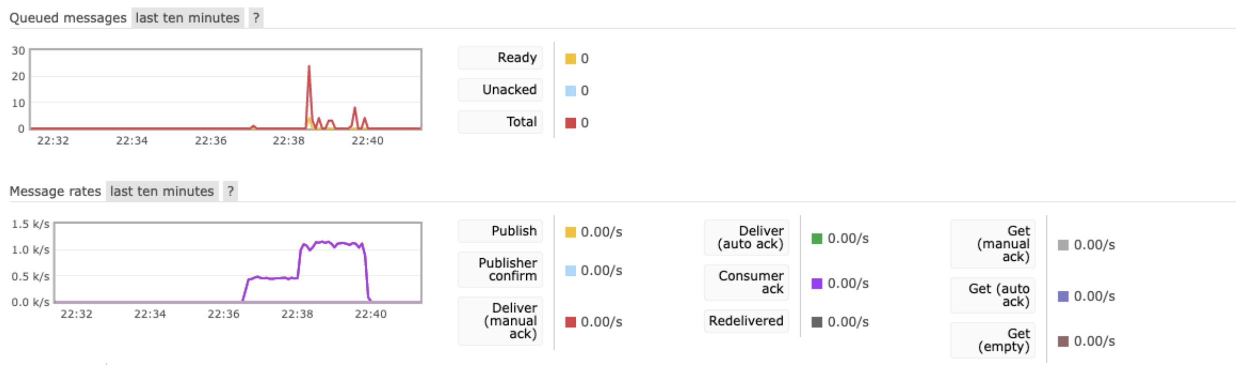
(x -axis refers to number of client threads, y-axis refers to the throughput)

64 client threads

Command lines:

-numThreads 64 -numSkiers 20000 -numLifts 40 -numRuns 10 -endpoint: zph-load-balancer-3a08bad27e8d7b22.elb.us-west-2.amazonaws.com

RMQ management windows



Queued messages last ten minutes ?

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last ten minutes ?

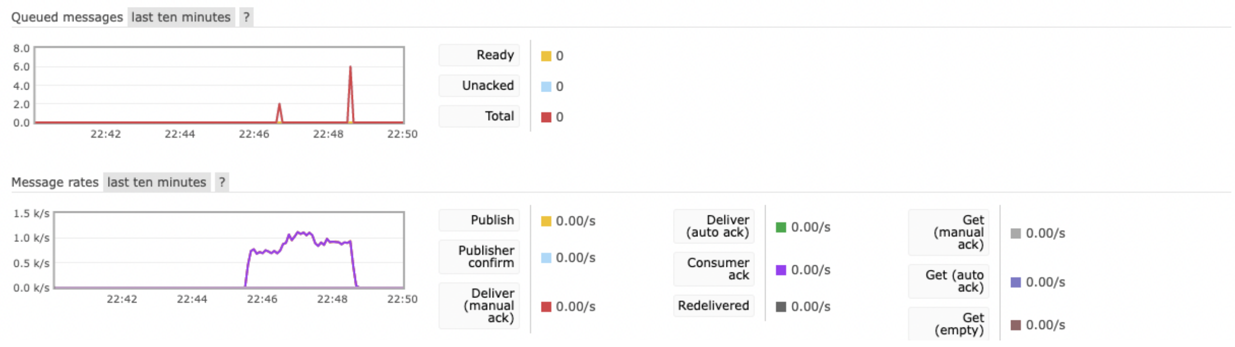
Publish	🟨 0.00/s	Deliver (auto ack)	🟩 0.00/s	Get (manual ack)	⬜ 0.00/s
Publisher confirm	🟦 0.00/s	Consumer ack	🟪 0.00/s	Get (auto ack)	🟦 0.00/s
Deliver (manual ack)	🟥 0.00/s	Redelivered	⬛ 0.00/s	Get (empty)	🟫 0.00/s

128 client threads

Command lines:

-numThreads 128 -numSkiers 20000 -numLifts 40 -numRuns 10 -endpoint: zph-load-balancer-3a08bad27e8d7b22.elb.us-west-2.amazonaws.com

RMQ management windows



Queued messages last ten minutes ?

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last ten minutes ?

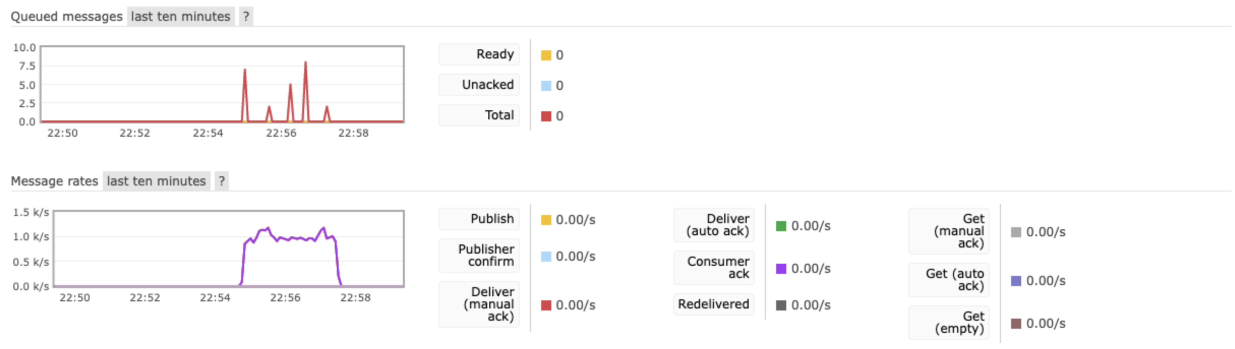
Publish	🟨 0.00/s	Deliver (auto ack)	🟩 0.00/s	Get (manual ack)	⬜ 0.00/s
Publisher confirm	🟦 0.00/s	Consumer ack	🟪 0.00/s	Get (auto ack)	🟦 0.00/s
Deliver (manual ack)	🟥 0.00/s	Redelivered	⬛ 0.00/s	Get (empty)	🟫 0.00/s

256 client threads

Command lines:

-numThreads 256 -numSkiers 20000 -numLifts 40 -numRuns 10 -endpoint: zph-load-balancer-3a08bad27e8d7b22.elb.us-west-2.amazonaws.com

RMQ management windows



Queued messages last ten minutes ?

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

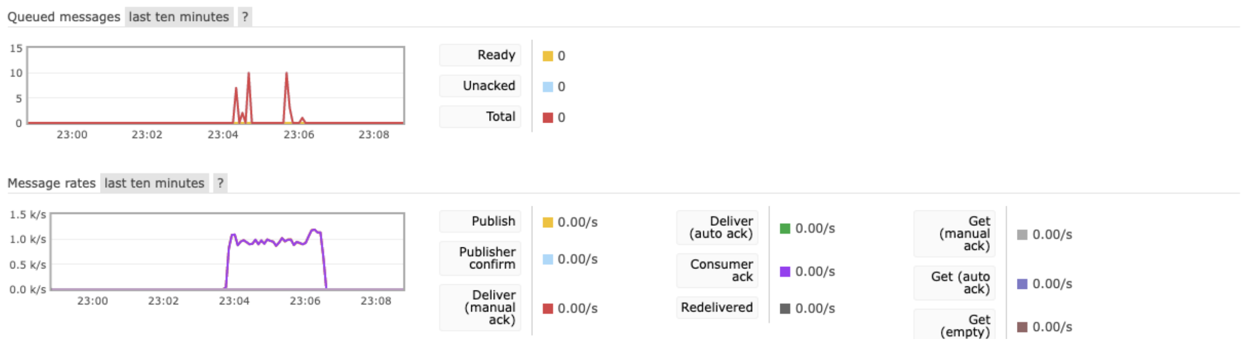
Message rates last ten minutes ?

Publish	🟨 0.00/s	Deliver (auto ack)	🟩 0.00/s	Get (manual ack)	⬜ 0.00/s
Publisher confirm	🟦 0.00/s	Consumer ack	🟪 0.00/s	Get (auto ack)	🟦 0.00/s
Deliver (manual ack)	🟥 0.00/s	Redelivered	⬛ 0.00/s	Get (empty)	🟫 0.00/s

Results for running 512 client threads

Command lines:
-numThreads 512 -numSkiers 20000 -numLifts 40 -numRuns 10 -endpoint: zph-load-balancer-3a08bad27e8d7b22.elb.us-west-2.amazonaws.com

RMQ management windows



Queued messages last ten minutes ?

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last ten minutes ?

Publish	🟨 0.00/s	Deliver (auto ack)	🟩 0.00/s	Get (manual ack)	⬜ 0.00/s
Publisher confirm	🟦 0.00/s	Consumer ack	🟪 0.00/s	Get (auto ack)	🟪 0.00/s
Deliver (manual ack)	🟥 0.00/s	Redelivered	⬛ 0.00/s	Get (empty)	🟫 0.00/s