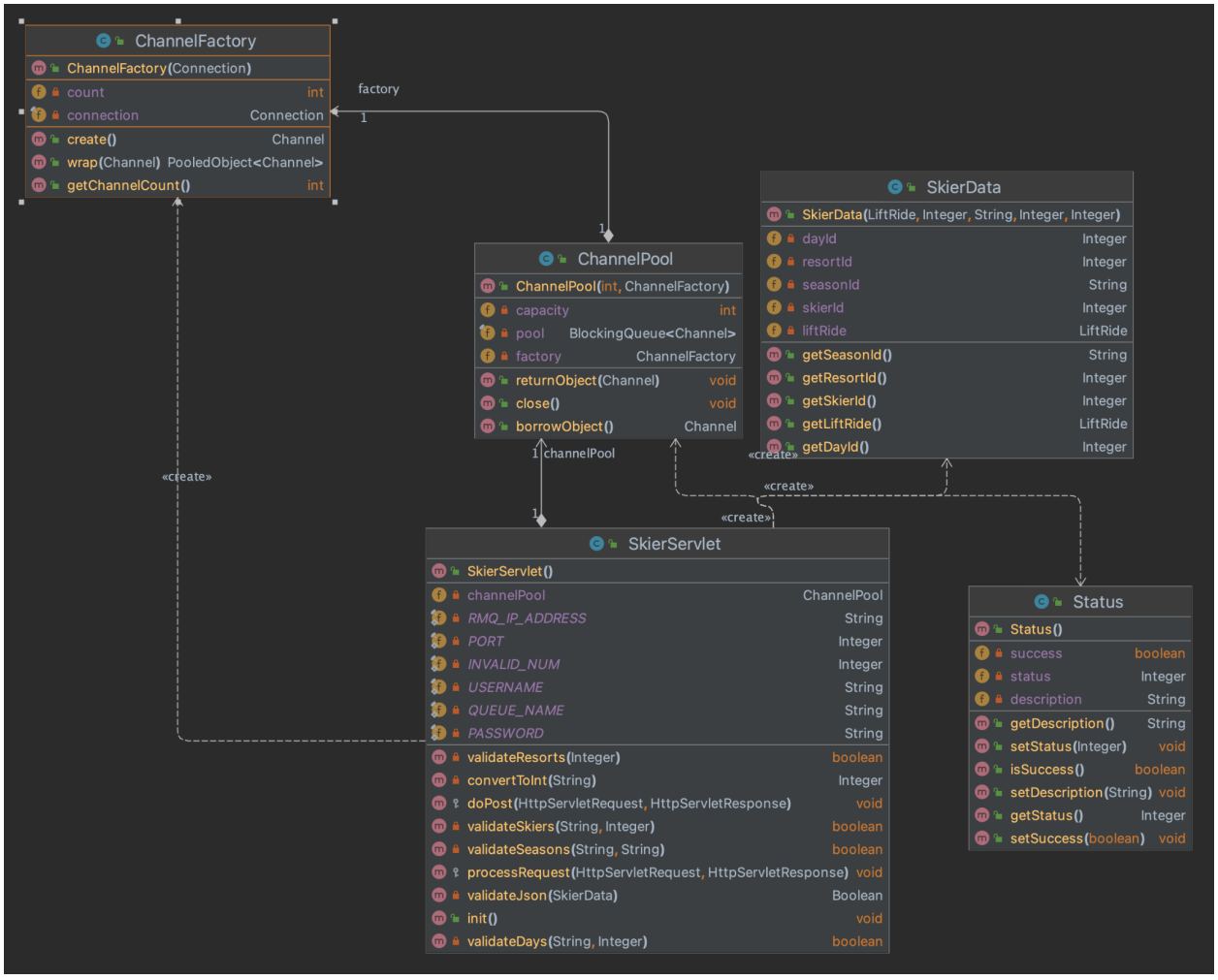
Github URL: https://github.com/jasongautama/cs6650-Distributed-Systems/tree/main/assignment2

Class Design

Design for Servlets (Server)

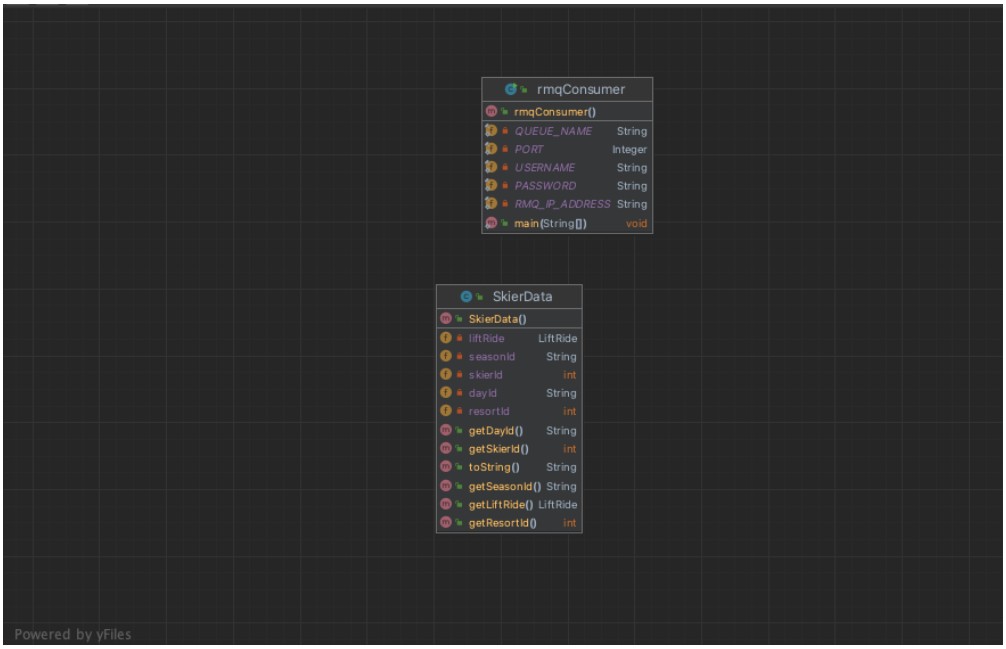
- This class handles the POST request that is validated for the path & LiftRide body when the user sends requests to the EC2 instance.
- Specify the private IP address of the RMQ and keep sending the payload of the **SkierData** to the RMQ
- Created a ***ChannelPool.java*** so we can reuse the thread whenever they have completed their work.



Design for RabbitMQ

- I installed RabbitMQ in the EC2 instance with IP 35.91.190.94 and then tested the Queue (name *skierQueue*) and was able to send the data to the rabbitMQ
- Success:
    - Setup the Security Group (ports, ip) and access the EC2 instance
    - Installed rabbitMQ to EC2 instance
    - Able to create privilege user to access the management console

Design for Consumer

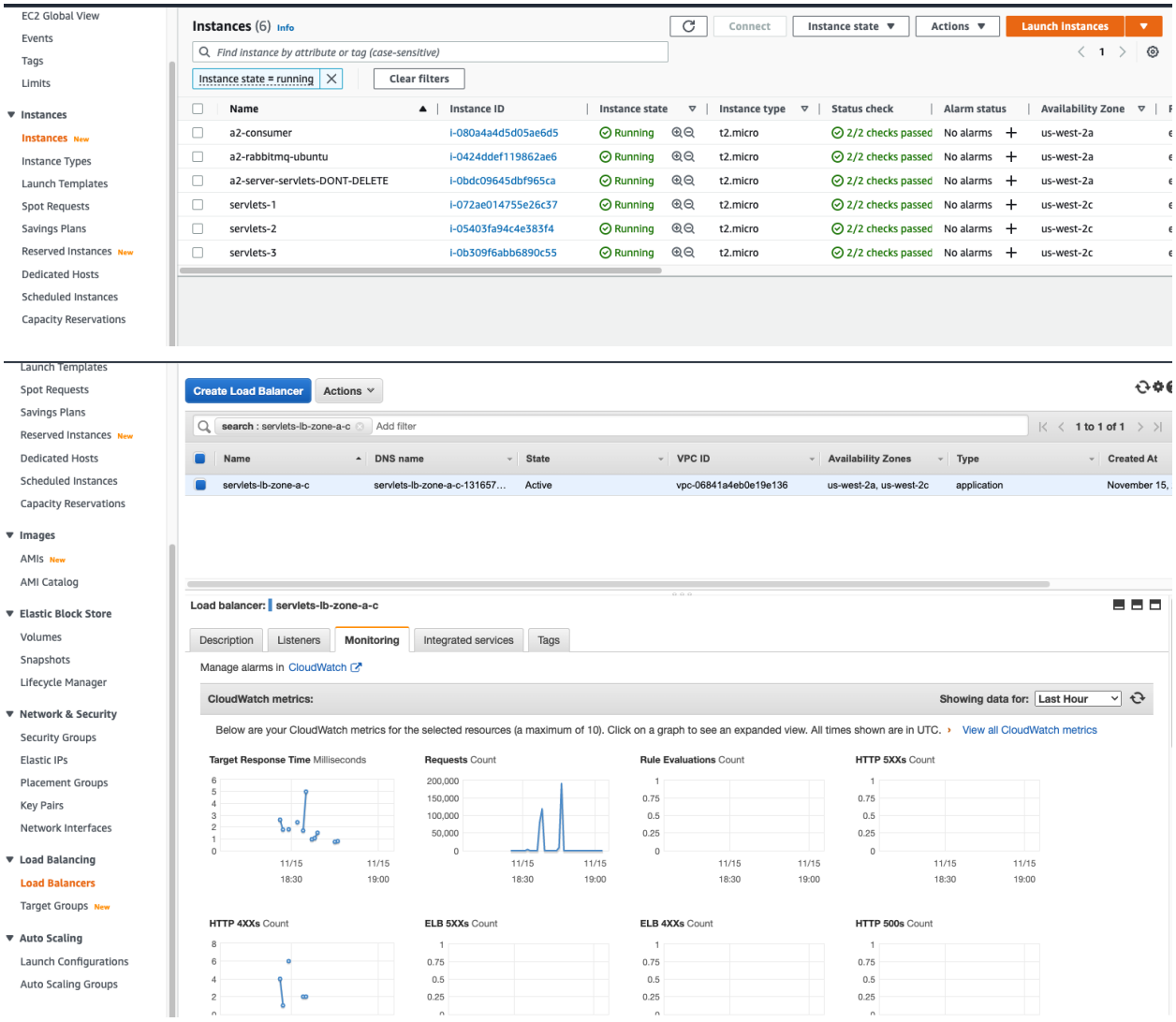


- The consumer here will retrieve the RabbitMQ *SkierQueue* and store the SkierData object into a ConcurrentHashMap()
- Create N threads (30, 100, …) and start consuming the data from the queue

AWS

For EC2 Instance the following is the instances (all using t2.micro):

- 4 x EC2 servlets - accept POST request from client, which is attached to a Load Balancer.
- 1 x EC2 rabbit MQ - queue to store LiftRide object from servlets and consumed by consumer
- 1 x EC2 consumer - run .jar consumer java file that consume from rabbit MQ

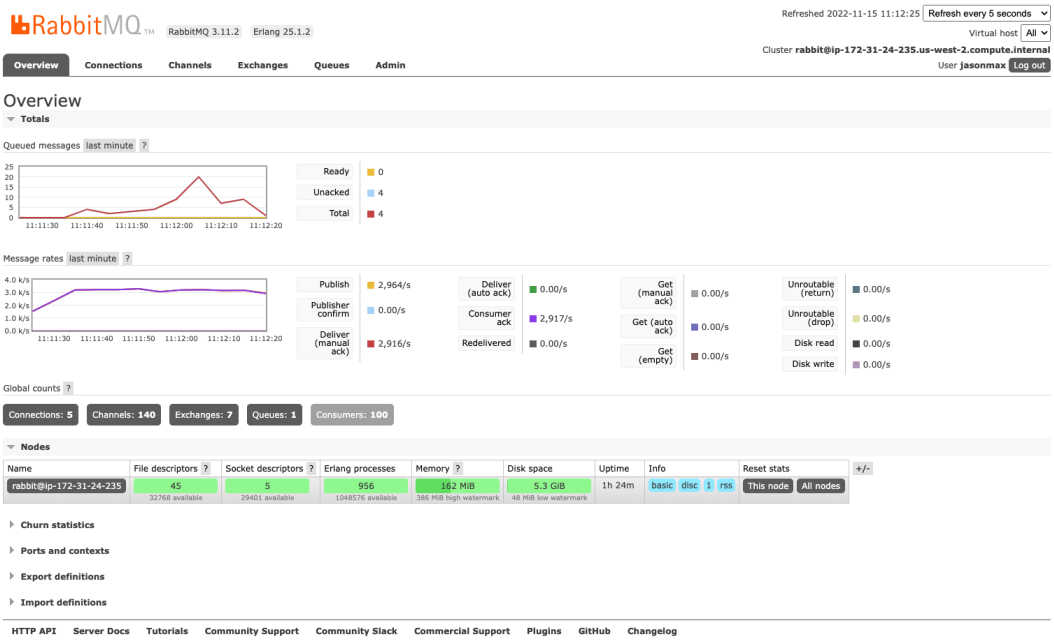

EC2 Global View
Events
Tags
Limits
Instances
Instances New
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances New
Dedicated Hosts
Scheduled Instances
Capacity Reservations
Instances (6) Info
Connect
Instance state
Actions
Launch Instances
Find instance by attribute or tag (case-sensitive)
Instance state = running
Clear filters
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
a2-consumer
i-080a4a4d5d05ae6d5
Running
t2.micro
2/2 checks passed
No alarms
us-west-2a
a2-rabbitmq-ubuntu
i-0424ddef119862ae6
Running
t2.micro
2/2 checks passed
No alarms
us-west-2a
a2-server-servlets-DONT-DELETE
i-0bdc09645dbf965ca
Running
t2.micro
2/2 checks passed
No alarms
us-west-2a
servlets-1
i-072ae014755e26c37
Running
t2.micro
2/2 checks passed
No alarms
us-west-2c
servlets-2
i-05403fa94c4e383f4
Running
t2.micro
2/2 checks passed
No alarms
us-west-2c
servlets-3
i-0b309f6abb6890c55
Running
t2.micro
2/2 checks passed
No alarms
us-west-2c
Launch Templates
Spot Requests
Savings Plans
Reserved Instances New
Dedicated Hosts
Scheduled Instances
Capacity Reservations
Images
AMIs New
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Network Interfaces
Load Balancing
Load Balancers
Target Groups New
Auto Scaling
Launch Configurations
Auto Scaling Groups
Create Load Balancer
Actions
search : servlets-lb-zone-a-c
Add filter
1 to 1 of 1
Name
DNS name
State
VPC ID
Availability Zones
Type
Created At
servlets-lb-zone-a-c
servlets-lb-zone-a-c-131657...
Active
vpc-06841a4eb0e19e136
us-west-2a, us-west-2c
application
November 15,
Load balancer: servlets-lb-zone-a-c
Description
Listeners
Monitoring
Integrated services
Tags
Manage alarms in CloudWatch
CloudWatch metrics:
Showing data for:
Last Hour
Below are your CloudWatch metrics for the selected resources (a maximum of 10). Click on a graph to see an expanded view. All times shown are in UTC.
View all CloudWatch metrics
Target Response Time Milliseconds
Requests Count
Rule Evaluations Count
HTTP 5XXs Count
HTTP 4XXs Count
ELB 5XXs Count
ELB 4XXs Count
HTTP 500s Count
11/15 18:30
11/15 19:00

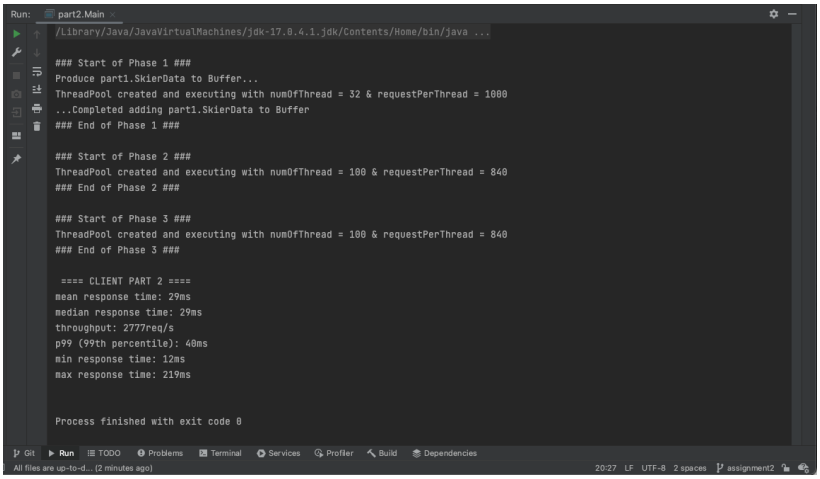
[Test Case Scenario] No Load Balancer

Explanation: Looking at the RabbitMQ management console, we can see how the message rates plateau around 3000 msg/s. The number indicates the throughput of the message getting received by the queue. Also, the Publish/Consumer-ack have more or less equal number, which is what we want

Phase 1: 32 threads @ 1,000 request / thread
Phase 2: 100 threads @ 840 request / thread
Phase 3: 100 threads @ 840 request / thread
# of Consumer thread: 30 threads

==== CLIENT PART 2 ====
mean response time: 29ms
median response time: 29ms
throughput: 2,777req/s
p99 (99th percentile): 40ms
min response time: 12ms
max response time: 219ms





[Test Case Scenario] Load Balancer (4 x EC2 t2.micro Instances)

Test with Load Balancer - t2.micro instances x4
Explanation: with 3 additional servlets, we can see that the **throughput increased by 30% in** comparison to a single servlet. From the RMQ console, we can see that the message rates increase to plateau at 4.5k/s *(50% increase in message rates)* as well as the publish/consumer ack is 1.5x faster.

In order to meet the baseline where production_rate ≈ consumption_rate, I increased the amount of thread from 30 to 100 threads on the consumer side to retrieve from the queue to reduce the amount of messages waiting in the Queue. If there's not enough threads to consume the queue, we will see a spike in the graph (ie /\), which tells me it was not consuming in equal amount it's producing.

Phase 1: 32 threads @ 1,000 request / thread
Phase 2: 100 threads @ 840 request / thread
Phase 3: 100 threads @ 840 request / thread

# of Consumer thread: 100 threads
==== CLIENT PART 2 ====
mean response time: 20ms
median response time: 20ms
throughput: 3636req/s
p99 (99th percentile): 27ms
min response time: 13ms
max response time: 237ms

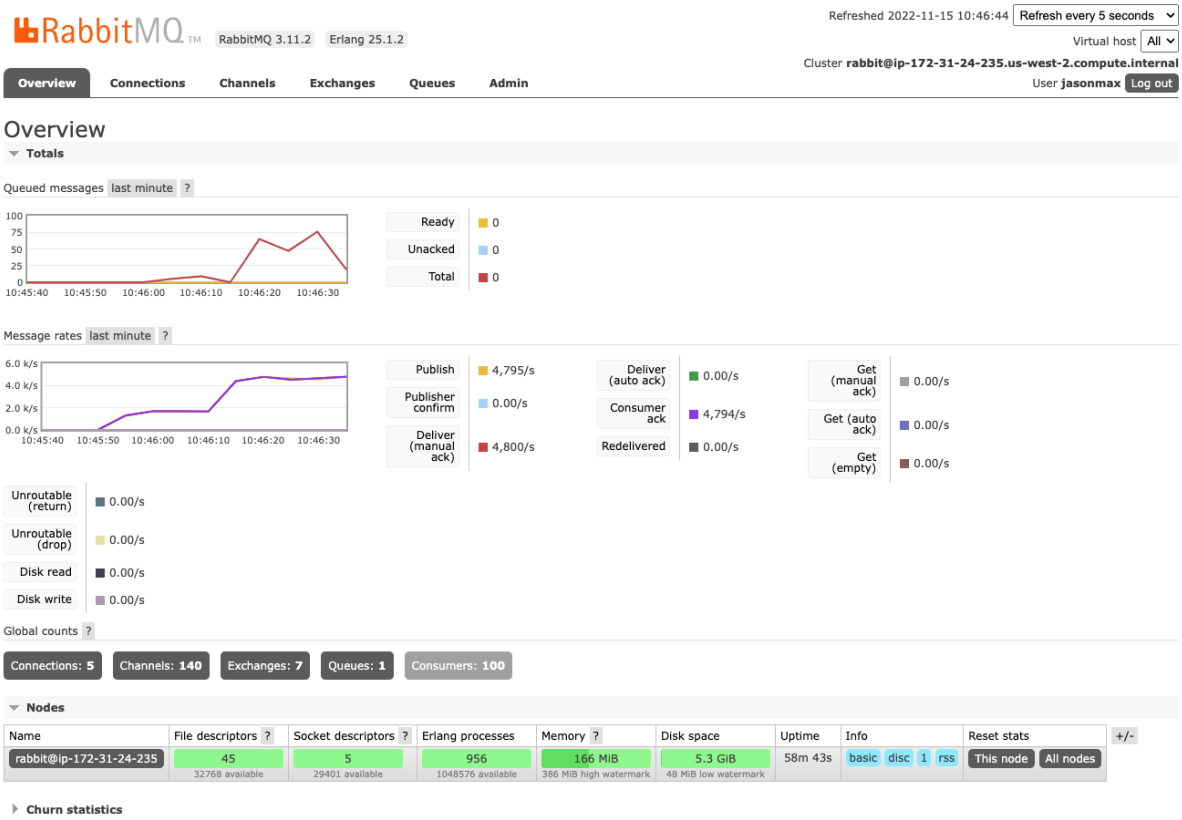


Image showing the RMQ management console where the rate of producing/consuming are almost equal

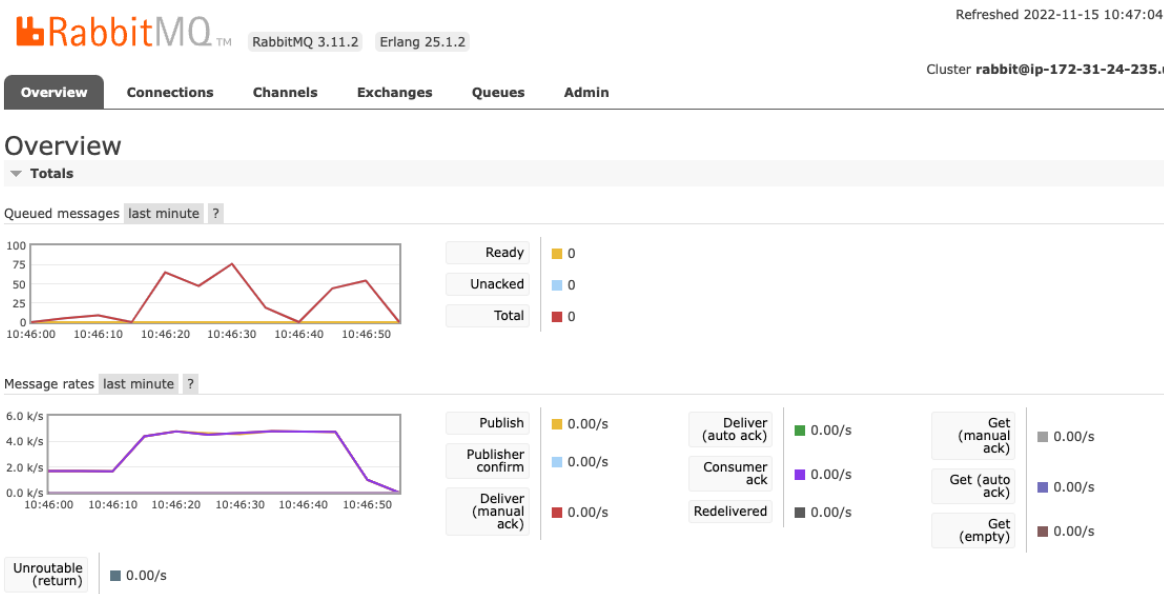
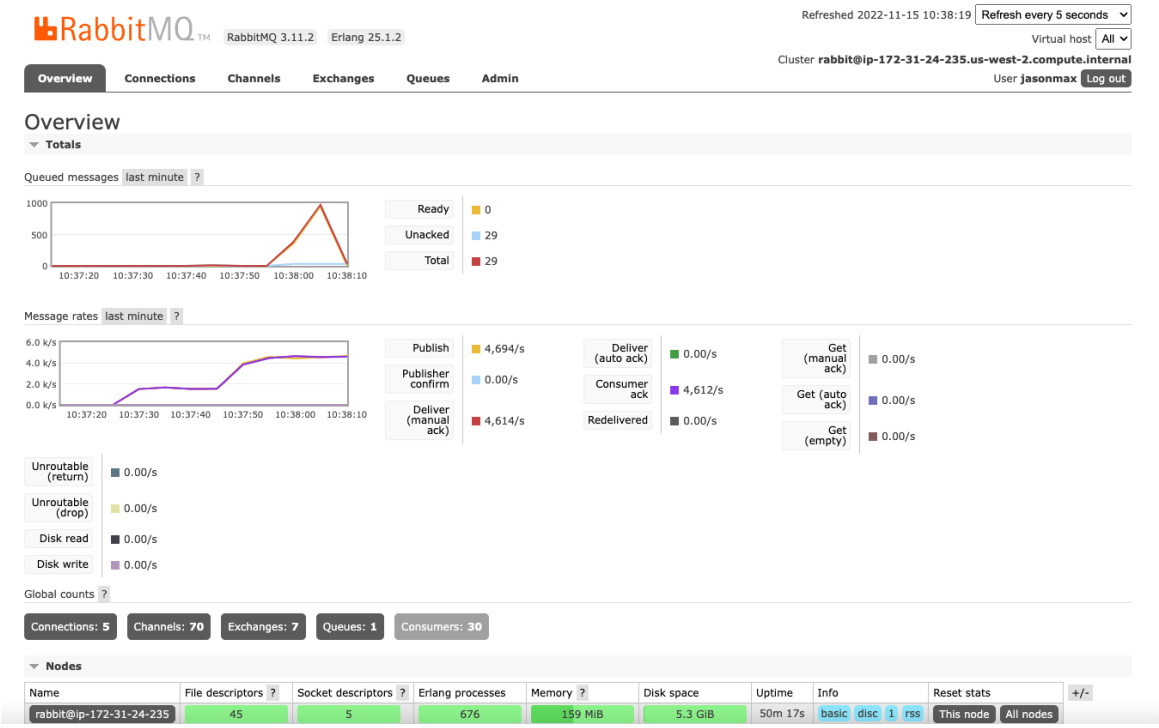


Image showing the RMQ management console receiving messages from 4 t2.micro EC2 instances with the use of Load Balancer



Example where not enough consumer consume the queue that cause a spike /\ in the RMQ