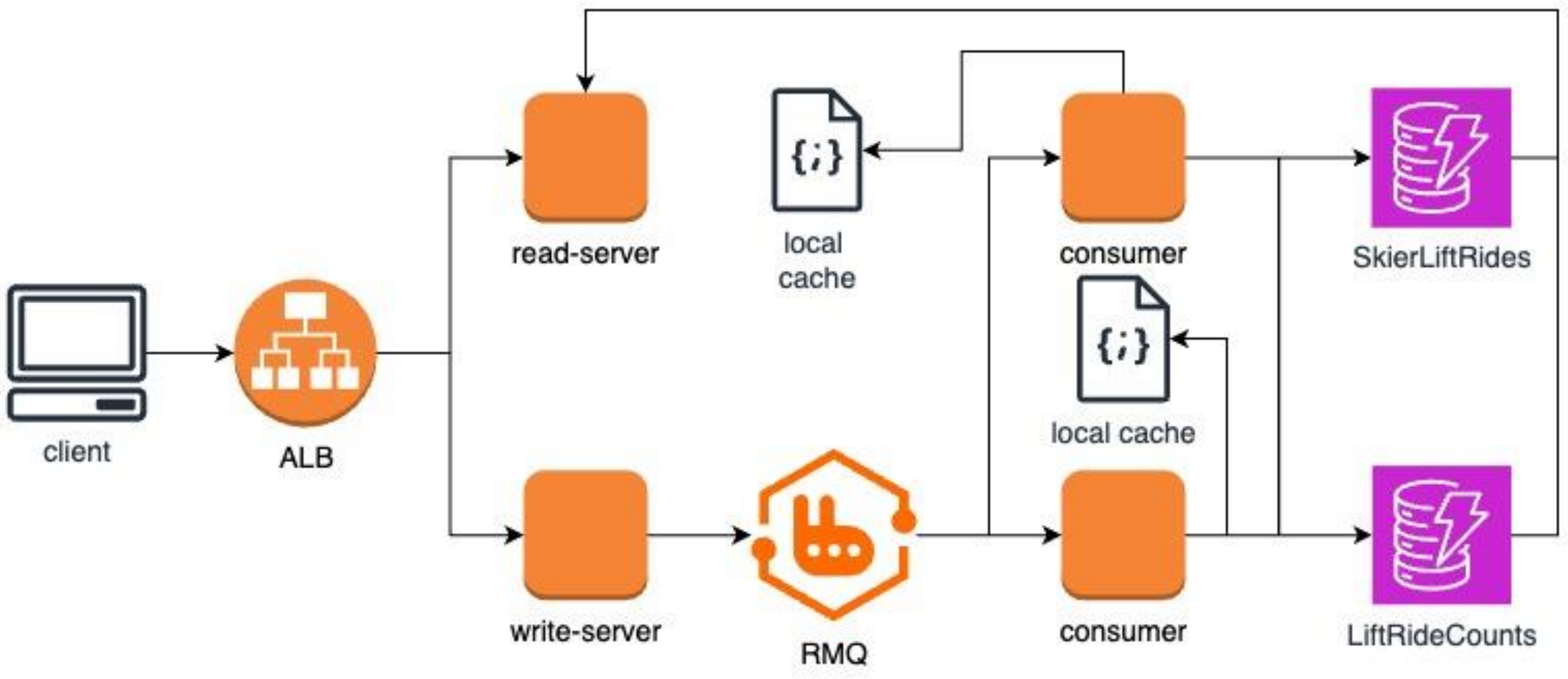
Architecture

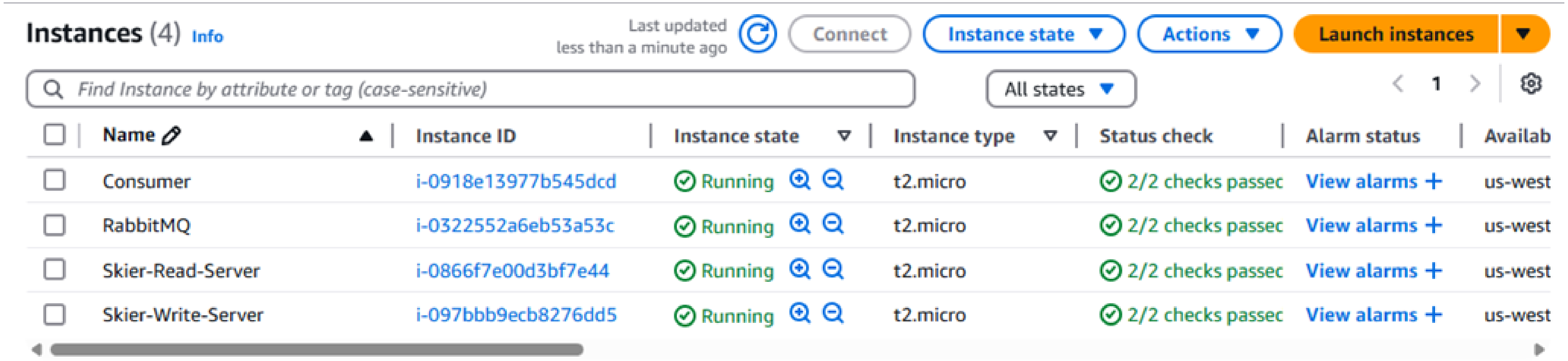


Architecture

- Load Balancer
  - distributes incoming traffic between read-server and write-server to ensure scalability and fault tolerance
- Write-server
  - receives write requests from clients and publishes messages RMQ
- Consumer
  - Messages from RMQ are acked first in batches before being written to DynamoDB
  - Messages are periodically flushed from memory to disk as a fail-safe
- Read-server
  - serves client queries, fetching data from DynamoDB
- DynamoDB
  - 2 Tables:
    - SkierLiftRides for detailed ride logs
    - LiftRideCounts for aggregated ride count per resort/year/day

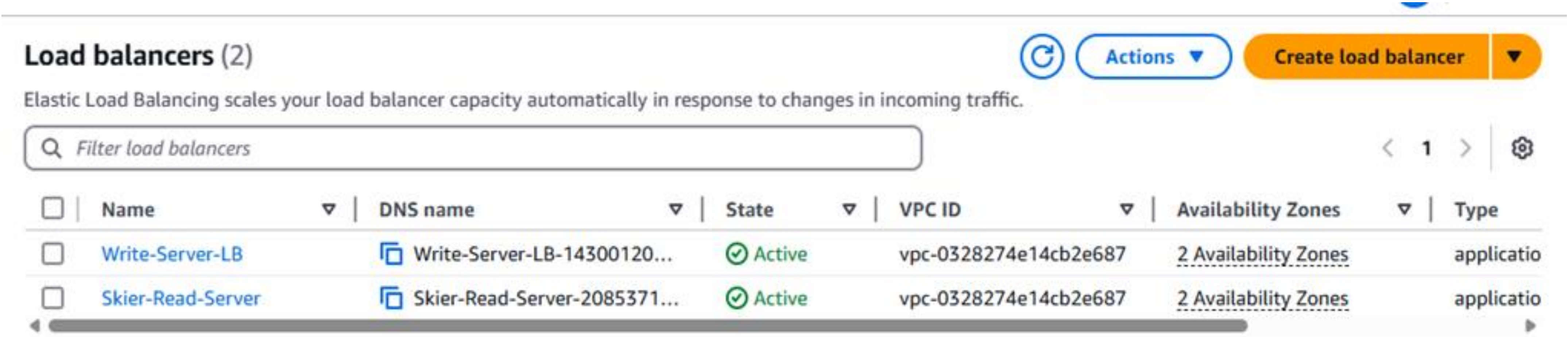
Deployment

1. E2 Instances



☐	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availab
☐	Consumer	i-0918e13977b545dcd	✓ Running	t2.micro	✓ 2/2 checks passec	View alarms +	us-west
☐	RabbitMQ	i-0322552a6eb53a53c	✓ Running	t2.micro	✓ 2/2 checks passec	View alarms +	us-west
☐	Skier-Read-Server	i-0866f7e00d3bf7e44	✓ Running	t2.micro	✓ 2/2 checks passec	View alarms +	us-west
☐	Skier-Write-Server	i-097bbb9ecb8276dd5	✓ Running	t2.micro	✓ 2/2 checks passec	View alarms +	us-west

2. Load balancers



Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

☐	Name	DNS name	State	VPC ID	Availability Zones	Type
☐	Write-Server-LB	Write-Server-LB-14300120...	✓ Active	vpc-0328274e14cb2e687	2 Availability Zones	applicatio
☐	Skier-Read-Server	Skier-Read-Server-2085371...	✓ Active	vpc-0328274e14cb2e687	2 Availability Zones	applicatio

Data model

We used **Amazon DynamoDB** with two tables for faster key-based lookup and AWS fully-management.

SkierLiftRides Table

Attribute	Type	Purpose
SkierID	Number	Partition Key
SeasonDayTimeID	String	Sort Key
ResortID	Number	To group rides by resort
SeasonID	Number	For filtering season
LiftID	Number	To calculate vertical
Time	Number	Ride times

SkierLiftRides (Main Table)

- Stores every individual ride event
- Composite key: SeasonID#DayID#Time ensures uniqueness and time-order
- Used for all skier-level vertical calculations

LiftRideCounts (Counter Table)

- Stores **daily totals** per resort
- Composite key: ResortID#SeasonID#DayID
- Fast response for GET /resorts/…/skiers API

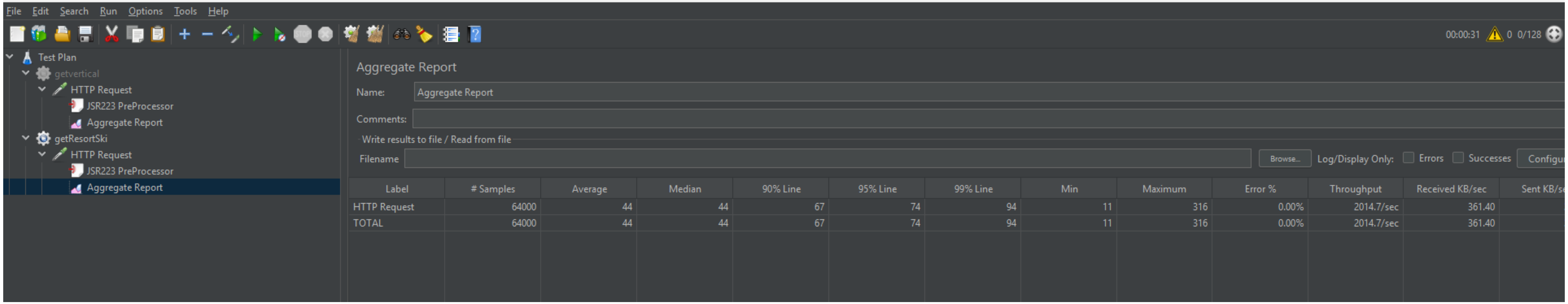
Trade-offs: Two tables increase heavier write loads, but we handled it by batching writes in memory and periodically flushing in the background.

LiftRideCounts

Attribute	Type	Purpose
ResortSeasonDayID	String	Partition Key
Count	Number	Total number of rides on that day

JMeter Results

1. GetResortSki: GET/resorts/{resortID}/seasons/{seasonID}/day/{dayID}/skiers



Aggregate Report

Name: Aggregate Report

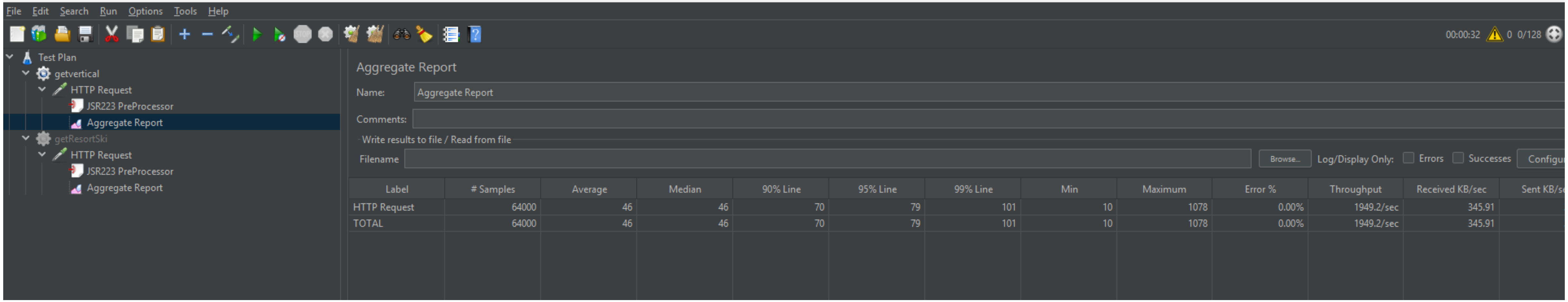
Comments:

Write results to file / Read from file

Filename Browse... Log/Display Only: □ Errors □ Successes Configu

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/se
HTTP Request	64000	44	44	67	74	94	11	316	0.00%	2014.7/sec	361.40	
TOTAL	64000	44	44	67	74	94	11	316	0.00%	2014.7/sec	361.40	

2. GetVertical: GET/skiers/{skierID}/vertical



Aggregate Report

Name: Aggregate Report

Comments:

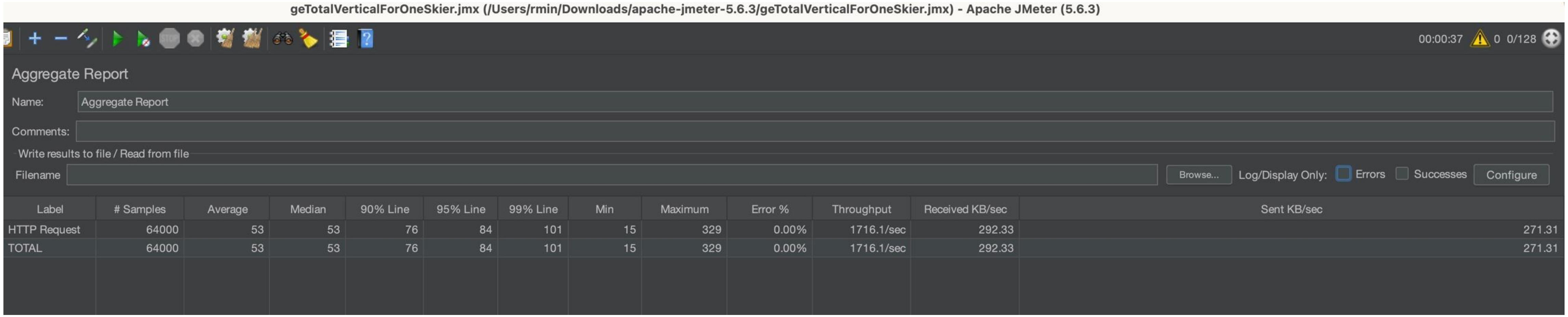
Write results to file / Read from file

Filename Browse... Log/Display Only: □ Errors □ Successes Configu

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/se
HTTP Request	64000	46	46	70	79	101	10	1078	0.00%	1949.2/sec	345.91	
TOTAL	64000	46	46	70	79	101	10	1078	0.00%	1949.2/sec	345.91	

JMeter Results

3. GET/skiers/{resortID}/seasons/{seasonID}/days/{dayID}/skiers/{skierID}



geTotalVerticalForOneSkier.jmx (/Users/rmin/Downloads/apache-jmeter-5.6.3/geTotalVerticalForOneSkier.jmx) - Apache JMeter (5.6.3)

Aggregate Report

Name: Aggregate Report

Comments:

Write results to file / Read from file

Filename

Log/Display Only: □ Errors □ Successes

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
HTTP Request	64000	53	53	76	84	101	15	329	0.00%	1716.1/sec	292.33	271.31
TOTAL	64000	53	53	76	84	101	15	329	0.00%	1716.1/sec	292.33	271.31

Future Enhancements

- ## Better Cache
  - API Endpoint: GET /resorts/{resortID}/seasons/{seasonID}/day/{dayID}/skiers
  - Problem:
    - High read frequency
    - Results in increased latency and high database load
  - Add In – Memory Cache (Redis)
  - Cache Invalidation Logic (POST):
    If the skier is new for the day, delete the corresponding Redis key to ensure the next GET returns the updated unique skier count; otherwise, do nothing.
- ## Throttling
  - Monitor RabbitMQ queue size in real time
  - Token Bucket Rate Limiting:
    - Dynamically adjust token refill rate based on current queue size
    - Low queue: allow more tokens (higher throughput)
    - High queue: fewer tokens (slower intake)
  - Circuit Breaker for Failure Isolation
    - Open the circuit when queue remains overloaded
  - Prevent system overload when consumers are falling behind