SANTA CLARA UNIVERSITY
School of Engineering

Bronco Career Alerts

Chaitanya Kothapalli(W1653623)

Lasya Malladi(W1650491)

Venkata Sai Srikar Jilla(W1652687)

Introduction

Background: Campus Buzz

- **Exciting Campus Life:** Universities are buzzing with important job events, but students can't keep up because there's just too much going on.

Challenge: Email Overload

- **Emails Everywhere:** Regular emails are a mess, bombarding students with tons of info, and they end up missing the good stuff.

Solution: Smart "Bronco-Career Alerts" Pub/Sub System

- **Smart Fix:** We made "Bronco-Career Alerts," a Pub/Sub system that sends students personalized event messages.
- **Student Power:** Students get to choose what they want to hear about, so they only get info that matters to them.

Related Work & Challenges

Flexible Communication Models:

- Research emphasizes the need for dynamic communication in large systems.
- "Bronco-Career Alerts" adopts subscription aging for enhanced performance.

Microservice Communication Survey:

- Highlights pub/sub in microservices for real-time applications.
- Influences "Bronco-Career Alerts" event-based communication approach.

Challenges:

- Overcoming Information overload-Mass emails causing information overload.
- Consistency in Event Ordering-Maintaining order in notifications.
- Fault Tolerance and Reliability-Ensuring continuous service despite system faults.

Design Choices

- **Microservices Architecture:** Utilizing Flask to create lightweight RESTful services (producer and consumer) aligns with the microservices approach, ensuring scalability and maintainability.
- **Messaging with RabbitMQ:** Choosing RabbitMQ as the message broker allows for reliable and scalable asynchronous message processing.The direct exchange type was selected for specific routing between producers and consumers, enabling precise message delivery.
- **Containerization with Kubernetes:** Deploying on Kubernetes suggests a decision for high availability, load balancing, automated deployment, and scaling.
- **Persistent volume configurations (kubernetes volumes):** Ensure that data is retained across pod restarts.
- **Topic-Based Routing:** Messages are routed based on topics, enabling selective message consumption.

Challenges

- **Service Orchestration:** Coordinating multiple services can become complex. Ensuring efficient and reliable communication between services, especially as the number of services scales up.

- **Message Durability and Delivery Guarantees:** Configuring RabbitMQ to ensure messages are not lost in transit, particularly in the event of service failures.

- **High Volumes:** Tuning RabbitMQ to handle high volumes of messages without significant delays or backlogs and Balancing the load across consumers effectively to prevent any single consumer from being overwhelmed.

- **Debugging:** Monitoring and tracing the flow of messages through RabbitMQ and Flask services in a distributed system is complex but is essential for diagnosing issues.

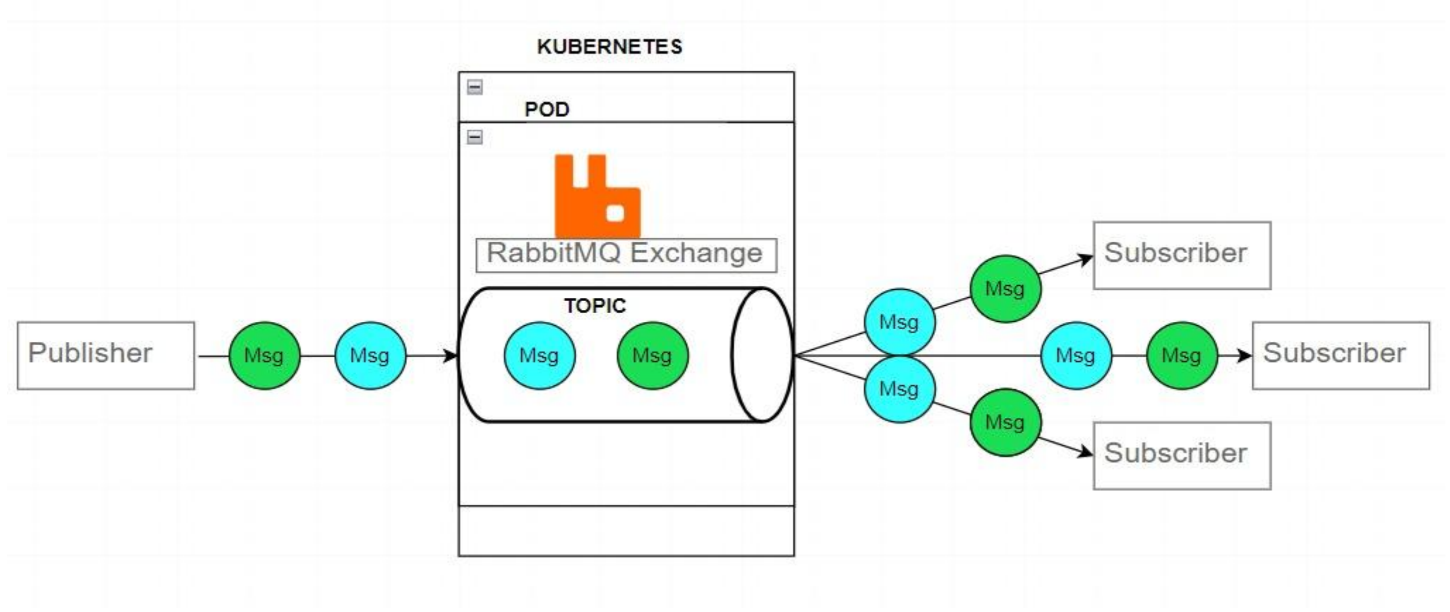
Techstack

Flask

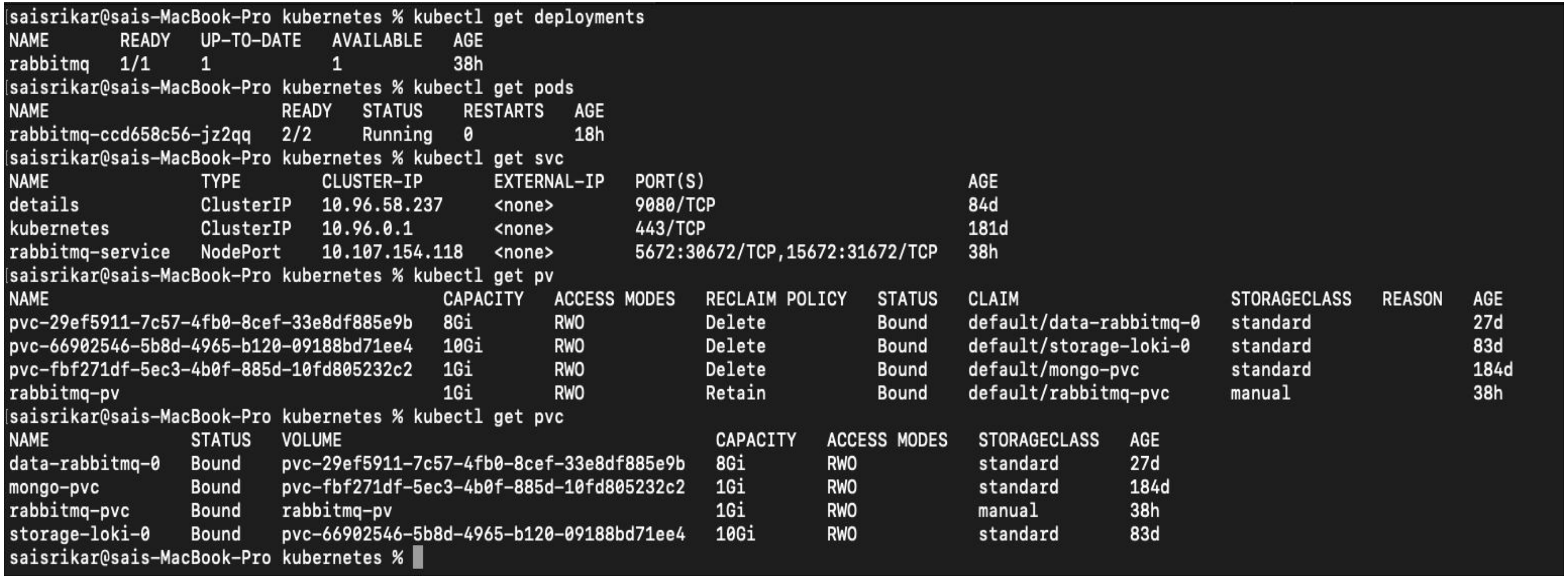
RabbitMQ

minikube

Architecture-Overview

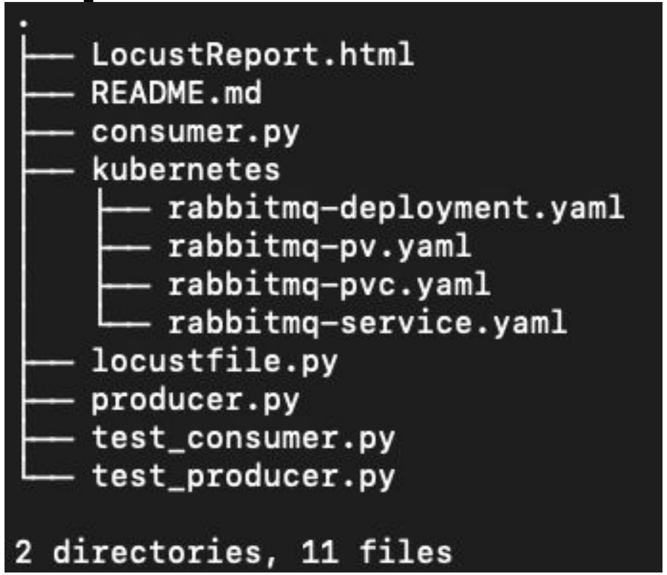


Kubernetes Objects



```
[saisrikar@sais-MacBook-Pro kubernetes % kubectl get deployments
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
rabbitmq   1/1     1            1           38h
[saisrikar@sais-MacBook-Pro kubernetes % kubectl get pods
NAME                       READY   STATUS    RESTARTS   AGE
rabbitmq-ccd658c56-jz2qq   2/2     Running   0          18h
[saisrikar@sais-MacBook-Pro kubernetes % kubectl get svc
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                          AGE
details            ClusterIP   10.96.58.237     <none>        9080/TCP                         84d
kubernetes         ClusterIP   10.96.0.1        <none>        443/TCP                          181d
rabbitmq-service   NodePort    10.107.154.118   <none>        5672:30672/TCP,15672:31672/TCP   38h
[saisrikar@sais-MacBook-Pro kubernetes % kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                      STORAGECLASS   REASON   AGE
pvc-29ef5911-7c57-4fb0-8cef-33e8df885e9b   8Gi        RWO            Delete           Bound    default/data-rabbitmq-0    standard                27d
pvc-66902546-5b8d-4965-b120-09188bd71ee4   10Gi       RWO            Delete           Bound    default/storage-loki-0     standard                83d
pvc-fbf271df-5ec3-4b0f-885d-10fd805232c2   1Gi        RWO            Delete           Bound    default/mongo-pvc          standard                184d
rabbitmq-pv                                1Gi        RWO            Retain           Bound    default/rabbitmq-pvc       manual                  38h
[saisrikar@sais-MacBook-Pro kubernetes % kubectl get pvc
NAME              STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
data-rabbitmq-0   Bound    pvc-29ef5911-7c57-4fb0-8cef-33e8df885e9b   8Gi        RWO            standard       27d
mongo-pvc         Bound    pvc-fbf271df-5ec3-4b0f-885d-10fd805232c2   1Gi        RWO            standard       184d
rabbitmq-pvc      Bound    rabbitmq-pv                                1Gi        RWO            manual         38h
storage-loki-0    Bound    pvc-66902546-5b8d-4965-b120-09188bd71ee4   10Gi       RWO            standard       83d
saisrikar@sais-MacBook-Pro kubernetes %
```

Repo Structure



```
.
├── LocustReport.html
├── README.md
├── consumer.py
├── kubernetes
│   ├── rabbitmq-deployment.yaml
│   ├── rabbitmq-pv.yaml
│   ├── rabbitmq-pvc.yaml
│   └── rabbitmq-service.yaml
├── locustfile.py
├── producer.py
├── test_consumer.py
└── test_producer.py

2 directories, 11 files
```

GITHUB: github.com/jvsaisrikar/COEN317-BroncoJobAlerts

Live Demo

Evaluation

- **Decoupled Microservices Architecture:** The design enables services to communicate asynchronously, reducing dependencies between them. This is beneficial for the scalability and maintainability of the system.

- **Scalability:** RabbitMQ effectively handles high-throughput and high-volume messaging, which is essential for real-time processing systems. The use of separate queues and topics supports scaling.

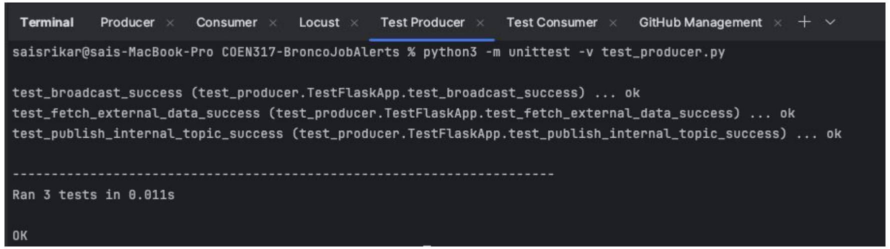
- **Load Testing with Locust:** The presence of a single failure in the publish request indicates a generally stable system but also highlights a area for improvement.

- **Reliability and Performance**: The consumer service uses a prefetch_count=1 which helps in distributing the load evenly across consumers.

Metrics for Evaluation

- **Throughput:** Measured in Requests Per Second (RPS), the system seems to handle a significant load with a maximum RPS of 42 for aggregated requests.
- **Latency:** The response time statistics show that the system has varying latencies for different operations. For instance, the subscribe and unsubscribe operations have higher maximum response times, which might indicate they are more resource-intensive.
- **Error Rate:** There is a low error rate observed in the load test, with only one publish operation failing. This indicates that the system is relatively stable under the tested conditions.

Unit Testing

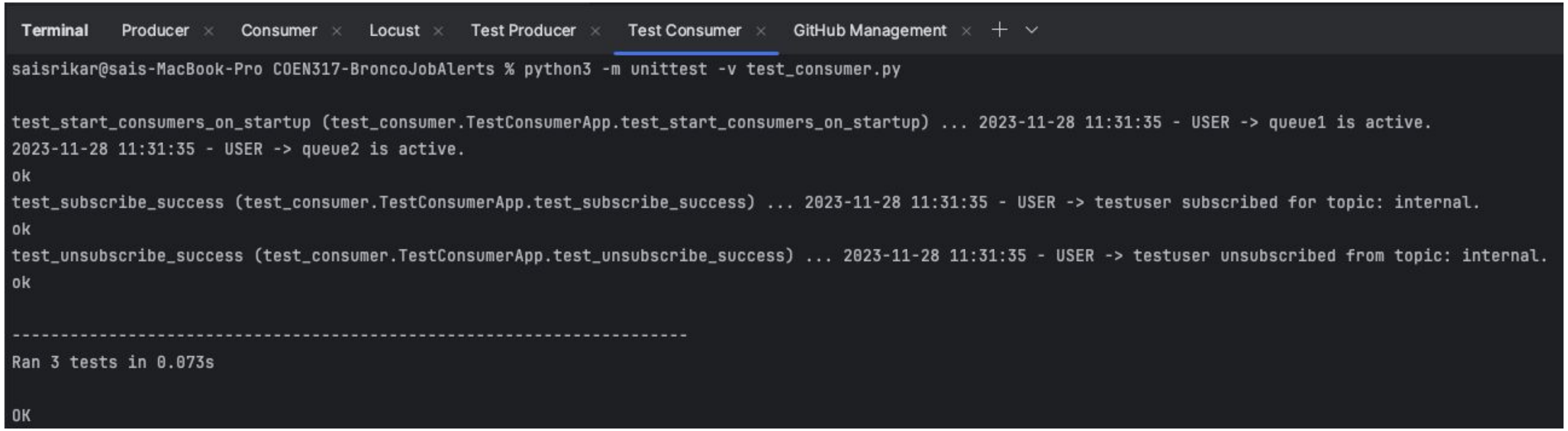


```
saisrikar@sais-MacBook-Pro COEN317-BroncoJobAlerts % python3 -m unittest -v test_producer.py

test_broadcast_success (test_producer.TestFlaskApp.test_broadcast_success) ... ok
test_fetch_external_data_success (test_producer.TestFlaskApp.test_fetch_external_data_success) ... ok
test_publish_internal_topic_success (test_producer.TestFlaskApp.test_publish_internal_topic_success) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.011s

OK
```



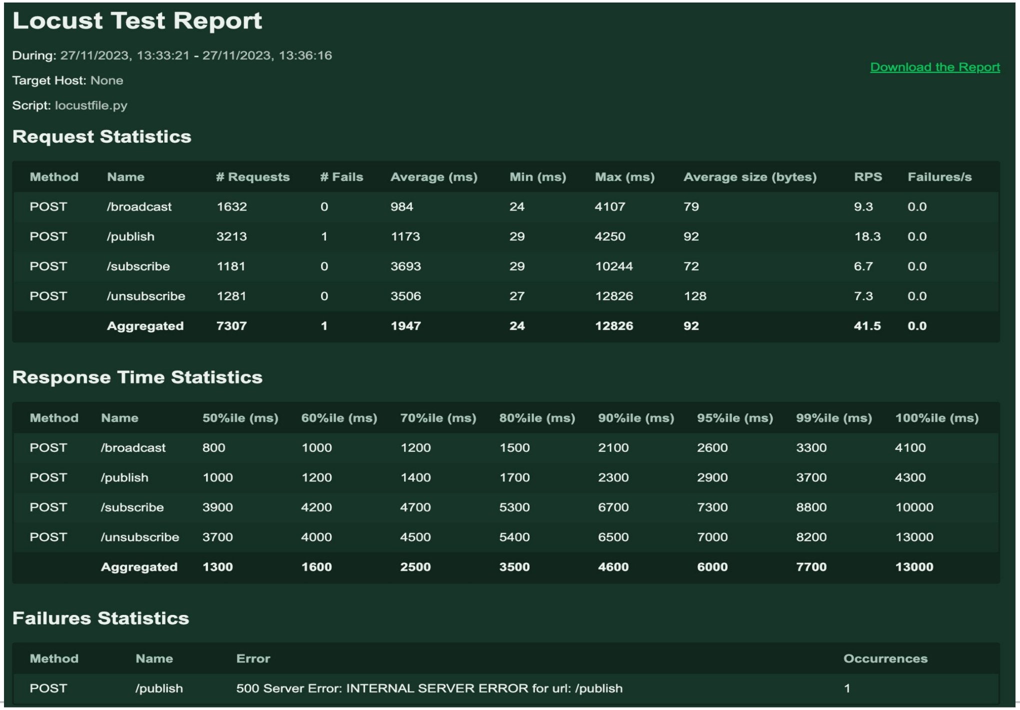
```
saisrikar@sais-MacBook-Pro COEN317-BroncoJobAlerts % python3 -m unittest -v test_consumer.py

test_start_consumers_on_startup (test_consumer.TestConsumerApp.test_start_consumers_on_startup) ... 2023-11-28 11:31:35 - USER -> queue1 is active.
2023-11-28 11:31:35 - USER -> queue2 is active.
ok
test_subscribe_success (test_consumer.TestConsumerApp.test_subscribe_success) ... 2023-11-28 11:31:35 - USER -> testuser subscribed for topic: internal.
ok
test_unsubscribe_success (test_consumer.TestConsumerApp.test_unsubscribe_success) ... 2023-11-28 11:31:35 - USER -> testuser unsubscribed from topic: internal.
ok

----------------------------------------------------------------------
Ran 3 tests in 0.073s

OK
```

Load Testing using Locust



Locust Test Report

During: 27/11/2023, 13:33:21 - 27/11/2023, 13:36:16

Target Host: None

Script: locustfile.py

Download the Report

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/broadcast	1632	0	984	24	4107	79	9.3	0.0
POST	/publish	3213	1	1173	29	4250	92	18.3	0.0
POST	/subscribe	1181	0	3693	29	10244	72	6.7	0.0
POST	/unsubscribe	1281	0	3506	27	12826	128	7.3	0.0
	Aggregated	**7307**	**1**	**1947**	**24**	**12826**	**92**	**41.5**	**0.0**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	/broadcast	800	1000	1200	1500	2100	2600	3300	4100
POST	/publish	1000	1200	1400	1700	2300	2900	3700	4300
POST	/subscribe	3900	4200	4700	5300	6700	7300	8800	10000
POST	/unsubscribe	3700	4000	4500	5400	6500	7000	8200	13000
	Aggregated	**1300**	**1600**	**2500**	**3500**	**4600**	**6000**	**7700**	**13000**

Failures Statistics

Method	Name	Error	Occurrences
POST	/publish	500 Server Error: INTERNAL SERVER ERROR for url: /publish	1

Future Plans

Optimization and Tuning:

- Given the latency and the single failure observed, the system might need optimization. This could involve tuning RabbitMQ configurations, optimizing Flask application code, or improving Kubernetes resource allocation.

Scalability Testing:

- Beyond the current load tests, plans to include testing the system's scalability by incrementally increasing the load until it reaches the system's maximum capacity. This will help identify at what point the system needs to scale out.

High Availability and Fault Tolerance:

- Implementing high-availability configurations for RabbitMQ and ensuring that the Kubernetes deployment can handle node failures without service disruption could be a priority.

Monitoring and Alerting:

- Setting up comprehensive monitoring and alerting for both the Flask application and RabbitMQ within Kubernetes to get real-time insights into the system's performance and health status.

Summary

- Microservices Architecture
- Message Brokering with RabbitMQ
- Consumer Service
- Producer Service
- Load Testing with Locust
- Logging and Monitoring
- Error Handling and Reliability
- Deployment in Kubernetes
- Python and Flask