COEN 317 Distributed Systems
Project Proposal
Bronco - Career Alerts

Team Details:

Name	Student ID	Email ID
Lasya Malladi	W1650491	smalladi2@scu.edu
Venkata Sai Srikar Jilla	W1652687	vjilla@scu.edu
Chaitanya Kothapalli	W1653623	ckothapalli@scu.edu

Description:

University life is buzzing with events such as career fairs, workshops, and training sessions. These activities are great for students' learning and future careers. But there's a problem: students often miss out because they don't get the right information on time.

To solve this, we're launching "Bronco-Career Alerts." It's a new system that lets students know about events that match their interests. This system uses the publisher-subscriber model, where students can pick what kind of event news they want. To sum it up, "Bronco-Career Alerts" makes sure students don't miss out on great career events. It's like a bridge linking students to the opportunities they care about.

Motivation:

The dynamic life of universities is characterized by a myriad of events, both internal and external. Internally, students encounter events like career fairs,workshops, and boot camps, intensive training sessions that delve deep into specialized areas of study or professional development. Externally, there are career expos at other institutions, off-campus workshops, and collaborative training boot camps with industry experts. While these events immensely enrich a student's academic and professional journey, they also bring forth a significant challenge: efficient communication.

Universities usually send a lot of information through mass emails, but this method often results in students being overwhelmed with information. With the broad spectrum of events, a student looking forward to a career fair might find their inbox flooded with details about unrelated workshops, while another who wants to improve their skills at boot camps might not notice a great chance because they get so many career fair emails. This project introduces a publish-subscribe system designed with a student's specific preferences in mind. The crux of

this project is to move away from the conventional one-size-fits-all communication method to a more personalized and efficient publisher-subscriber model.

Literature Review

1) The research and design of Pub/Sub Communication Based on Subscription Aging

The research paper introduces a novel communication model for large-scale distributed systems, emphasizing the need for flexible, decentralized, and dynamic communication models. The proposed pub/sub architecture, with subscription aging and prioritized subscribers, enhances performance and reduces system load as subscribers increase. This approach mitigates single points of failure and improves data transmission efficiency, making it suitable for modern distributed systems.

2) PUBSUB: An efficient publish/subscribe system

The paper introduces PUBSUB, an efficient publish/subscribe system for event processing, outperforming BE-Tree and Siena. Its efficiency is due to smart data structures and matching algorithms, excelling in uniform tests and remaining competitive in Zipf tests. PUBSUB's adaptable architecture accommodates various data structures for different attribute types, making it a robust choice for event-driven systems, showcasing scalability and efficiency in managing diverse subscription loads and attribute dimensions.

3) A Survey of Publish/Subscribe Middleware Systems for Microservice Communication

This survey paper clearly explains the publish -subscribe architecture.It also highlights the differences between pub-sub and client-service architectures.It delved into different projects where publish/subscribe paradigm was used with Apache Kafka as a message broker in application of Internet of Things, streaming image data.It explores the vital role of the Pub/Sub messaging pattern in microservice communication, particularly within the context of real-time distributed applications. It highlights the suitability of microservices for large-scale applications and monolithic code base for smaller applications. The paper emphasizes event-based communication in microservices, where events are broadcast and subscribed to by other microservices, enabling eventual consistency.

4) Storm Pub-Sub: High Performance, Scalable Content Based Event Matching System Using Storm

This paper introduces a system for efficiently matching events and subscriptions with high throughput. It departs from traditional broker overlays and uses parallelization through bolts for various pub-sub operations. The system is compared to the Siena broker-based architecture, demonstrating its scalability and throughput.It highlights the system's distribution, high performance, and scalability, with an easy-to-deploy Storm topology. Parallelism is achieved by increasing the number of bolt and spout processes. The system is fault-tolerant and excels in delivering events with low latency.

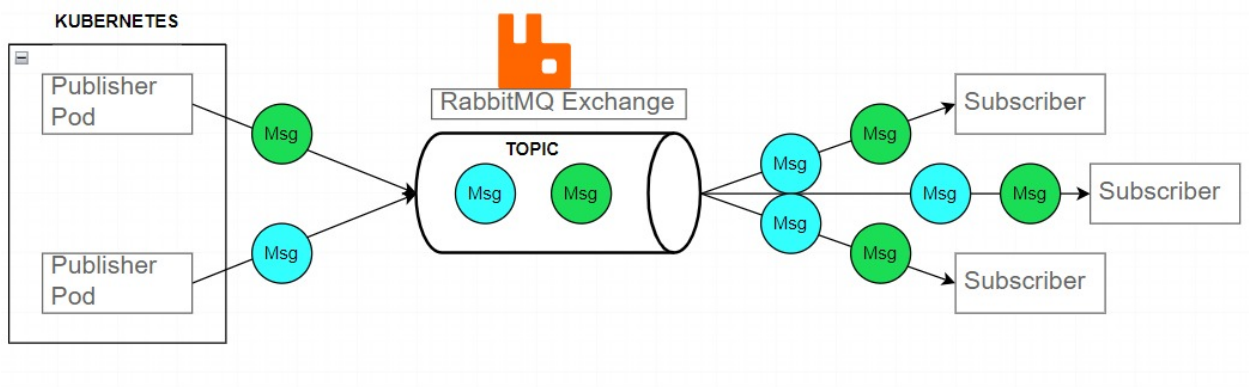
5) The Hidden Pub/Sub of Spotify

The article outlines the structural design of Spotify, a renowned music streaming service, which is built on the pub/sub communication framework. By detailing the system's workload, the paper offers insights for modeling pub/sub workloads in academic studies. Additionally, the study delves into the pub/sub traffic patterns within Spotify to identify prevailing trends.

6) Underlying Techniques for Large-Scale Distributed Computing Oriented Publish/Subscribe System

The paper introduces the design of a mobility support service tailored for distributed publish/subscribe systems. This enhanced support, similar to the one mentioned, provides features like buffering incoming messages and managing the transition of subscriptions between different publish/subscribe access points.

Architecture/methodology



Starting the project, we're going to set up a development environment. We'll use Docker to containerize the RabbitMQ service. At the same time, we'll use Kubernetes for container orchestration.

Moving on to the main design, we'll use RabbitMQ as Message Exchange for sending and receiving messages. With Topic Exchanges, we can sort messages based on the type of event they're about. Python-based publishers will be developed to send out notifications about events through RabbitMQ and match them with the right events. Python-based subscribers will be developed who will listen to events that resonate with the students selected interests.

For the system to work together, whenever a new event is added, the publisher will send out a notice about it. Students will be able to sign up and choose the event topics they want to hear about.

Lastly, when putting everything in place, we'll use Kubernetes. It'll hold the RabbitMQ and Publisher Pods. With Kubernetes Deployments and Services, we can easily manage and make sure everything works together.

List of Algorithms

1. Concurrency Control & Mutual Exclusion: Ensure that simultaneous operations do not conflict with each other, especially when multiple publishers try to send notifications at the same time. To ensure that only one operation happens at any given moment within these critical sections, mutex locks will be employed. This ensures that if one student is updating their preferences or signing up for an event, another student's simultaneous operation will be queued until the first operation completes.

2. Broadcast/Multicast: In the context of notifying students about events, broadcasting or multicasting becomes essential. Broadcast for Urgent Alerts: For extremely important or university-wide events, a broadcast mechanism can be used to send messages to every subscriber irrespective of their topic subscription.

3. Replication: Ensuring the system's availability and fault-tolerance is crucial for its consistent performance and reliability. With numerous students relying on the "Bronco-Career Alerts" system, any downtime can lead to missed opportunities. Replication ensures that even if one instance of the system fails, others can take over, ensuring continuous service.

Tech Stack: Python Flask, Docker, Kubernetes, External API-Greenhouse/Smart Recruiters Job Board API, Messaging Queue-RabbitMQ.

Milestones:

#	Ownership	Assigned Task	Target
1	Team	Literature Review-Going through IEE papers related to pub-sub architecture	10/13/2023
2	Team	Proposal Submission	10/15/2023
3	Lasya	Designing the Pub-Sub architecture	10/31/2023
4	Chaitanya	Setting up development environment	11/05/2023
5	Team	Integration with event systems	11/10/2023
6	Srikar	Deployment using Kubernetes	11/15/2023
7	Team	Testing and Documentation	11/20/2023
8	Team	Iteration-Implementing necessary changes	11/25/2023
9	Team	Presentation Preparation	11/26/2023
10	Team	Code Submission and Final report preparation	11/28/2023

References:

1. D. Yang, M. Lian, Z. Zhang and M. Li, "The research and design of Pub/Sub Communication Based on Subscription Aging," 2018 IEEE International Conference on Intelligence and Safety for Robotics (ISR), Shenyang, China, 2018, pp. 475-479, doi: 10.1109/IISR.2018.8535938.
2. T. B. Mishra and S. Sahni, "PUBSUB: An efficient publish/subscribe system," 2013 IEEE Symposium on Computers and Communications (ISCC), Split, Croatia, 2013, pp. 000606-000611, doi: 10.1109/ISCC.2013.6755014.
3. S. Kul and A. Sayar, "A Survey of Publish/Subscribe Middleware Systems for Microservice Communication," 2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey, 2021, pp. 781-785, doi: 10.1109/ISMSIT52890.2021.9604746.
4. M. A. Shah and D. B. Kulkarni, "Storm Pub-Sub: High Performance, Scalable Content Based Event Matching System Using Storm," 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, Hyderabad, India, 2015, pp. 585-590, doi: 10.1109/IPDPSW.2015.95.
5. Setty, Vinay & Kreitz, Gunnar & Vitenberg, Roman & van Steen, Maarten & Urdaneta, Guido & Gimaker, Staffan. (2013). The hidden pub/sub of spotify. 231-240. 10.1145/2488222.2488273.
6. J. G. Ma, T. Huang, and J. L. Wang, ”Underlying Techniques for Large-Scale Distributed Computing Oriented Publish/Subscribe Sys- tem”, Journal of Software, vol. 17, pp. 134-147, Jan. 2006.