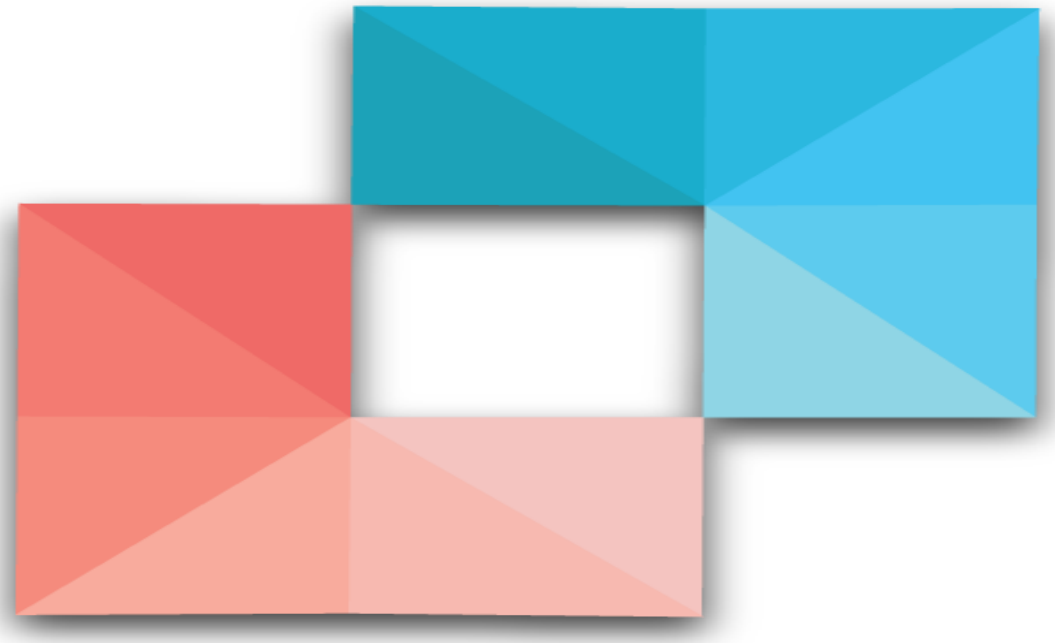
Final Report: Science Gateway Architecture

Fall 2017

FA17-BL-CSCI-B649-35071



Kumar Satyam

Email-id: ksatyam@indiana.edu

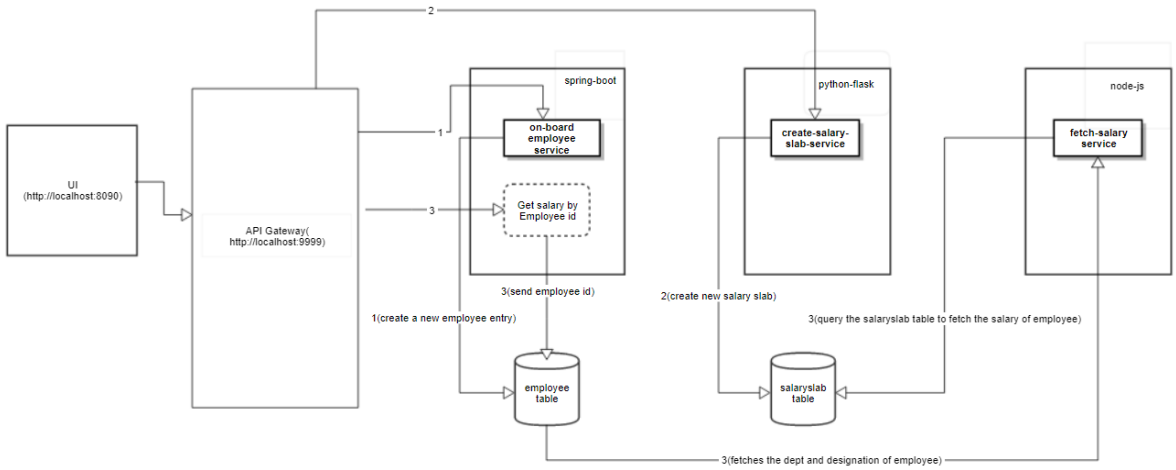
Table of Contents:

Assignment 1:

Problem statement:

Develop a set of micro services including few components in 3 different programming languages, expose them through a API server. Develop a UI framework to consume these API's and facilitate user interactions.

Solution:



I developed a microservice architecture using 3 services and an API gateway. In this project we are assuming that an employee working in a department will have same salary as the other employee working in the same department with same designation. So, we can fetch the salary of the employee using the mapping of department and employee. This architecture had 3 services 'onboarding an employee service', 'department-salary service' and 'fetching salary of an employee service'. It had UI interface and API gateway as well.

Contribution:

Wiki link: https://github.com/airavata-courses/satyamsah/wiki/Assignment1

Evaluation

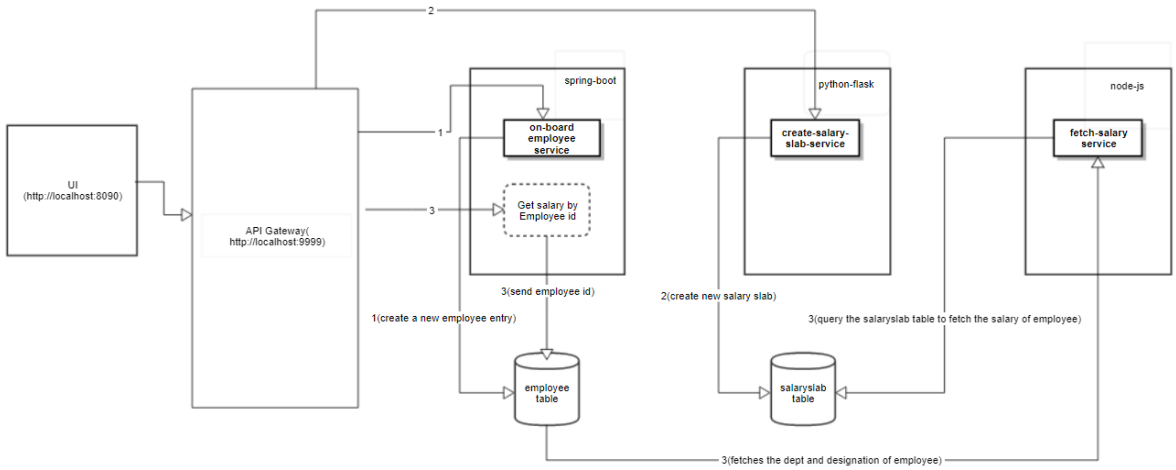
The whole set was tested by me and associate instructors locally.

Assignment 2:

Problem statement:

Dockerized all the microservices developed in assignment 1. Use RabbitMQ for communication between the microservice. Use continuous deployment to deploy whole setup using Jenkins

Solution:



Firstly, dockerized all the microservices. Installed RMQ. Installed Jenkins and Rabbit MQ in different machines

Deployed whole setup of microservices using Jenkins which download all the files from GitHub and deploy it into a destination server.

Contribution:

Wiki link : https://github.com/airavata-courses/satyamsah/wiki/Assignment2

Evaluation

The whole set was tested by me and associate instructors.

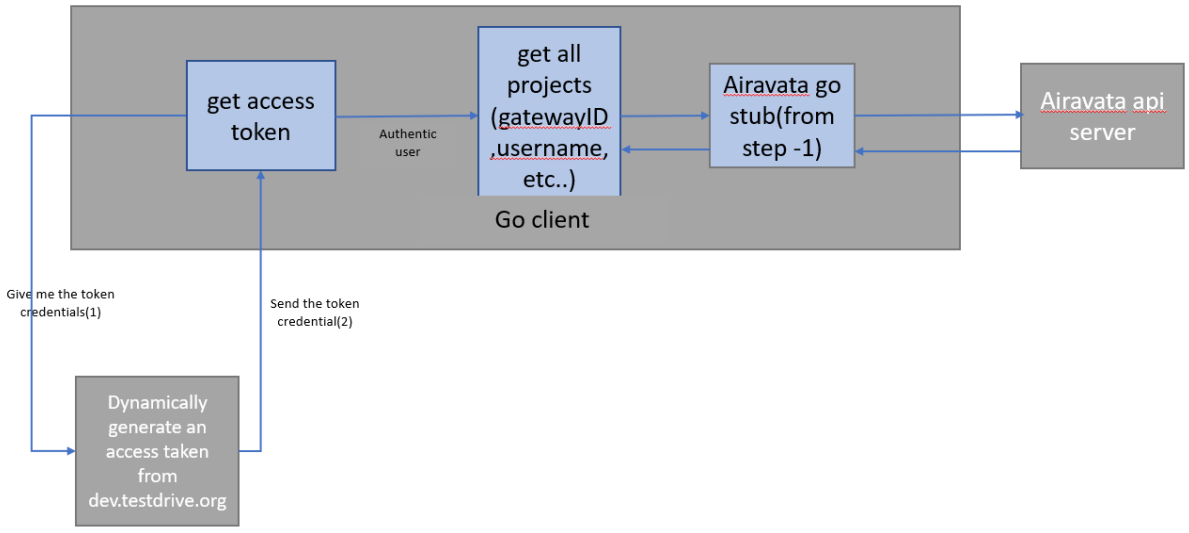
Assignment 3:

Problem statement:

To develop a new 'golang' client for airavata. It will give new capability inside airavata to serve new users who want to interact with airavata using an application written in 'Go'.

Solution:

The main task is to replicate the task of Marcus: https://github.com/machristie/airavata-python3-client where he developed a python client. This will help any new application which was developed in the Go language to be easily integrated with Airavata.



Communication: https://issues.apache.org/jira/browse/AIRAVATA-2522

Pull Request: https://github.com/apache/airavata/pull/119

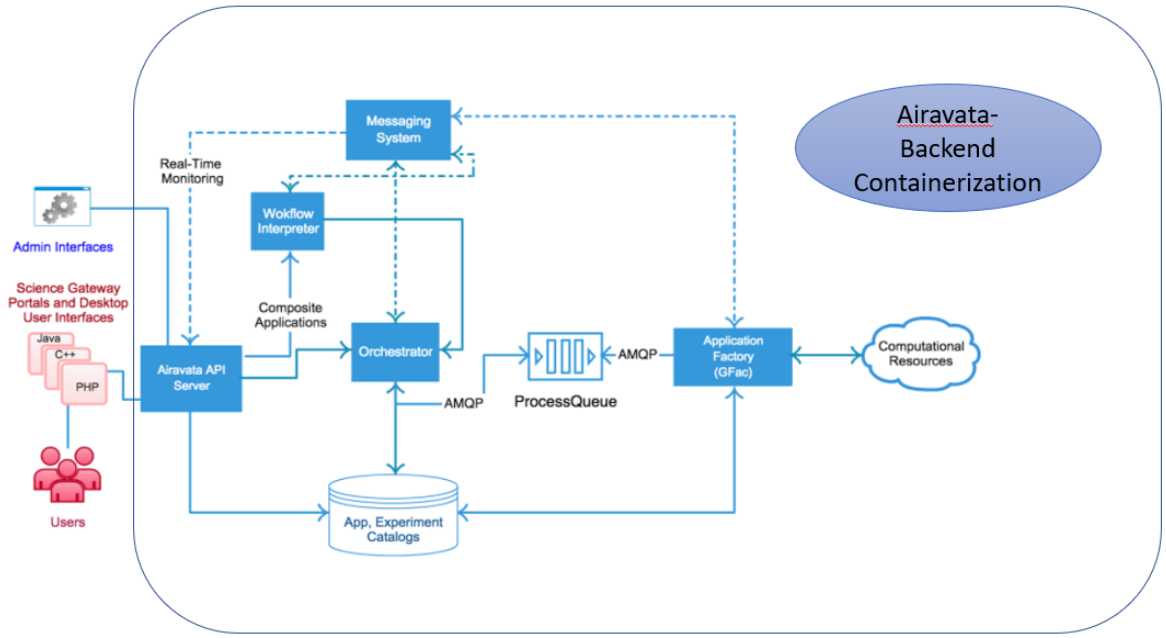
Contribution:https://github.com/airavata-courses/satyamsah/wiki/assignment3

Evaluation: The whole setup was tested associate instructors locally

Assignment 4:

Problem statement:

Understanding Kubernetes for Orchestrate Backend of Airavata using Kubernetes.



Solution method: First approach was to understand Kubernetes. So, first I understood Kubernetes concept. Deployed management components of Kubernetes in one VM. The set was deployed using minikube only in 1 node cluster. Dockerized sample apps and the deployed it through Kubernetes

Evaluation and Recommendations:

The evaluation feedback was to develop the whole setup in clustered mode with many nodes.

Jira story: https://issues.apache.org/jira/browse/AIRAVATA-2562

Airavata developer list discussions:

Since Jeffry was working on dependent sub task to the dockerizing the backend of Airavata , I consulted with him as well as Professor Suresh Marru.

List all your commits, pull request:

https://github.com/apache/airavata/pull/128

Assignment 5:

Problem statement:

Continuation of Problem statement 4: To Orchestrate Backend of Airavata using Kubernetes. The problem still exists that all the Dockerized components were running in same hosts.

Solution:

To remove this coupling, we used 4 jetstream machines with 1 master and 3 slave nodes. I deployed the components like RabbitMQ as an external compnents. The setup was using its own embedded Zookeeper and DerbyDB with externally connected RabbitMQ.

Evaluation and Recommendations:

The solution was evaluted by the Assistant instructors and by the peer reviewers. This was not fault tolerant and there was only 1 instance of airavata running. There was scope of using external Mariadb and external Zookeeper.

Specific contribution:

Wiki link : https://github.com/airavata-courses/satyamsah/wiki/assignment5(orchestration-of-airavata-backend-using-kubernetes)

Jira story:

https://issues.apache.org/jira/browse/AIRAVATA-2562 (Links to an external site.)

Airavata developer list discussions:

Since Jeffry was working on dependent sub task related to the same issue, I consulted with him, Professor Suresh Marru

pull requests:

https://github.com/apache/airavata/pull/132 (Links to an external site.)Links to an external site.

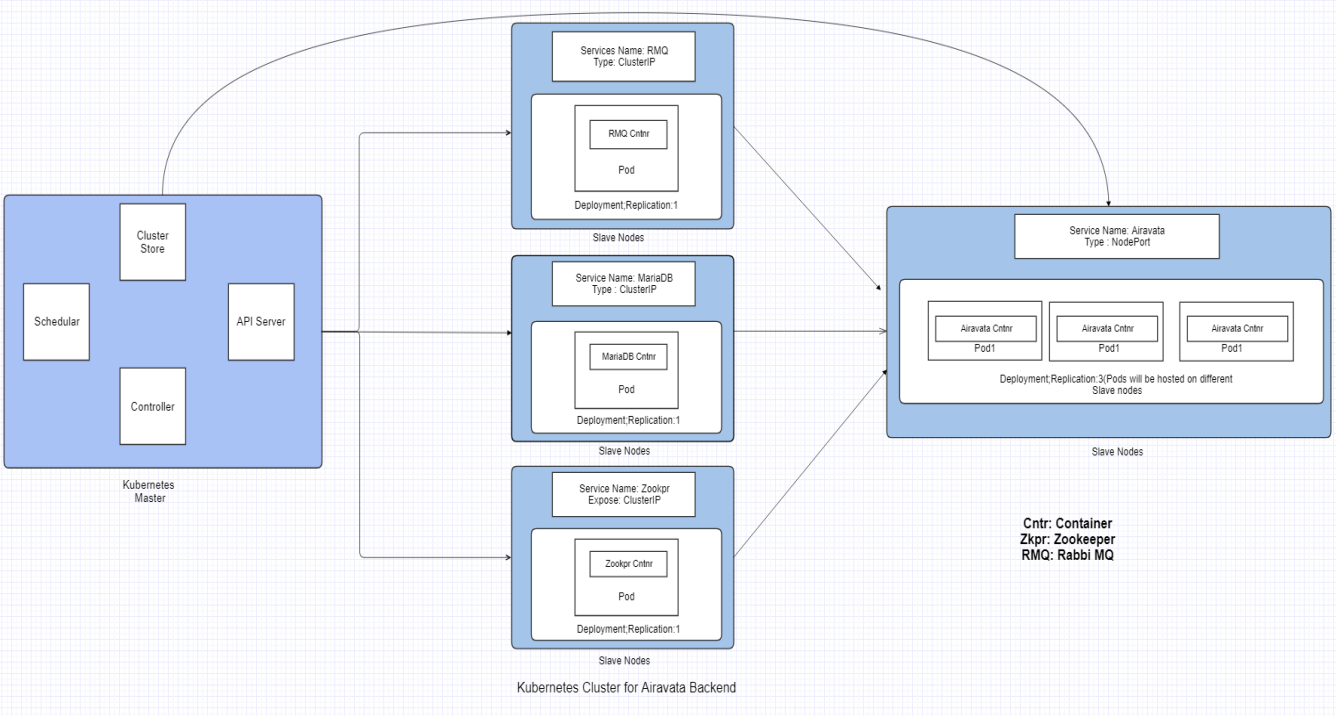
Assignment 6:

Problem statement:

Extension of assignment 5. Orchestration of Airavata Components in cluster mode.

Solution method:

Here I implemented the solution in full cluster mode using the Deployment and Services module. Here deployment helped in making sure that fault tolerance is maintained using replica sets and easy roll-forward and rollback is possible. Used external mariadb and external zookeeper to be acting as a dependency of airavata backend. Below is the diagram to explain this



Evaluation:

Evaluation was by done by testing a sample client which gives API version and get experiment.

Your specific contributions:

Wiki link : https://github.com/airavata-courses/satyamsah/wiki/assignment6

Jira story:

https://issues.apache.org/jira/browse/AIRAVATA-2562

Airavata developer list discussions:

I consulted with Jeffry and changed the Dockerfile as per my requirement

pull requests:

https://github.com/apache/airavata/pull/142

Assignment 7:

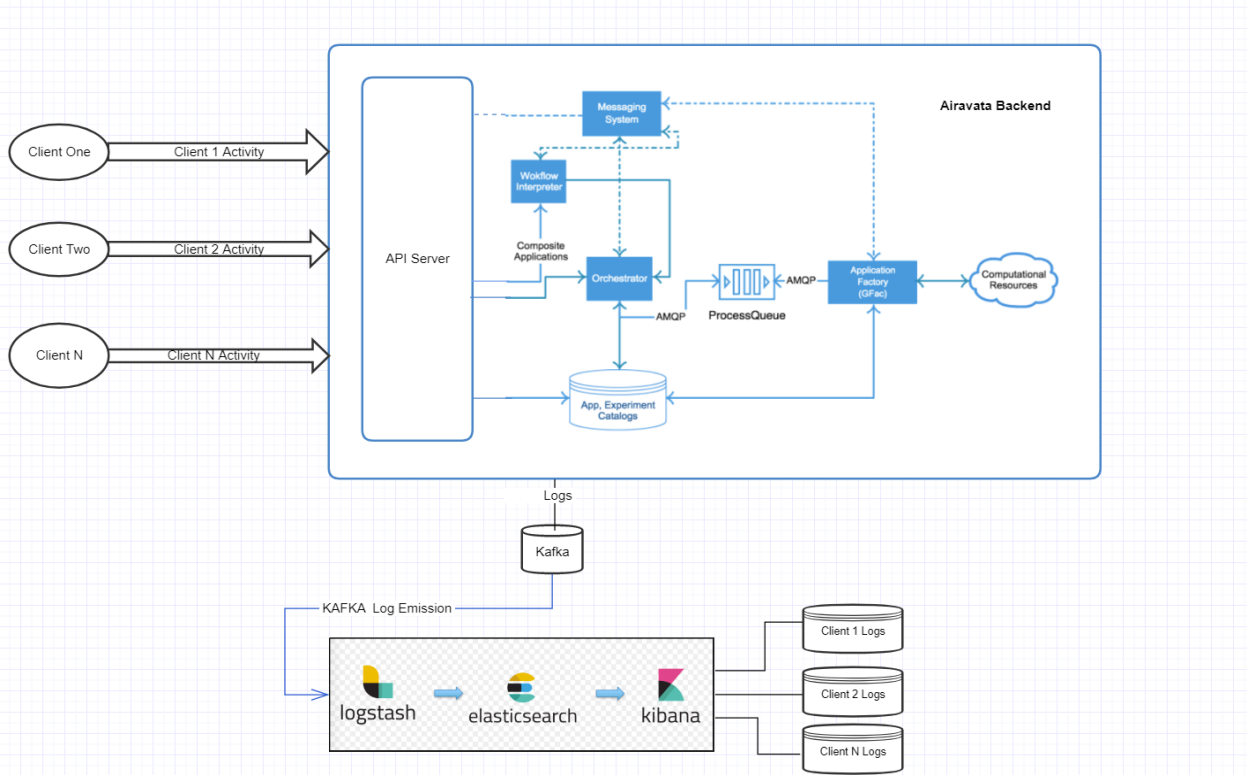
Problem statement:

Fetching client gateway level logs for apache airavata.

Airavata is being used by many scientists from many university. Each university uses its own client gateway to connect to API-gateway of Airavata server. So, my task is to analyze/fetch logs related to a specific client.

Solution method:

As it is a topic of new research, the first approach is to find class level logs. By class level logs, we mean, to find the modules inside Airavata that are emitting the logs and then to reverse propagate to the source of the logs. So, first I deployed ELK stack and integrated it with Kafka. So, first test of integrating Airavata with elk stack via Kafka is successful.



Contributions:

Wiki link: https://github.com/airavata-courses/satyamsah/wiki/assignment7

Jira :

https://issues.apache.org/jira/browse/AIRAVATA-2604

Airavata developer list discussions:

I was working with Marcus and Jeffry to understand the problem

Pull requests:

https://github.com/apache/airavata/pull/151