PLH419 Distributed Systems

Project Presentation

2023-2024

Authors:

- Chalvatzis Kyriakos 2018030043
- Georgakopoulos Panos 2015030049
- Konofaos George 2018030175
- Tsiampokalos Stavros 2018030164

https://github.com/kyri56xcaesar/Katanemimena.git

Tools/Frameworks Used. Python!

- Flask for each API service.
- Flask-sqlalchemy for the Auth service DB
- Requests for each HTTP client side use
- Pyjwt “Json Web Token”, cryptography, pycryptodome for authentication
- Kubernetes python client api library
- etcd3 python etcd client api library
- tenacity for concurrency control

System Architecture and Logic

- **Client script:** Used by the user to perform all system actions. Sends request to UI and AUTH.
- **Auth Service:** Authenticates users, holds user credentials in a DB, generates tokens.
- **UI Service:** Handles traffic from client script, forwards authenticated user job requests to the Manager,provides admin interface.
- **Manager Service:** Creating a Kubernetes Job and coordinating MapReduce Job. Performs preprocess, shuffling operation and spawns workers.
- **Persistent Volume:** Shared File system for Manager and workers
- **etcd DDS:** Keep state information of the Manager. Used to provide unique Job ids and fault tolerance of the manager

Client

Provides functionality for the user to interact with our services.

The script works as a command-line tool to submit jobs but also as an interactive shell for the admin panel.

In order for the user to use the script for any option he has to login to acquire a token in JWT form, that he needs to provide to UI_service.

Options:

-Datafiles: View the name of the datafiles available in cluster

-Jobs [filename, mapper, reducer]: Submit Job to MapReduce service

-Jobs : View Jobs status or get Job result

-admin: Access the admin interface

Auth_service

Auth_service is responsible for authenticating users and providing them with a valid token. It has access to the database and also the endpoints to update it as an admin.

Functionalities:

-Authenticates users through login with password

-Generates token for users based on their username with auth_service's **private key**.

-Manages the Database of users.

UI_service

UI_service’s endpoints require the token from the user’s request in order to verify him.

Endpoints:

-Send_jobs:Accepts 3(data,mapper,reducer) files as arguments and creates a job request to a manager,forwarding the files provided.

-Admin_panel:Handles the options an admin has.

-View_jobs:Request Jobs status or result from manager.

Other functionalities:

Verify_user:Verifies that the token the user provides is correct based on Authservice public key.

Get_manager: Do a basic load balance to choose on which manager the job will be assigned to.

Manager

Manager as a service receives requests from the UI to:

- submit a job
- view a job
- get job results

As a master/coordinator he performs the following actions:

- Splits the requested datafile into chunks. Chunk size is defined to 1MiB. For I/O he uses his Persistent Volume which is also visible to the workers.
- Estimates the number of workers needed according to the chunk plurality
- Spawns Mapper Workers by setting and applying a Kubernetes Job.
- Waits for Map job to be completed, then performs the Shuffle operation to identify unique keys and “collect” them
- Then similarly estimated the amount of reducers needed, spawns Reducer Pods by applying the Reduce job.

Output is written to the PV.

Distributed Data Store

To implement fault tolerance to the system in the case of a node PlayMaster failure we implemented a distributed data storage service namely ETCD.

In order for a Master node to be able to continue managing the jobs assigned to him by the UI service in case of failure they were implemented as a stateful set. When kubernetes restarts a manager pod the information saved in ETCD is used to recover the state of the pod and reschedule any unfinished jobs. ETCD is a distributed and reliable key-value store for critical data of a distributed system.

Keys: Manager ID + Job ID + Value Index

Values:

- **Filename**
- **Map Function**
- **Reduce Function**
- **Number of Mappers**
- **Number of Reducers**
- **State**

States:

- **Mapping**
- **Shuffling**
- **Reducing**
- **Completed**

Master node replication and handling of concurrent requests

By adding the name of the pod to the key we ensure mutual exclusivity between replicas as far as access to etcd key-values is concerned but how do we handle concurrent requests?

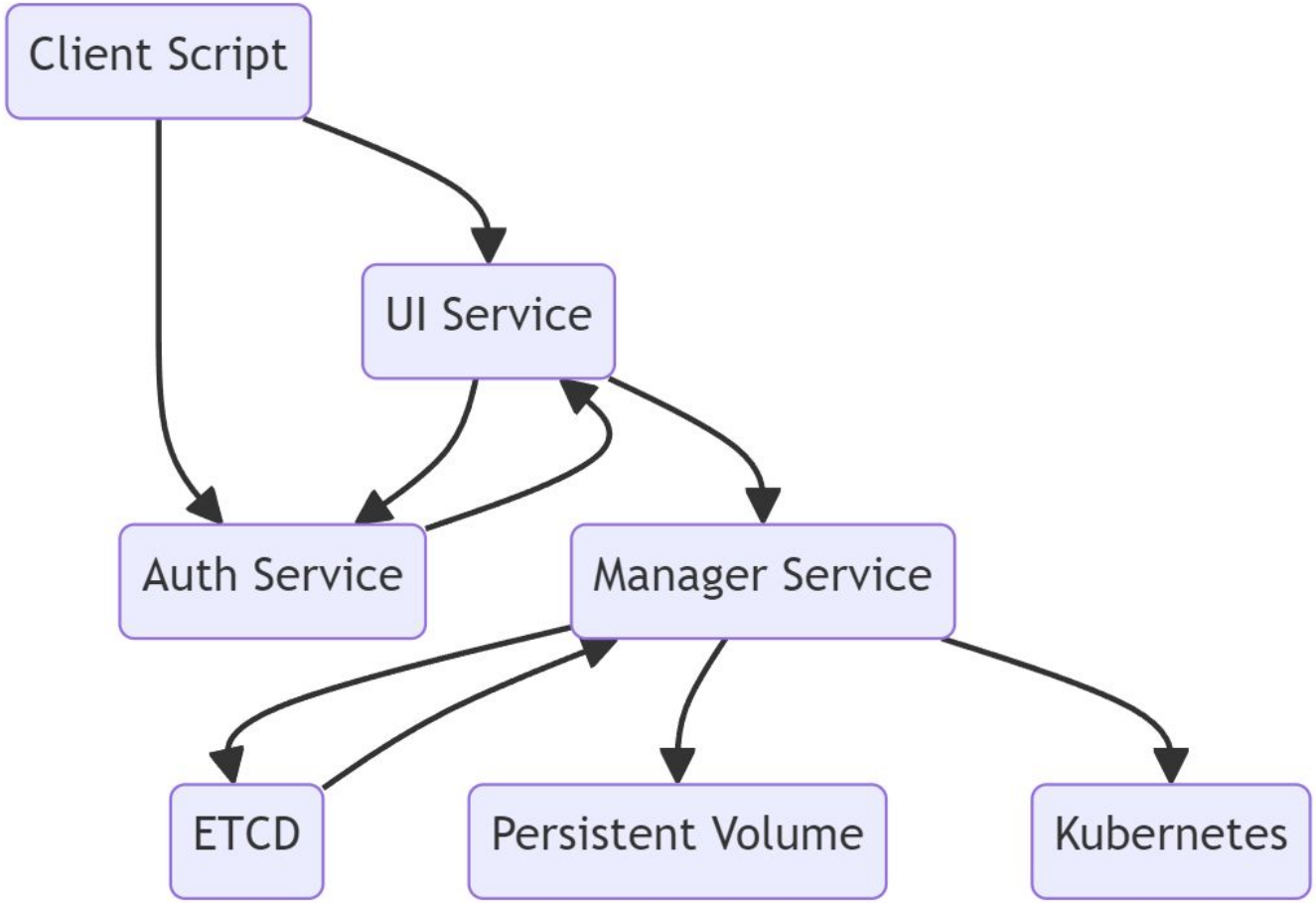
More precisely how do we ensure that we assign unique Job IDS for each job?

This is where etcd comes in to ensure mutual exclusivity by providing locks for keys.

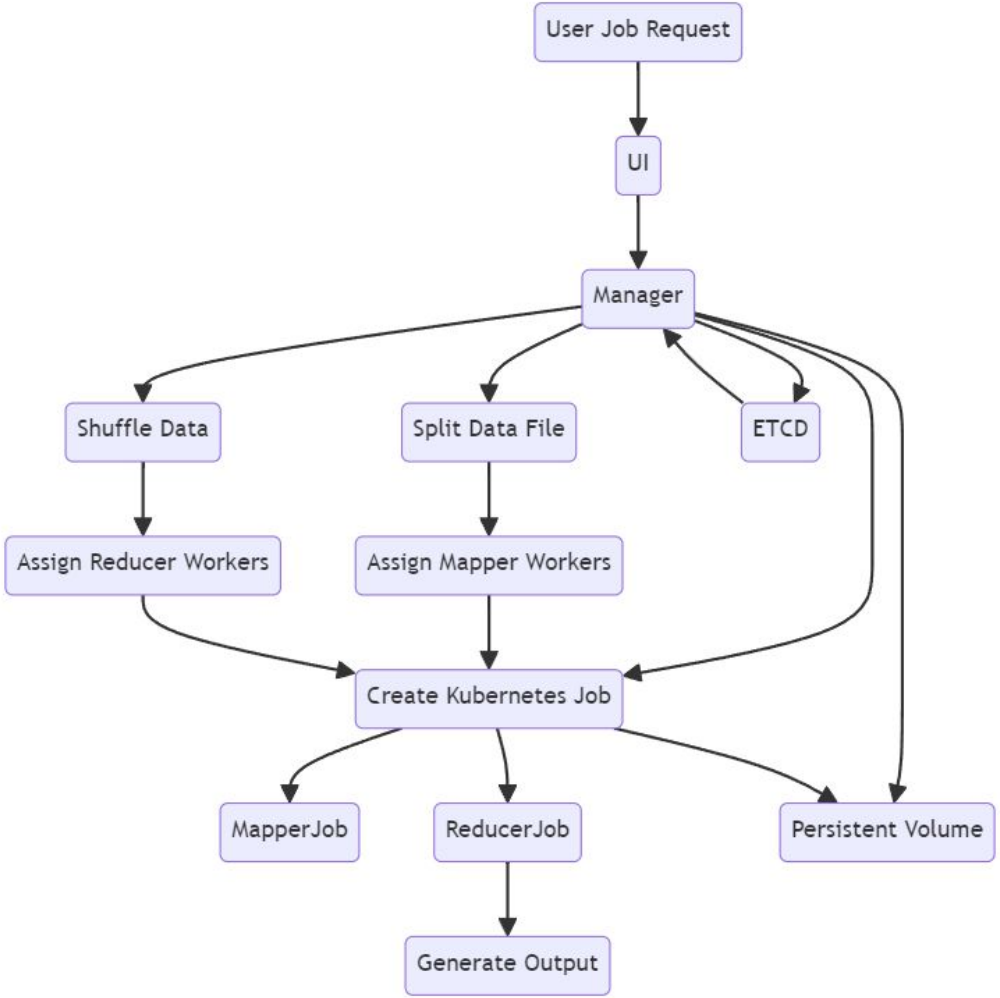
Each manager stores a key value pair where the key is the manager pod name and the value is the number of jobs created.

When a new request arrives to the endpoint we use the etcd api to lock that key increment the value by 1 and update it.This value along with the pod name serves as a unique Job ID which is then used as a key for etcd to save the state of the pod and all the required information in order to recover that state.

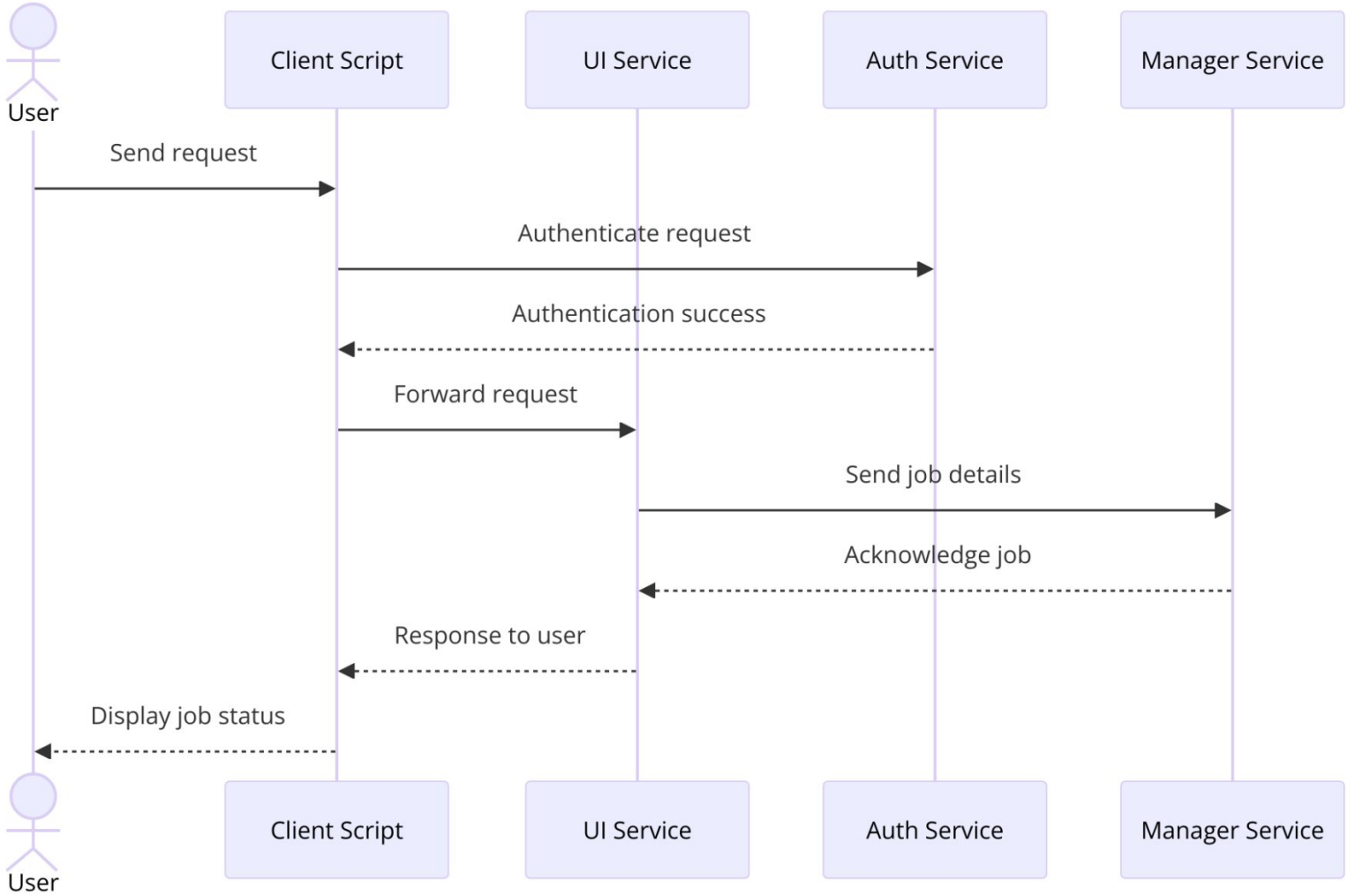
System Components



Manager or Master workflow



Sequence Diagram



Demo…


TERMINAL
python
vsamming...
vsamming...
vsamming...
vsamming...
vsamming...
vsamming...
vsamming...
vsamming...

Sources:

- Kubernetes documentation

- etcd documentation

- ChatGPT, mostly to translate k8s manifests to python kubernetes API client code…

- Lipsum.com for generating random text…

Thanks!