

**FRAUD DETECTION ON BANK PAYMENTS**

Experiment implemented on top of HPE Ezmeral Unified analytics platform.

RAY

Preface:

Ray is a high-performance distributed execution framework targeted at large-scale machine learning and reinforcement learning applications. It achieves scalability and fault tolerance by abstracting the control state of the system in a global control store and keeping all other components stateless. It uses a shared-memory distributed object store to efficiently handle large data through shared memory, and it uses a bottom-up hierarchical scheduling architecture to achieve low-latency and high-throughput scheduling. It uses a lightweight API based on dynamic task graphs and actors to express a wide range of applications in a flexible manner.

In addition to the above details Ray is an open-source unified compute framework that makes it easy to scale AI and Python workloads from reinforcement learning to deep learning to tuning, and model serving.

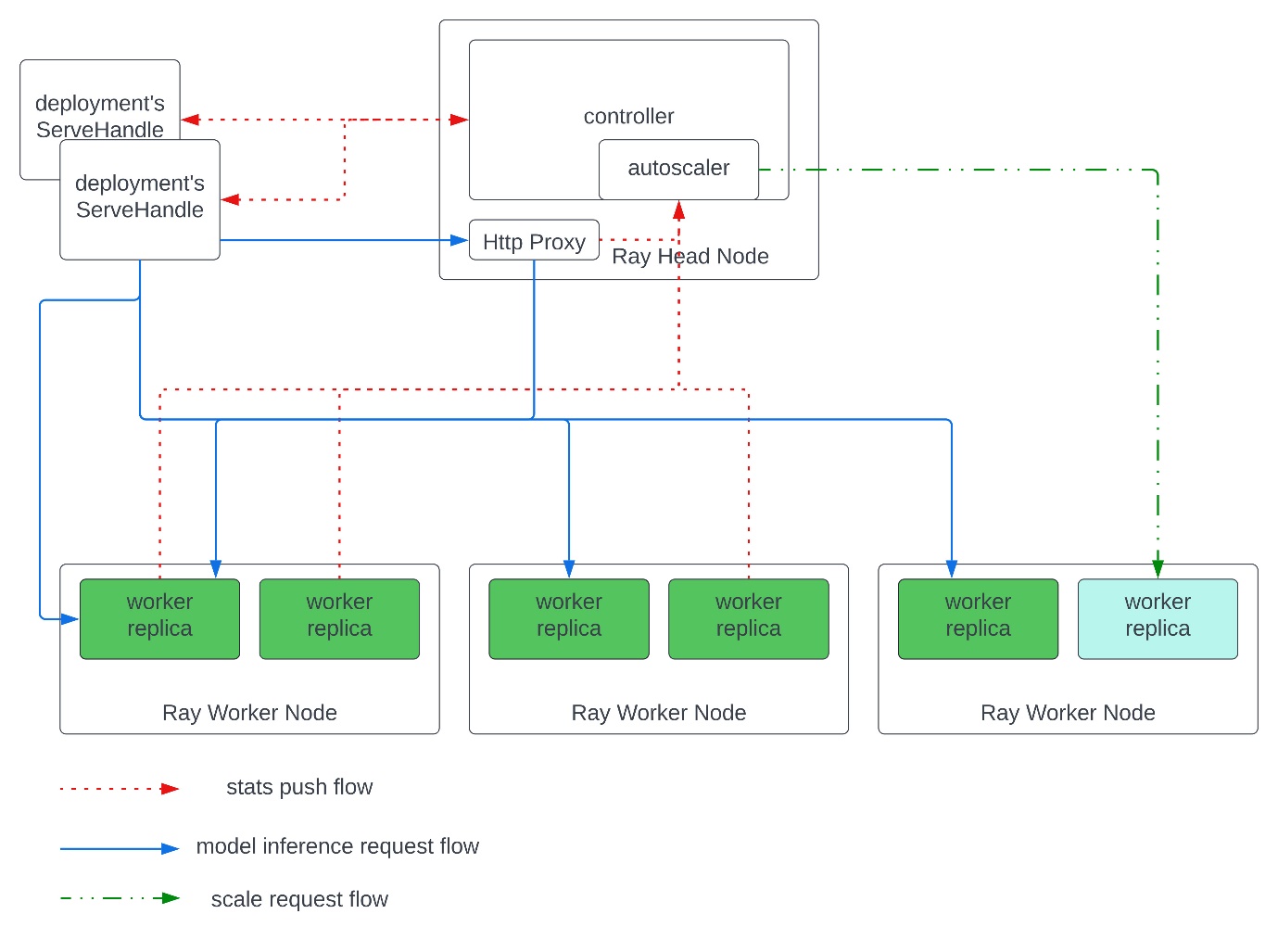


Fig1: Ray semantic diagram how it works

Ray cluster details:

The RAY is deployed in below namespace:

* Kuberay

Types of workloads and resource allocation we have configured:

Each task in Kuberay consumes such resources:

The EzAF recommending the infrastructure (not the executors) here we have got such resources for all components of infrastructure (components, services, operators), (so this is a summarization of all infrastructure setup what we configured

default level).

**CPU configured cluster details**

The current application cluster RAY distributed framework been configured with the following resources for head, operator, autoscaller and worker node, pv/c.

The ray-head pod we are default configuring following resources:

* **Ray head node**

resources:

limits:

* + - cpu: '2'
    - memory: 8G

requests:

* + - cpu: '1'
    - memory: 1G
* **Ray autoscaler:**

resources:

* + - limits:
    - cpu: 500m

memory: 512Mi

requests:

* + - cpu: 500m
    - memory: 512Mi
* **Ray operator-nightly:**

resources:

* + - limits:
    - cpu: 100m

memory: 512Mi

requests:

* + - cpu: 100m
    - memory: 512Mi
* **Ray worker pod:**

resources:

**limits**:

* + - cpu: '3'
    - memory: 8G

**requests**:

* + - cpu: '1'
    - memory: 1G

Experiment details:

In this kernel experiment I have leveraged the flowing services provided by Ezmeral Unified analytics platform.

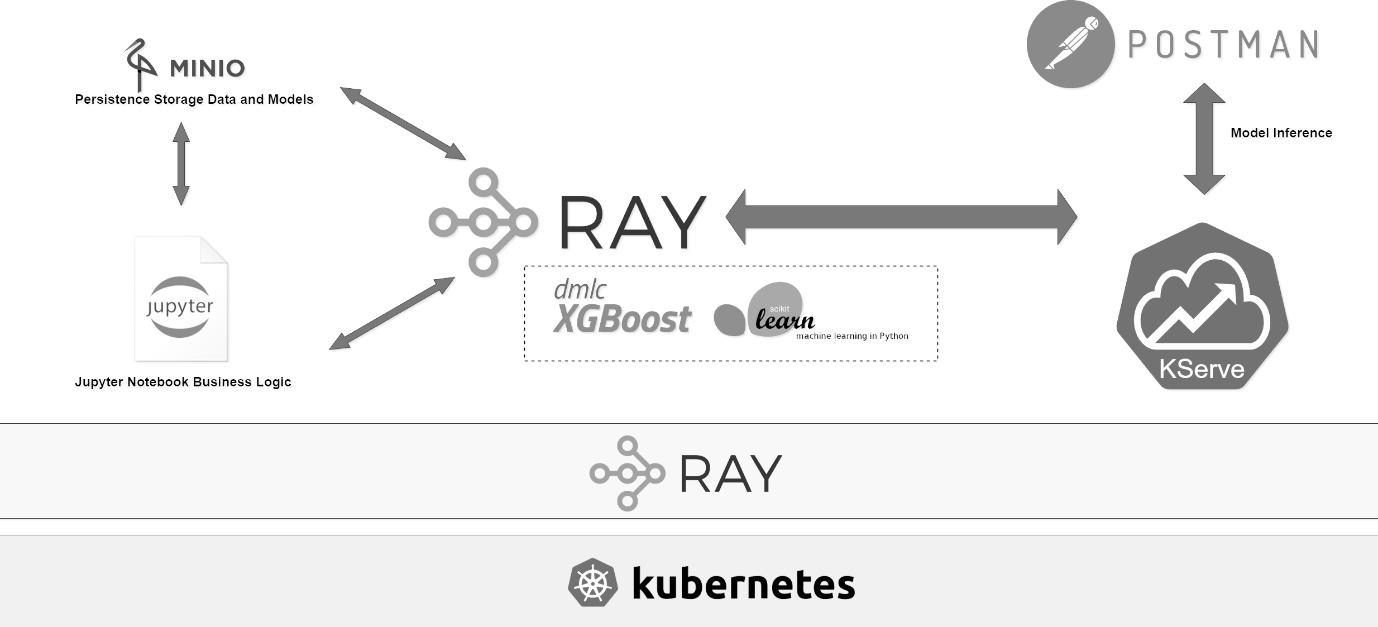
Services:

* Kubeflow – Kserve
* Minio (object storage)
* Ray Tuning
* Ray Training
* Xgboost Model
* Liner Regression Model
* We can add as much as model we need but based on Ray cluster version, we must need to follow guidelines.

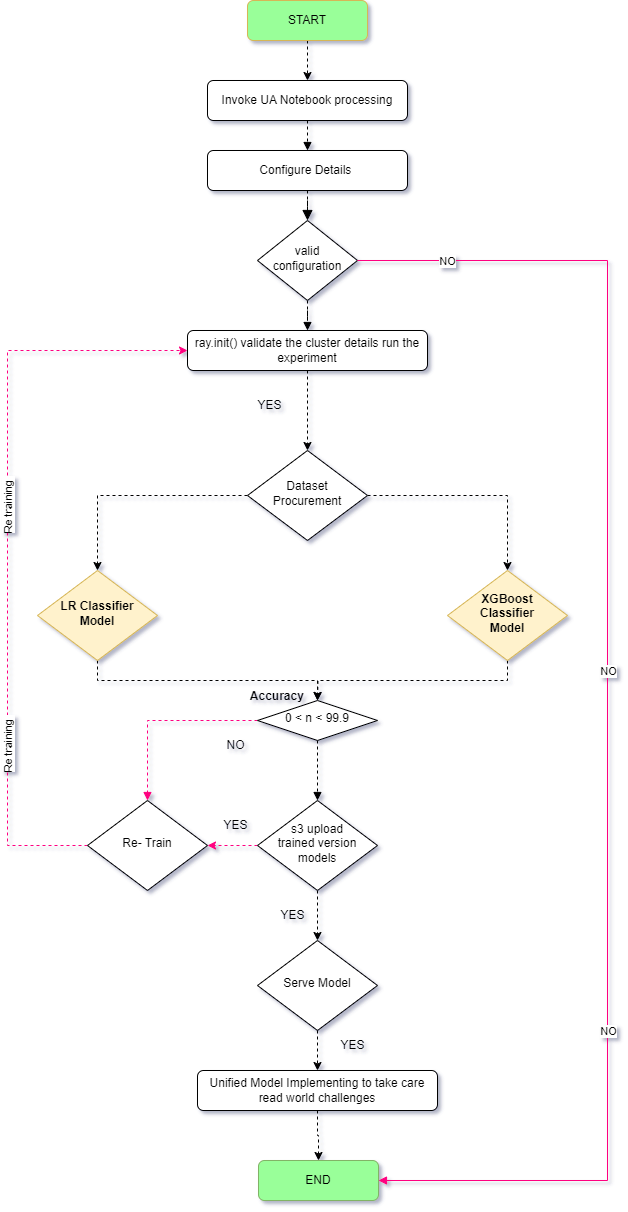
Experiment techniques:

* SMOTE: Synthetic Minority Over-sampling Technique, <https://jair.org/index.php/jair/article/view/10302>
* Rest of details are added in reference area please have a look

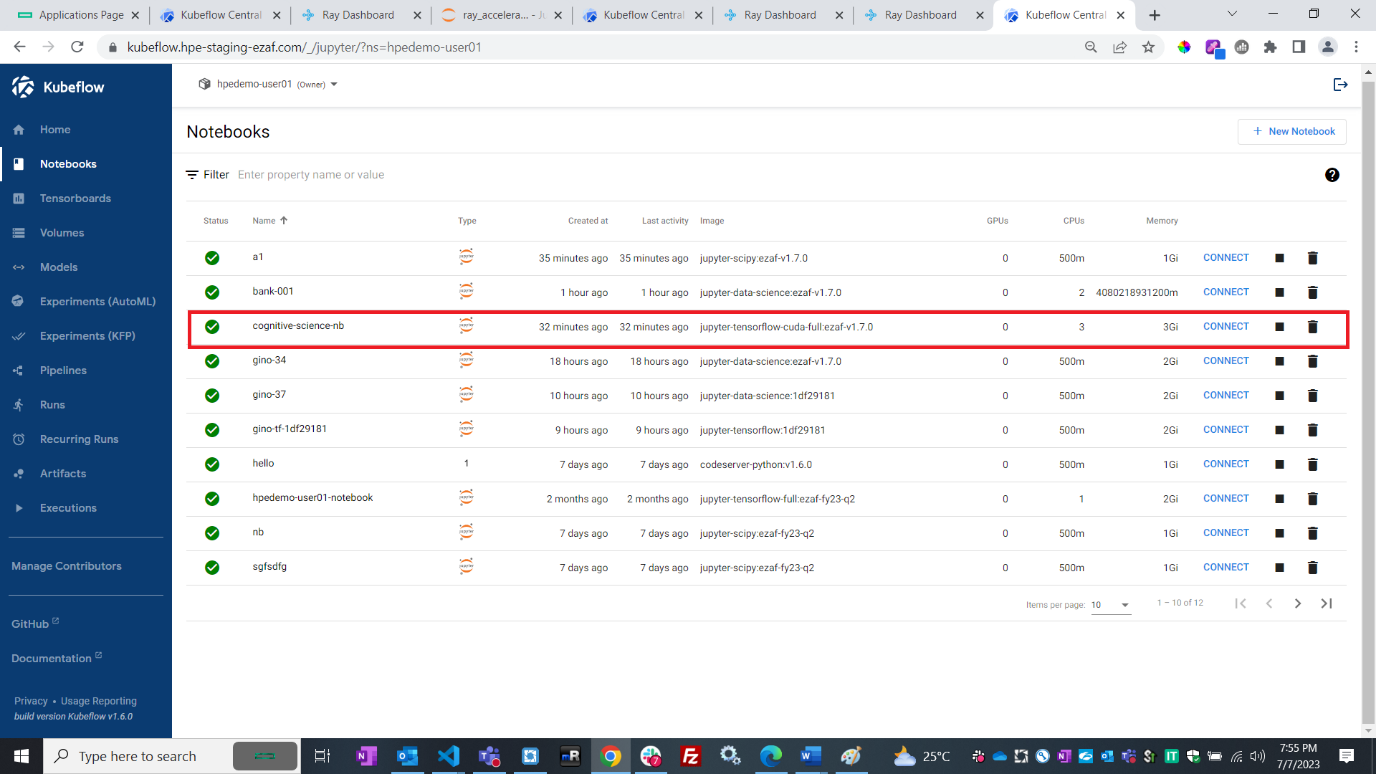
Top-level view this is how the experiment looks like:



The complete experiment flow diagram

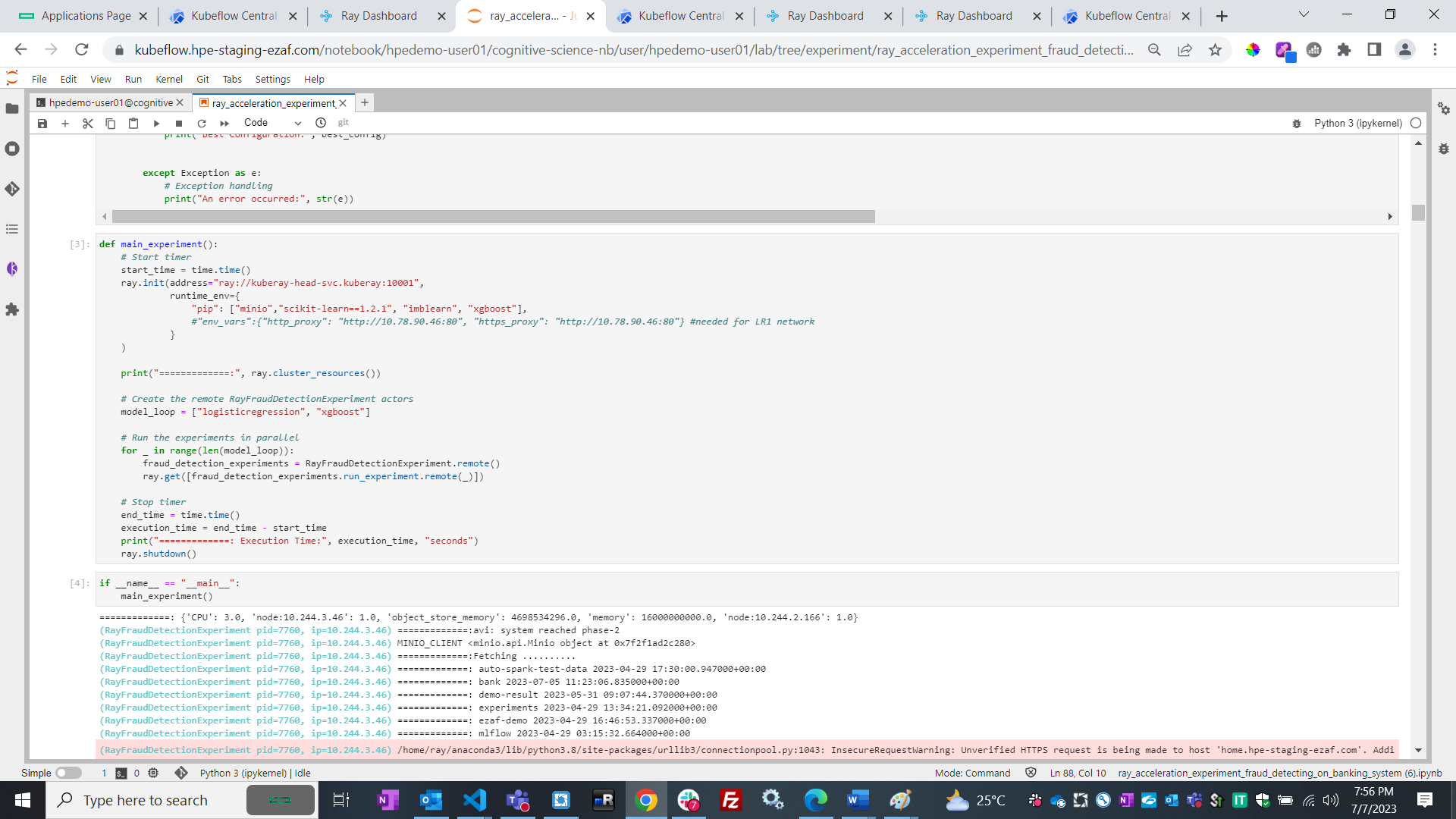


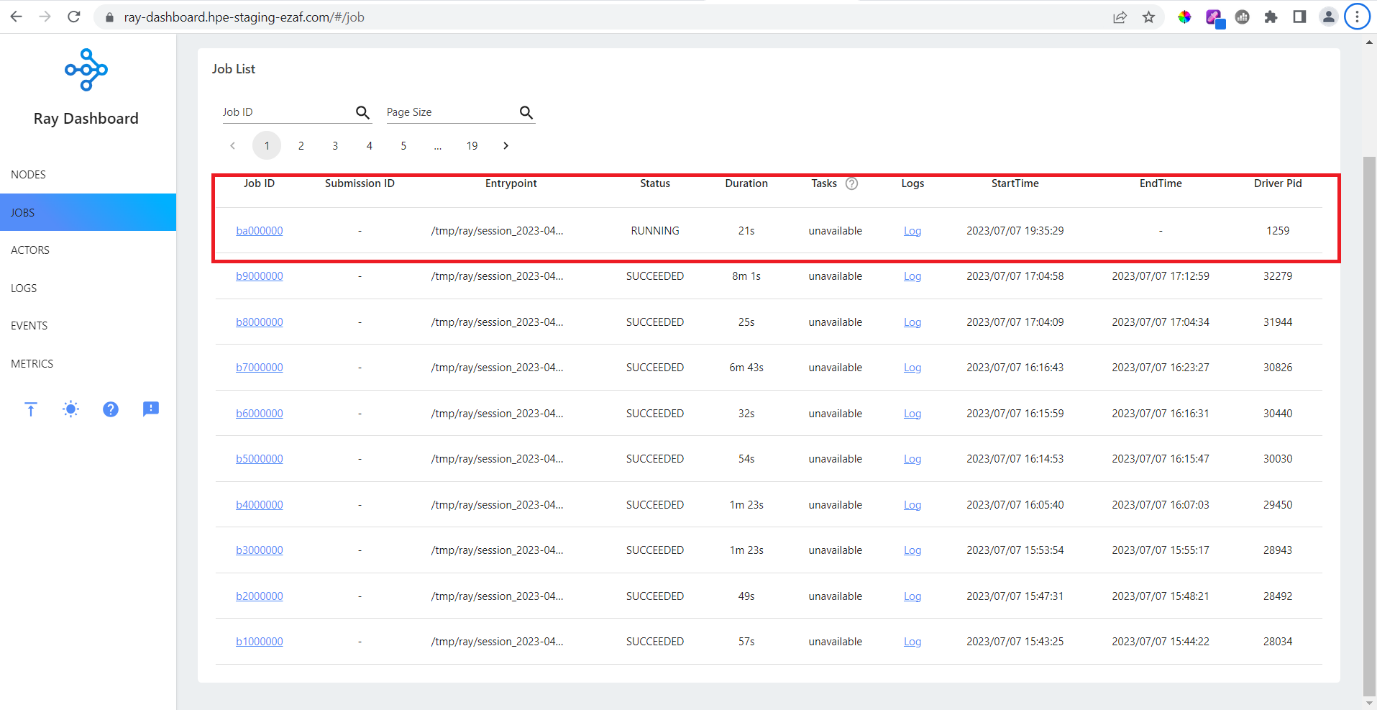
**Notebook**: cognitive-science-nb

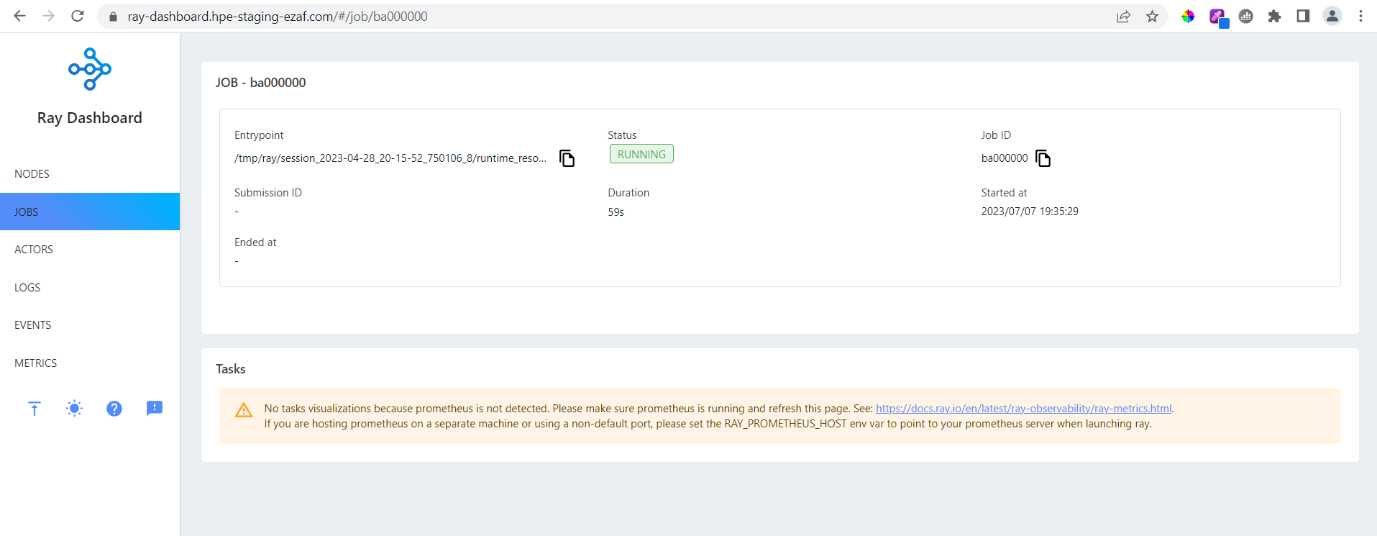


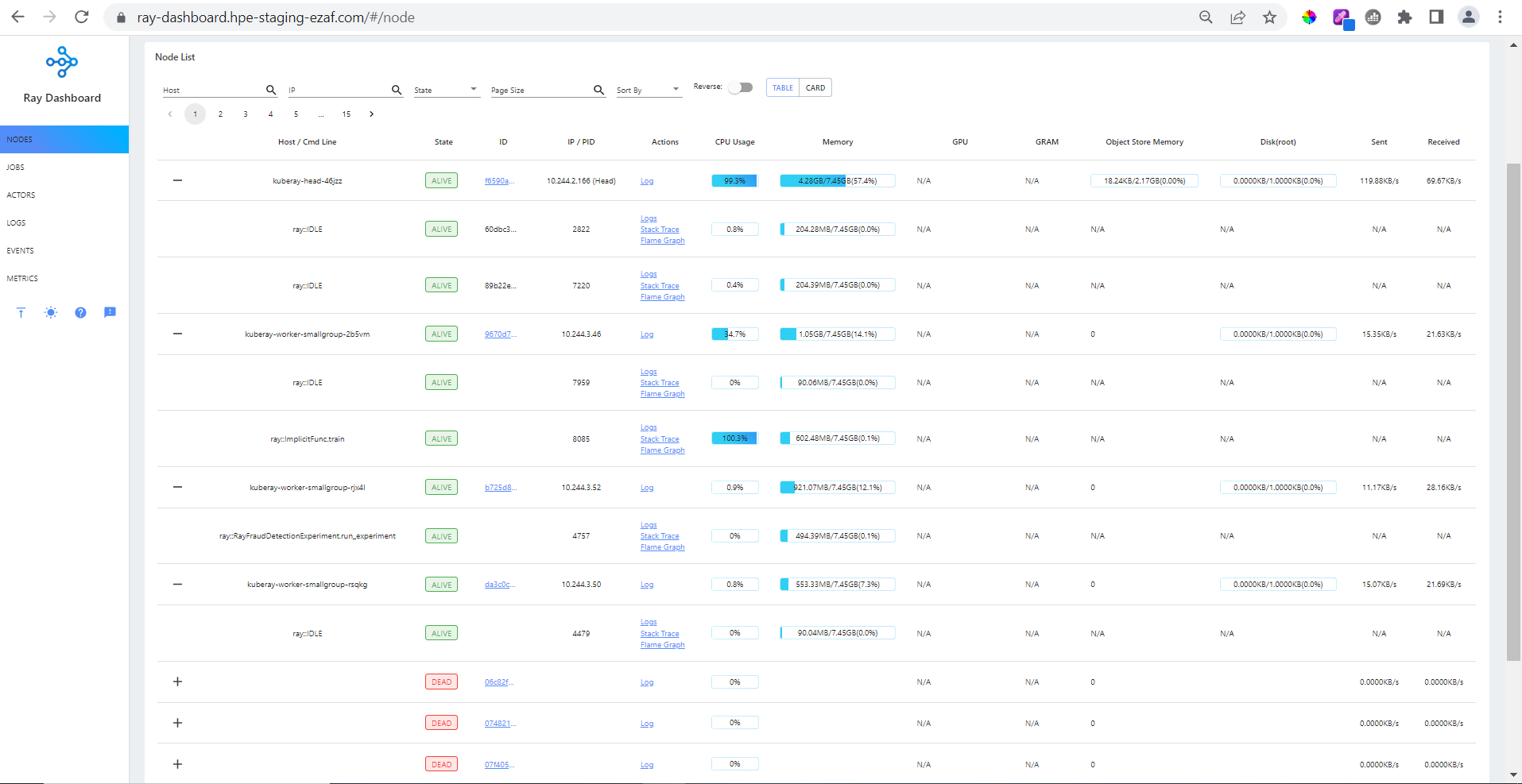
Notebook In-house developed RAY AI and ML codebase snip

Here we are invoking ray.init(“ray://kuberay-head-svc.kuberay:10001")

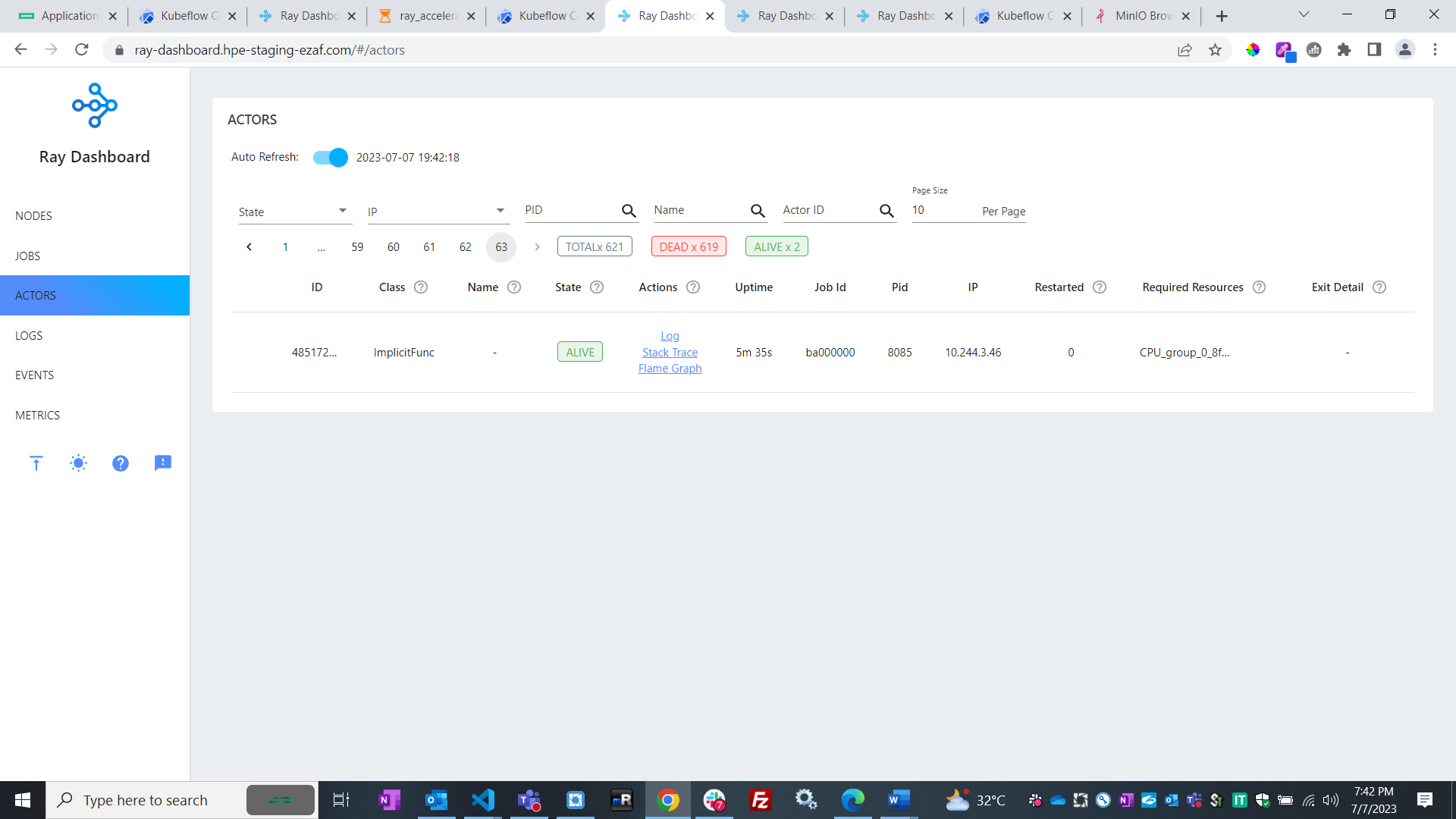


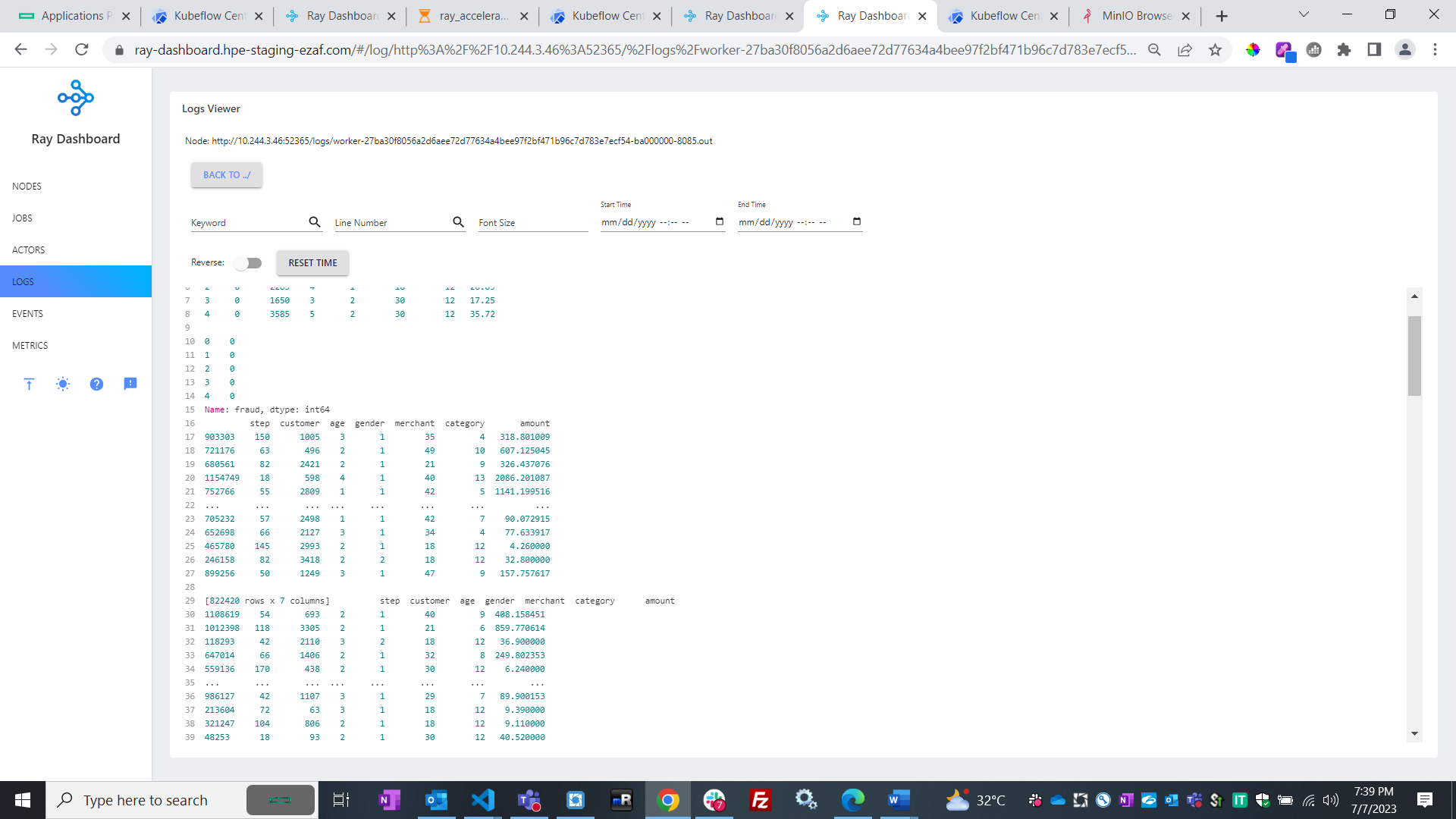




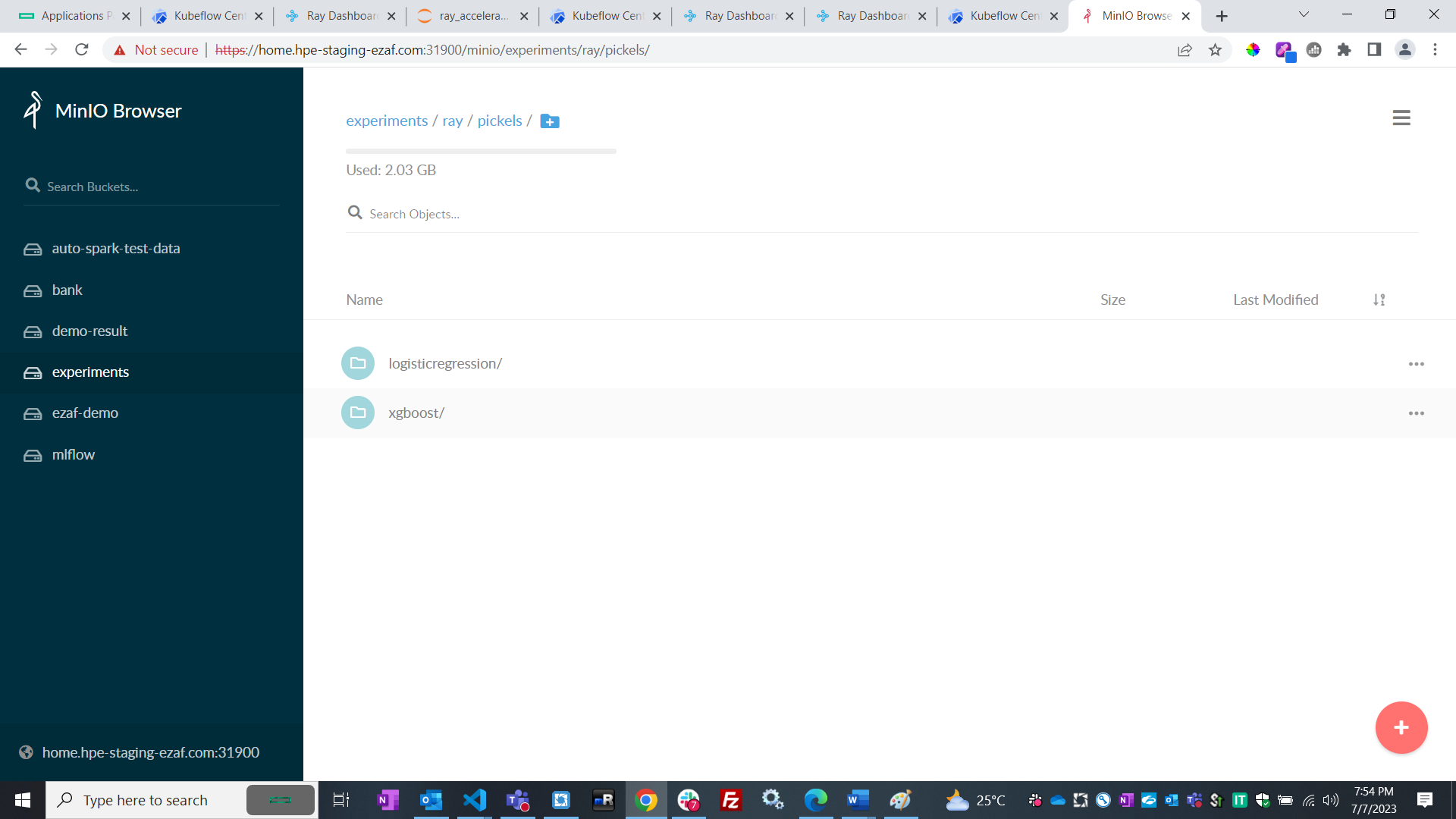


**Actor snip: RayFraudDetectionExperiment**

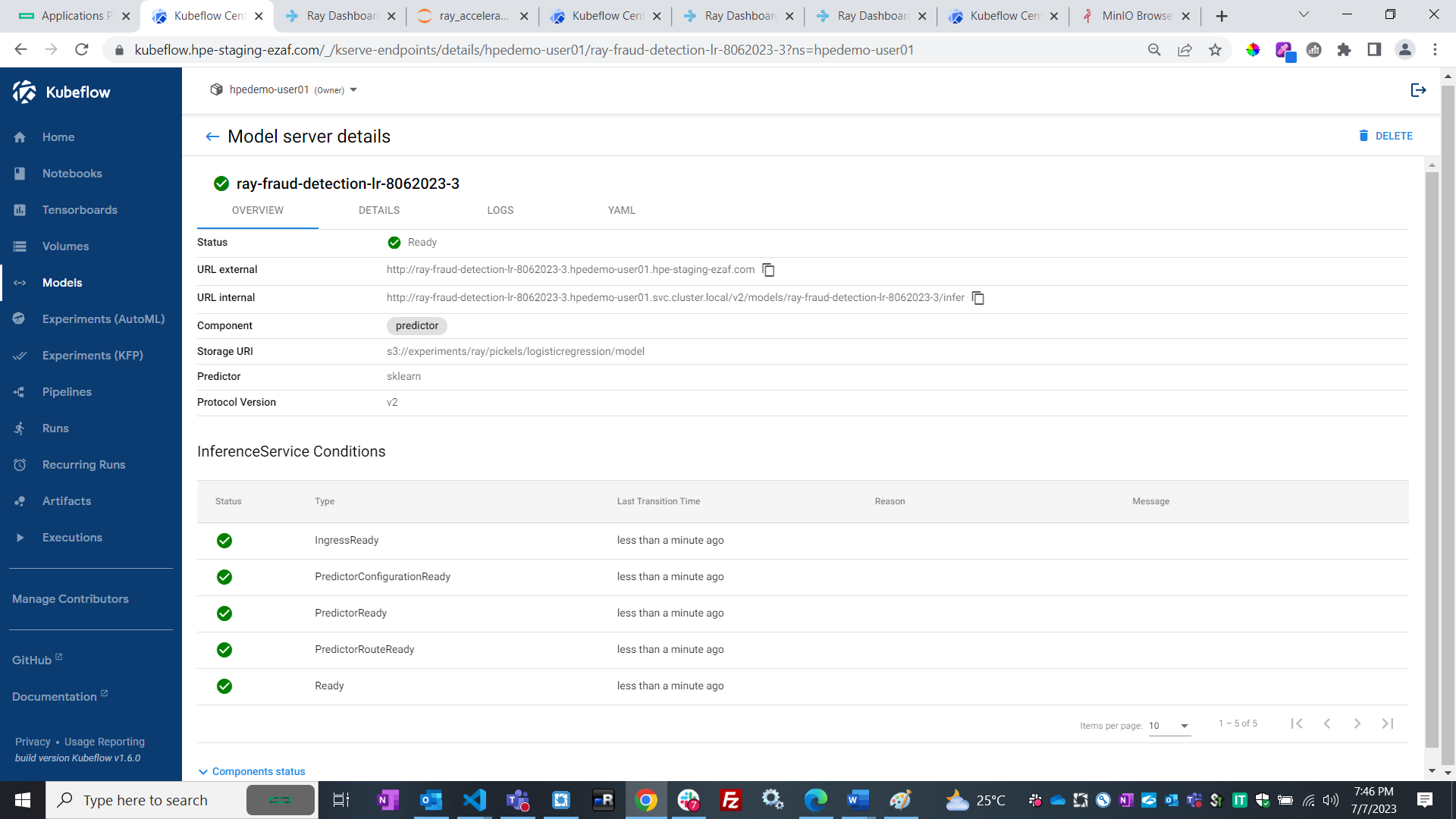




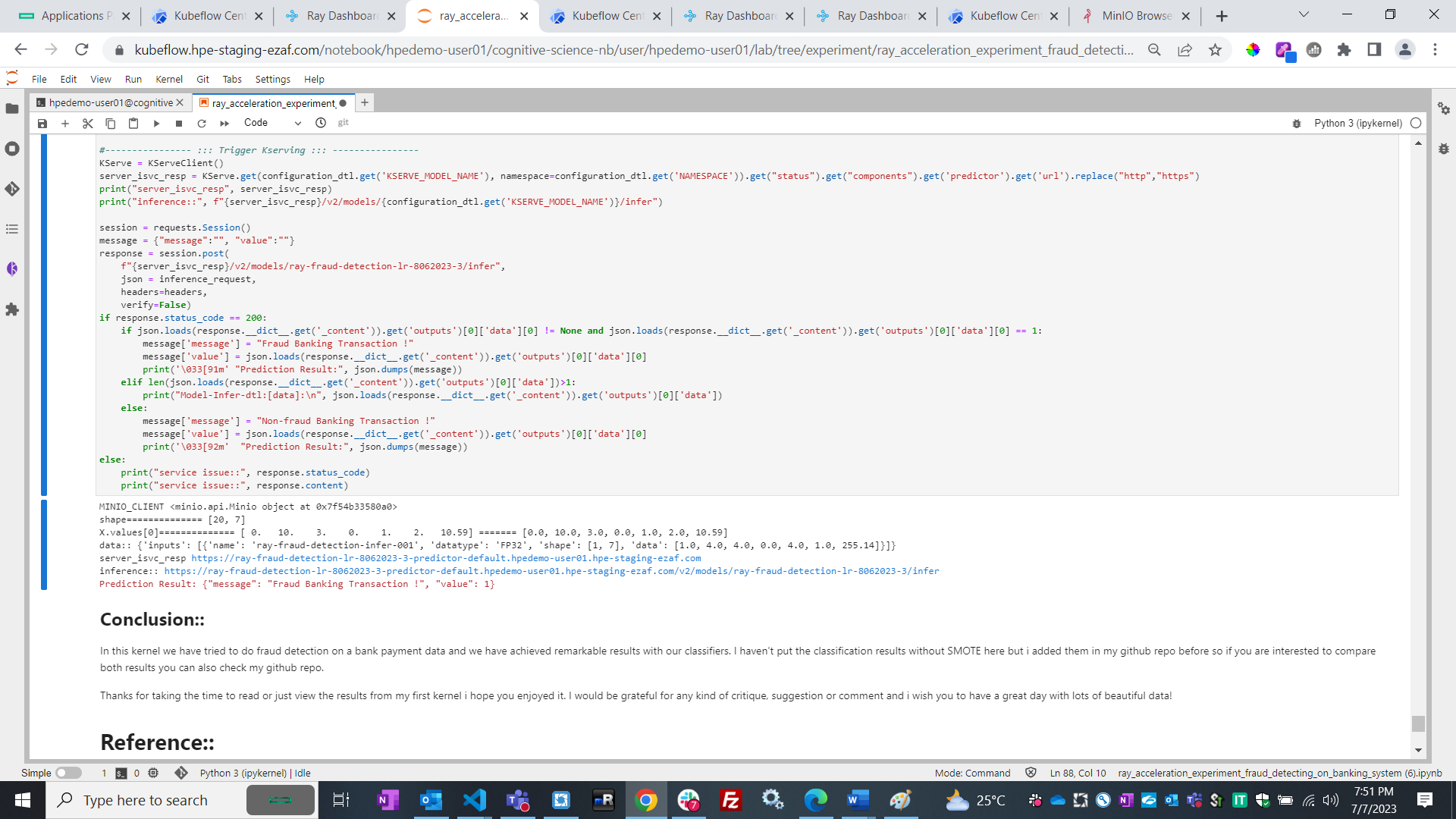
Model upload into object store (s3/data frabric customized minio)



Model deployed into Kubeflow (kserve)



Inference and prediction results:



**Conclusion:**

In this kernel we have tried to do fraud detection on a bank payment data and we have achieved remarkable results with our classifiers. I haven't put the classification results without SMOTE here but i added them in my github repo before so if you are interested to compare both results you can also check my github repo.

Thanks for taking the time to read or just view the results from my first kernel i hope you enjoyed it. I would be grateful for any kind of critique, suggestion or comment and i wish you to have a great day with lots of beautiful data!

**Reference:**

1. Lavion, Didier; et al, "PwC's Global Economic Crime and Fraud Survey 2022",
2. <https://www.pwc.com/gx/en/services/forensics/economic-crime-survey.html> |
3. <https://www.pwc.com/gx/en/services/forensics/gecs/outcomes-of-platform-fraud.svg> |
4. <https://www.pwc.com/gx/en/forensics/gecsm-2022/pdf/PwC%E2%80%99s-Global-Economic-Crime-and-Fraud-Survey-2022.pdf> \*\*(pdf) | PwC.com. Retrieved PwC’s Global Economic Crime and Fraud Survey 2022 \*\*
5. SMOTE: Synthetic Minority Over-sampling Technique, <https://jair.org/index.php/jair/article/view/10302>
6. Banksim Data Set,paper <http://www.msc-les.org/proceedings/emss/2014/EMSS2014_144.pdf> **(pdf)**