

NUMBER RECOGNITION ON AWS

Deborah Dore 1994616
Benigno Ansanelli 1836893
Vincenzo Imperati 1836893



Content



Overview



Design



Implementation



Deployment



Tests and Results



Conclusions

OVERVIEW

The goal of this project is to construct a web application and test its ability to scale when stressed using appropriate tools.

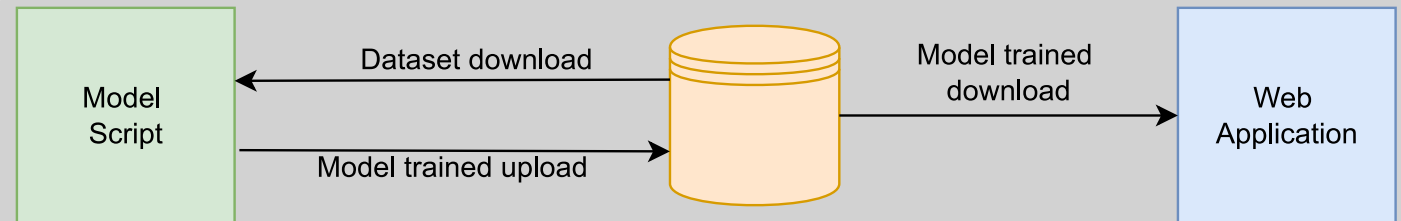
The project is fundamentally composed of 3 parts:

- *Web application*
- *Storage*
- *A Python script that contains the implementation of a machine learning model*



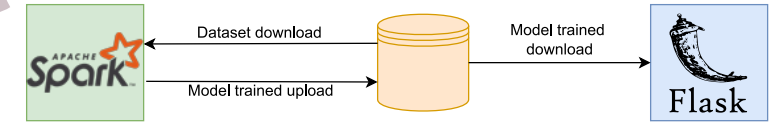
DESIGN

- The model is trained with 70,000 examples of handwritten digits (MNIST Dataset)
- The Web Application should display the capabilities of our model
- The storage is the contact point between the two parts: in here both the model and the dataset are stored

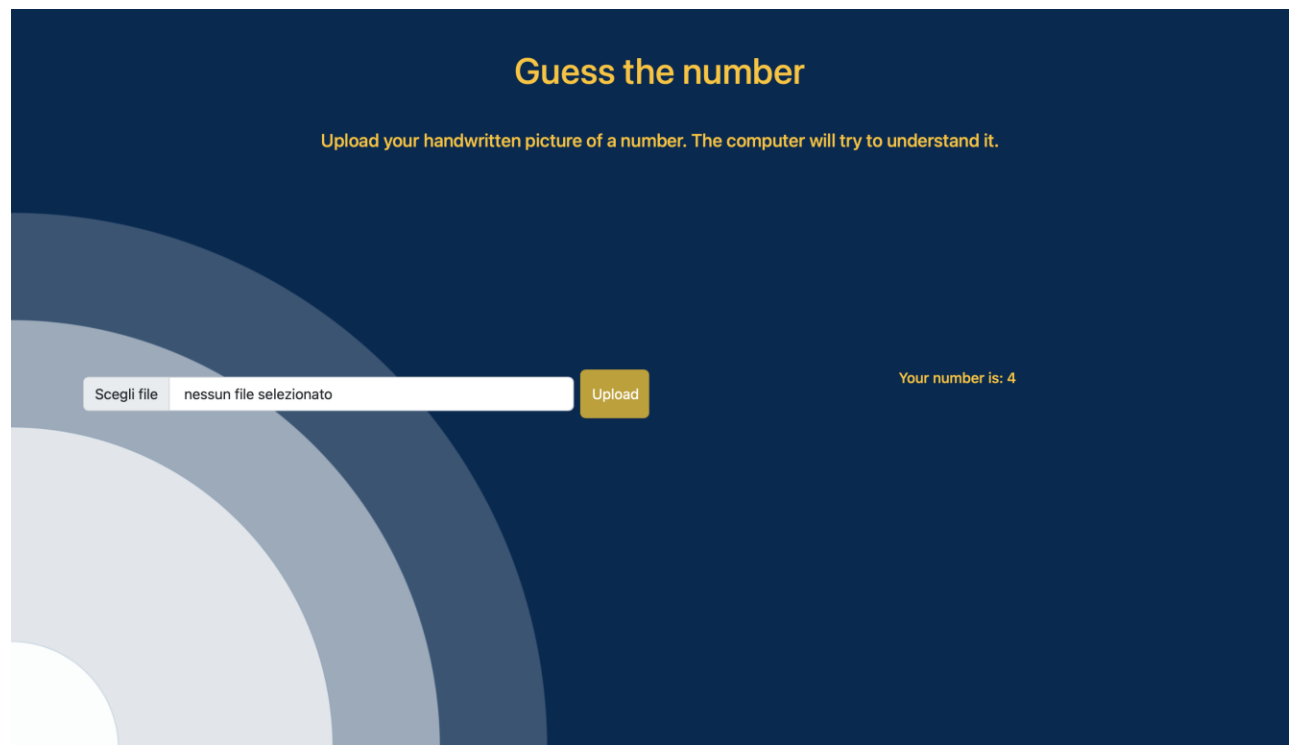
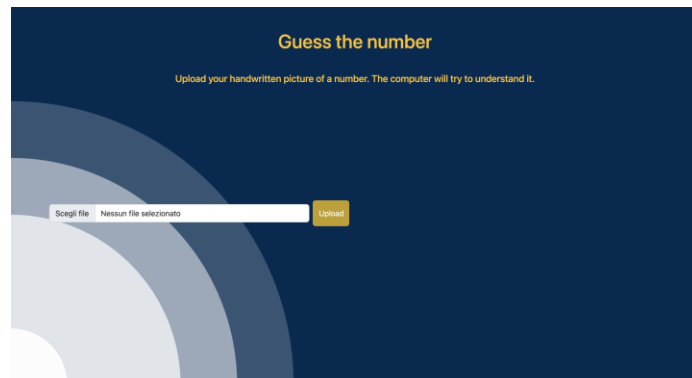
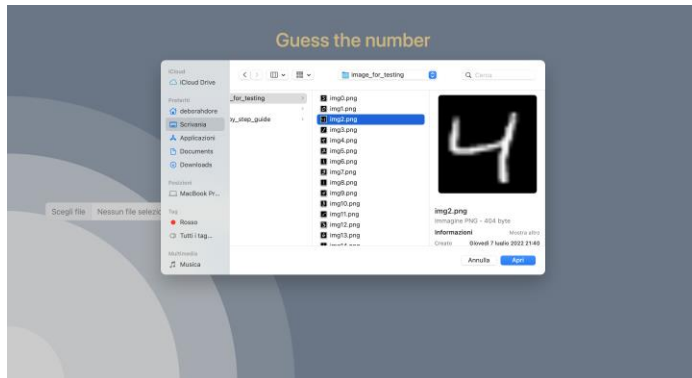


IMPLEMENTATION

- *Model Script*: Given the vast amount of instances that the model must use for the training, the best option is to use a framework capable of managing massive amount of data: Spark
- *Web Application*: Flask is a Python framework that makes web developing easy



Web App Home Page





Amazon
S3



AWS ECR



Amazon
EC2

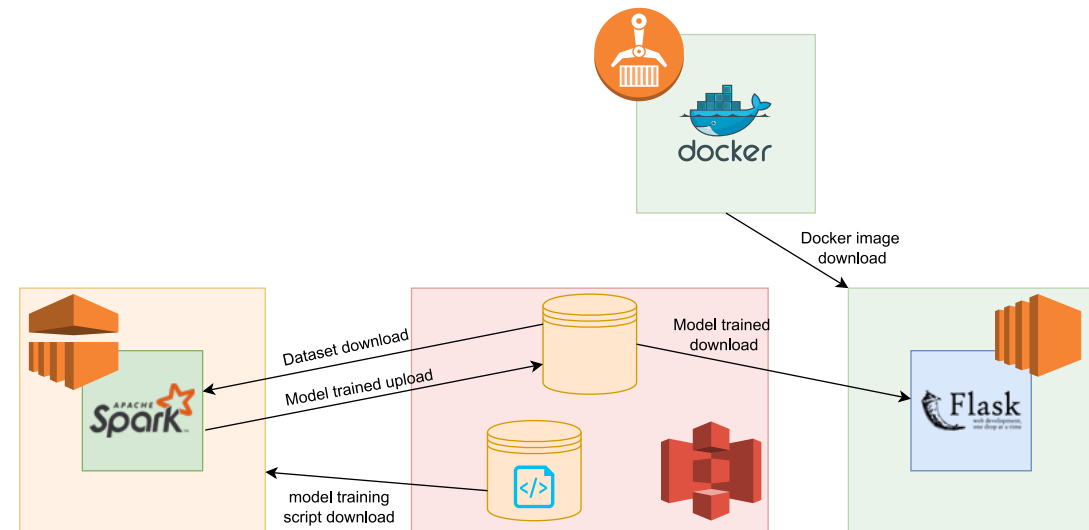


amazon
EMR

DEPLOYMENT

DEPLOYMENT

- The service corresponding to the storage is AWS S3
- The model was trained using AWS EMR
- The Web app was dockerized, its image saved on AWS ECR and deployed on AWS EC2 Autoscaling



TESTS AND RESULTS

Trade off between *cost* and *accuracy* in Amazon EMR

Testing the web application using JMeter



amazon
EMR



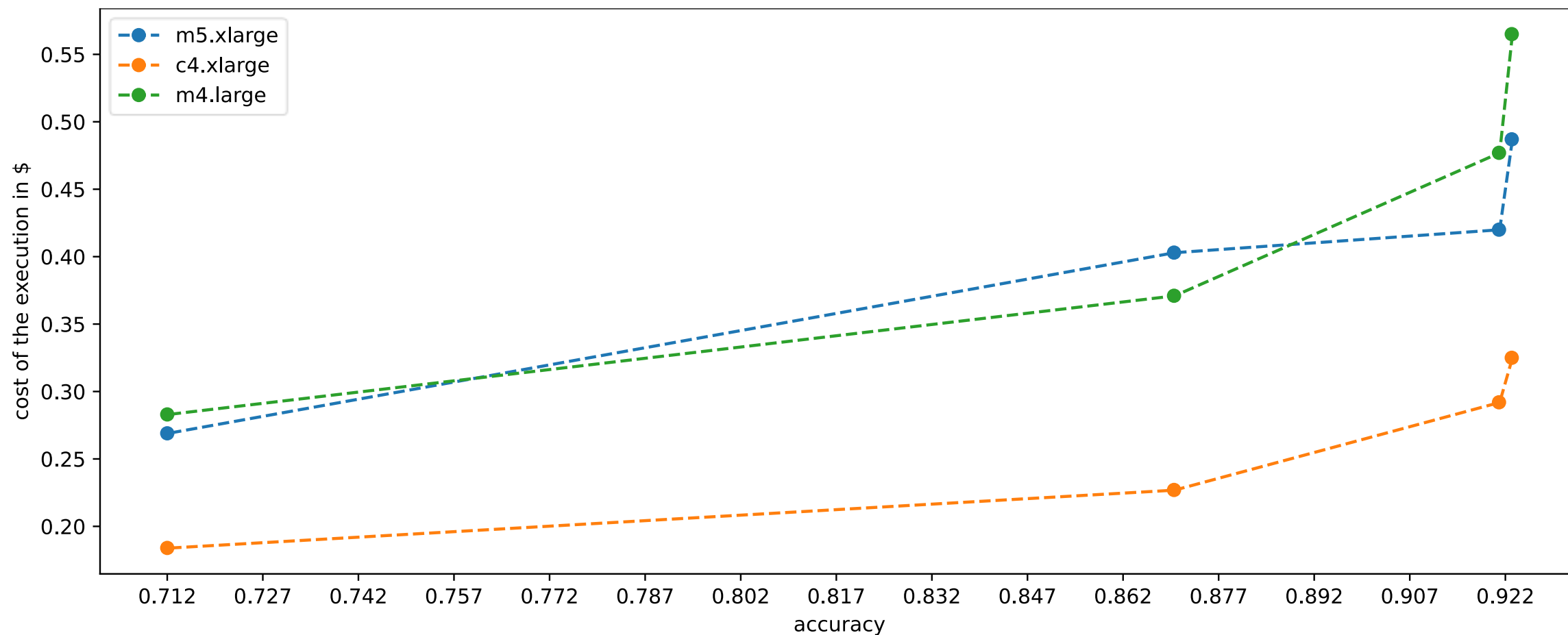
TRADE OFF BETWEEN COST AND ACCURACY IN AMAZON EMR

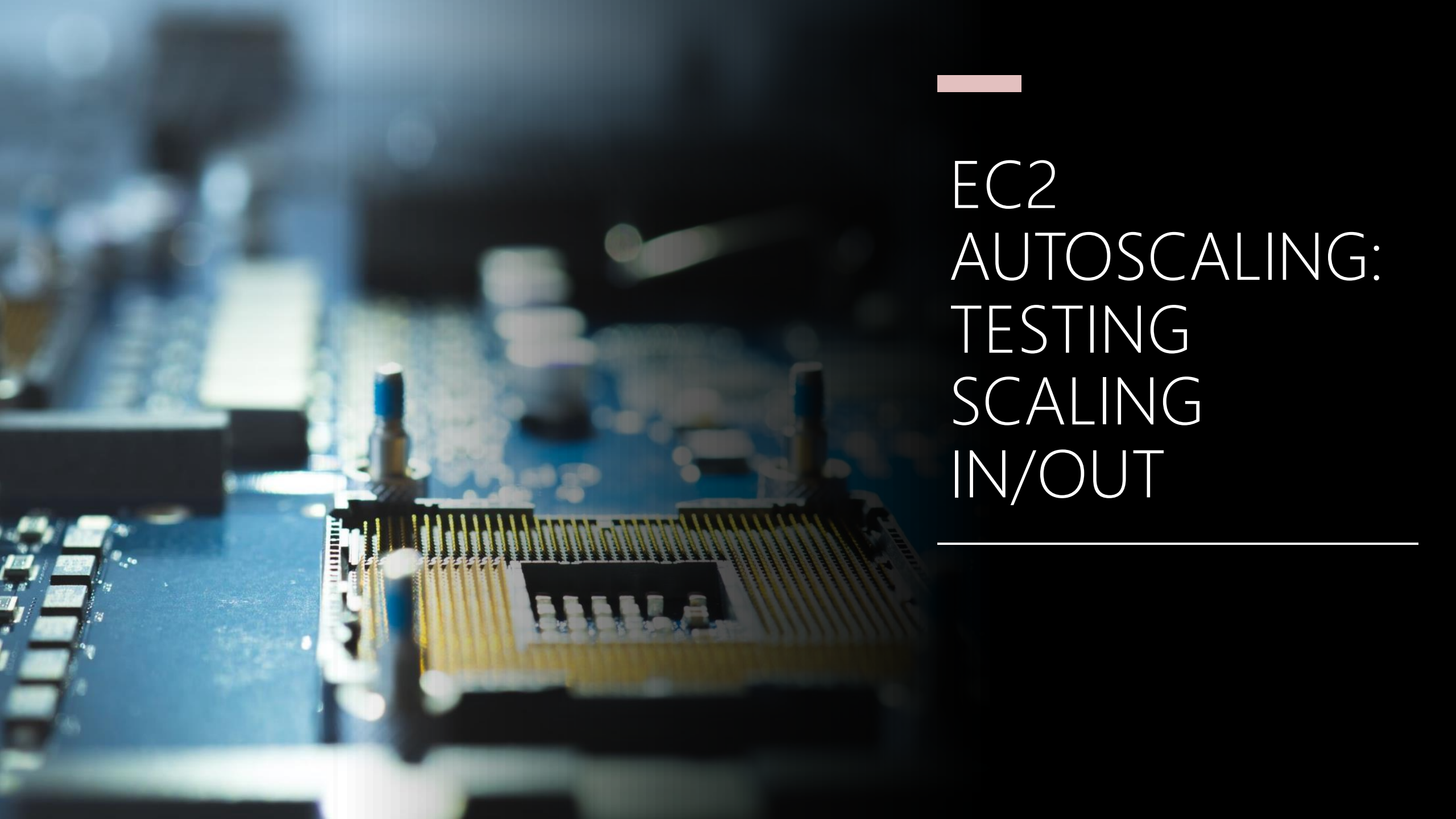
Instance type	dataset percentage	Number of instances used	accuracy	time (minutes)	cost per hour single instance	cost aws EMR per hour	cost of the execution
m5.xlarge	100%	5	0,923	29	0,192	0,048	0,487
m5.xlarge	80%	5	0,921	25	0,192	0,048	0,420
m5.xlarge	50%	5	0,870	24	0,192	0,048	0,403
m5.xlarge	30%	5	0,712	16	0,192	0,048	0,269
c4.xlarge	100%	3	0,923	30	0,199	0,052	0,325
c4.xlarge	80%	3	0,921	27	0,199	0,052	0,292
c4.xlarge	50%	3	0,870	21	0,199	0,052	0,227
c4.xlarge	30%	3	0,712	17	0,199	0,052	0,184
m4.large	100%	5	0,923	64	0,100	0,030	0,565
m4.large	80%	5	0,921	54	0,100	0,030	0,477
m4.large	50%	5	0,870	42	0,100	0,030	0,371
m4.large	30%	5	0,712	32	0,100	0,030	0,283



amazon
EMR

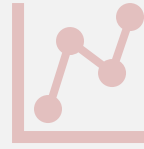
TRADE OFF BETWEEN *COST* AND *ACCURACY* IN AMAZON EMR



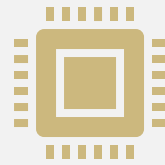


EC2 AUTOSCALING: TESTING SCALING IN/OUT

WHAT ARE WE TESTING



Given an image in input,
predict a number



CPU BOUND

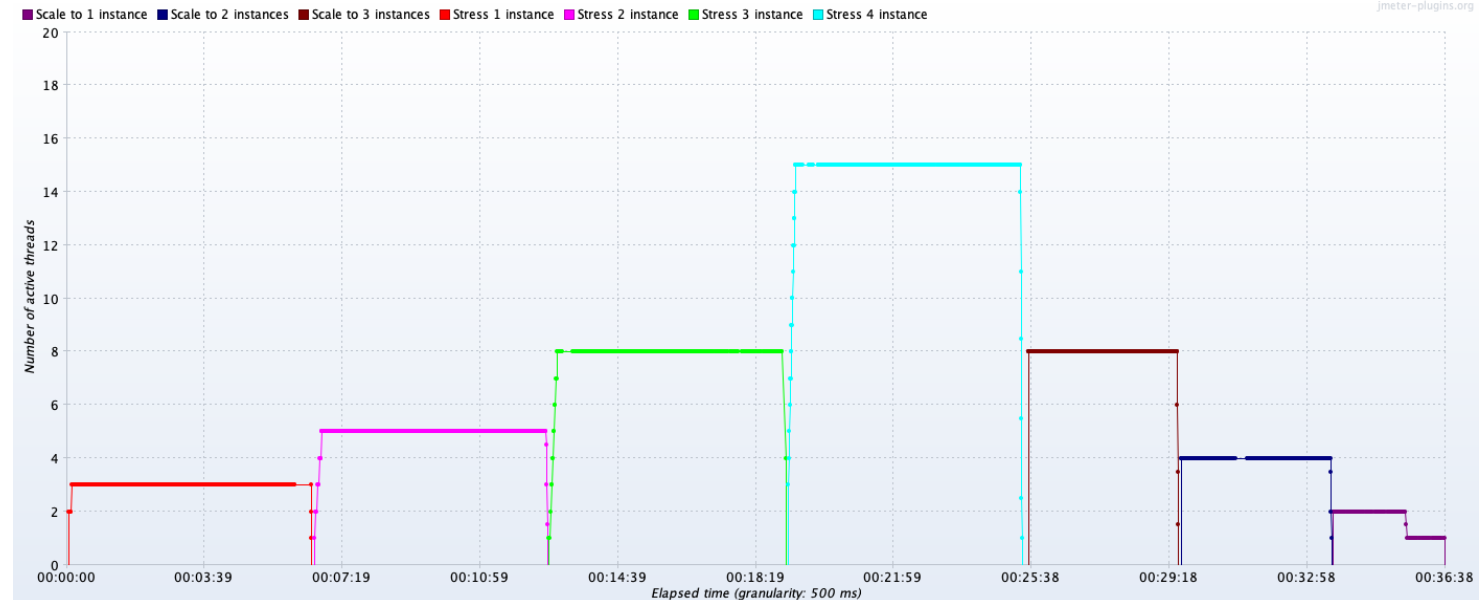


GET VS POST

JMETER CONFIGURATION



- Threads
- Concurrent requests
- Increase and decrease the load
- Different size images



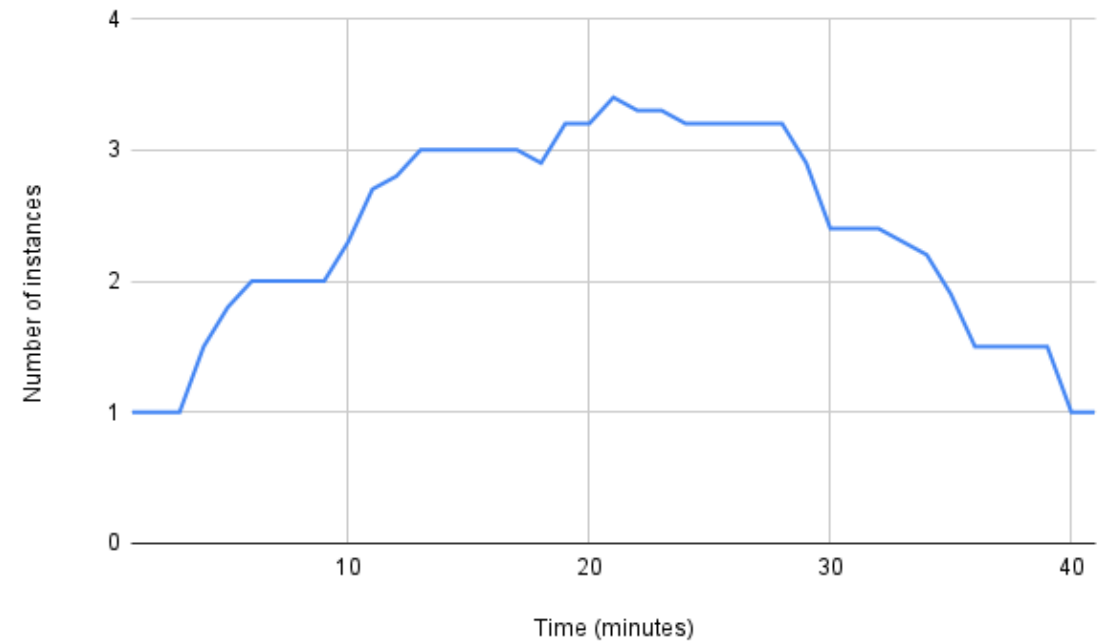
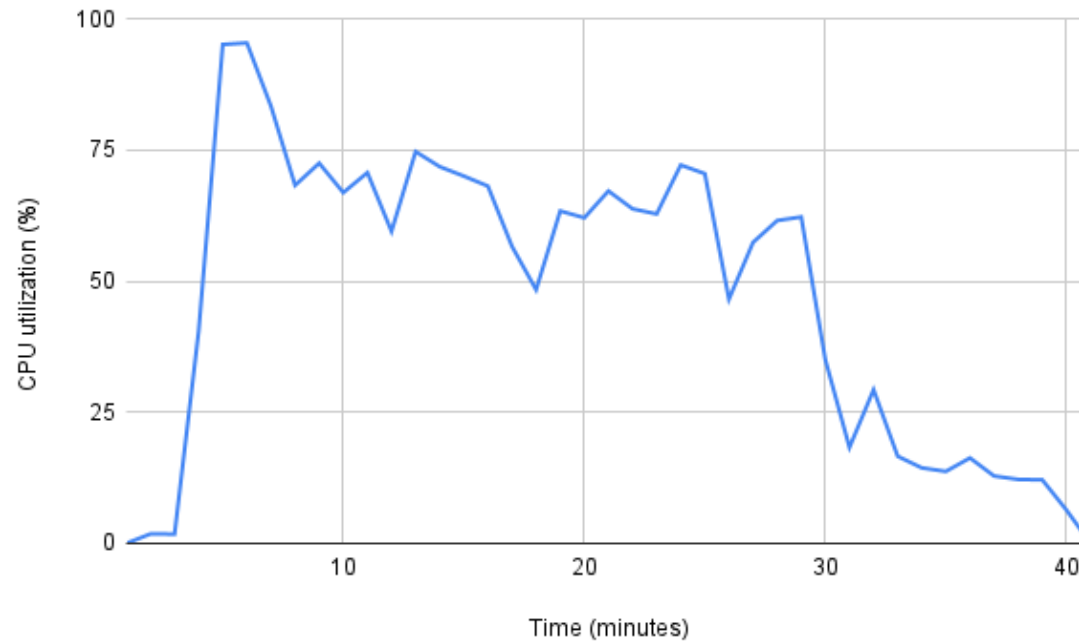
AUTOSCALING GROUP CONFIGURATION

- Simple scaling policy
- If average load >66% add 1 instance
- If average load <40% remove 1 instance

Instance type	vCPU	RAM (GiB)	Cost per hour
t2.small	1	2	0,023\$
t2.medium	2	4	0,0464\$

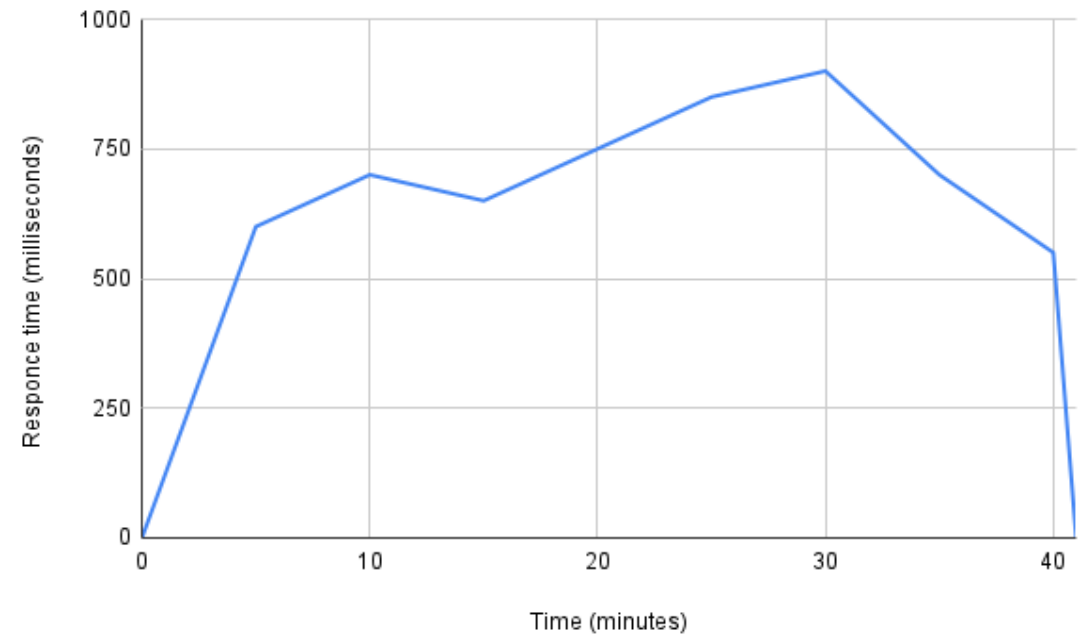
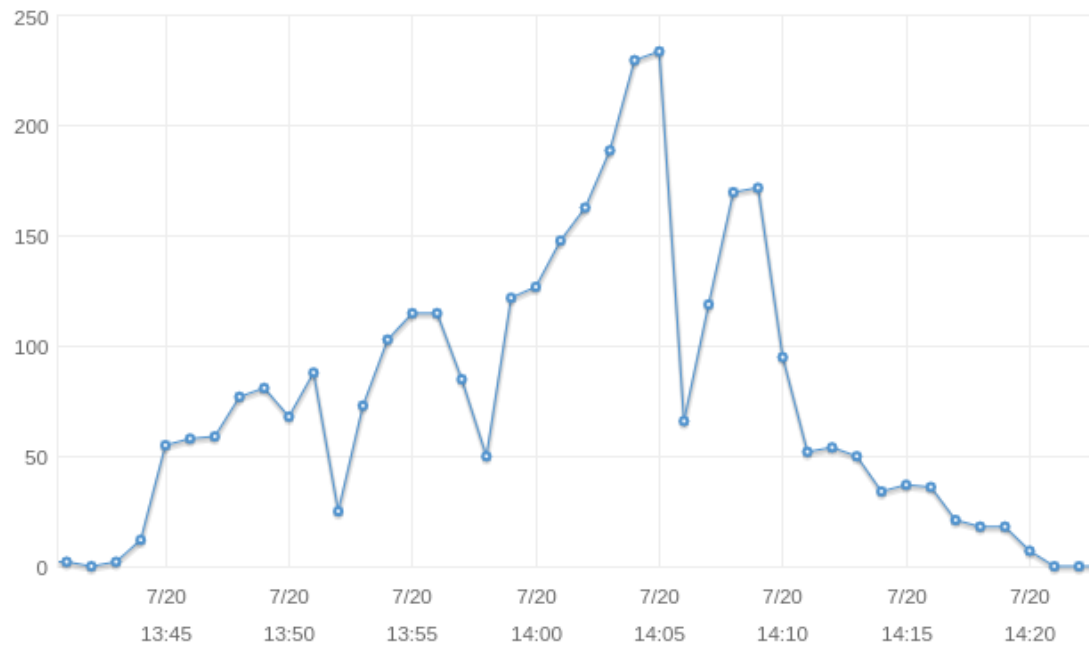
TEST ON SMALL INSTANCE

- CPU Utilization (left)
- Number of instances (right)



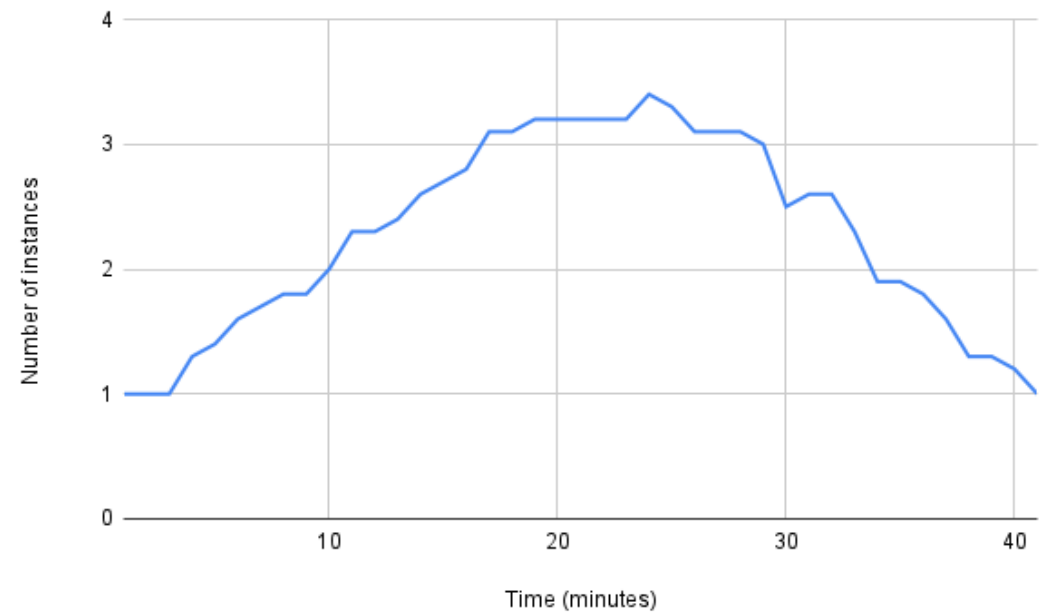
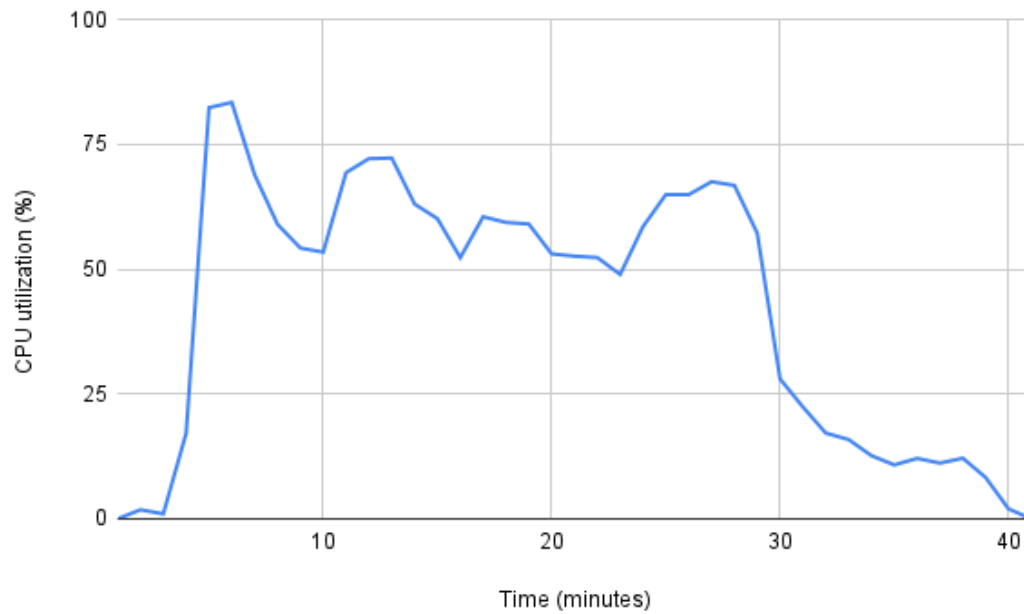
TEST ON SMALL INSTANCE

- Number of requests (left)
- Response Time (right)



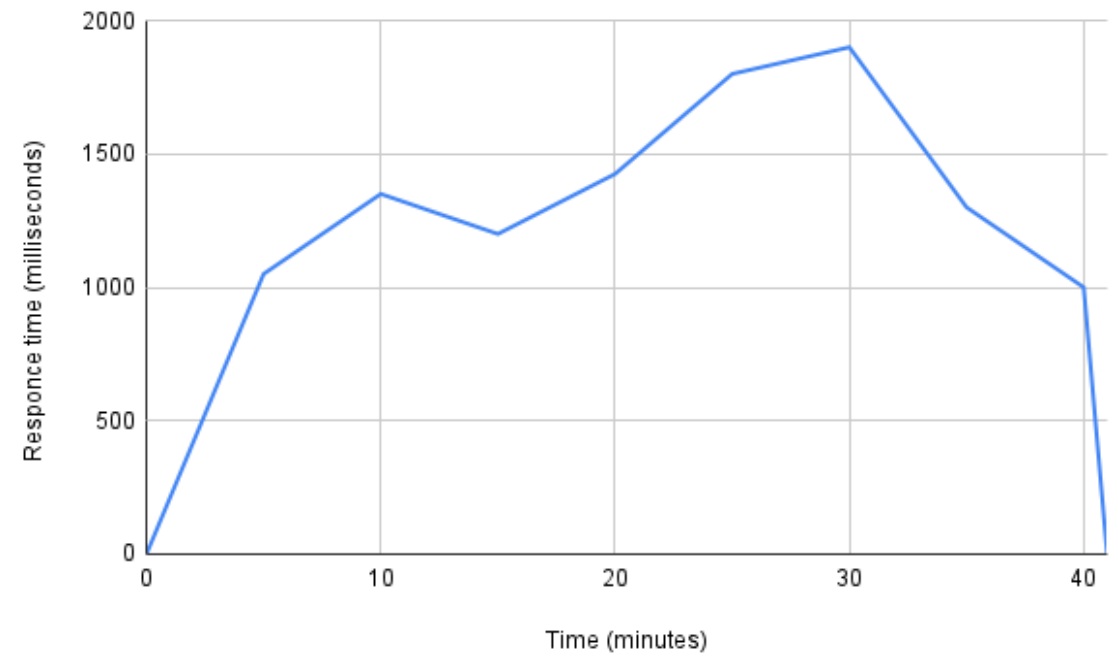
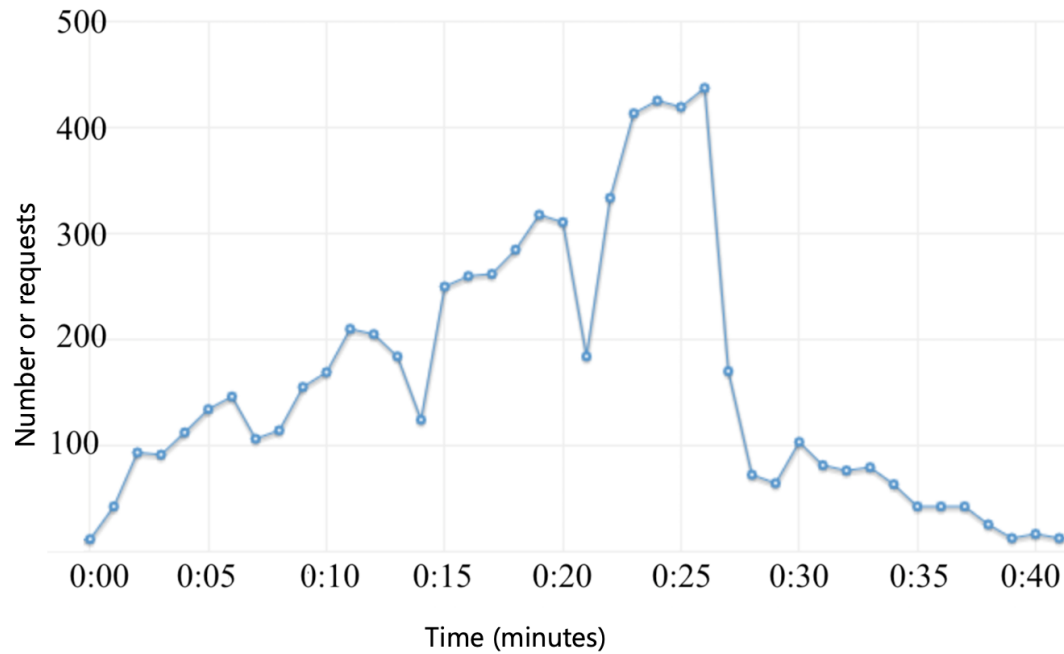
TEST ON MEDIUM INSTANCE

- CPU Utilization (left)
- Number of instances (right)

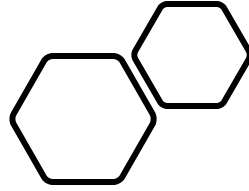


TEST ON MEDIUM INSTANCE

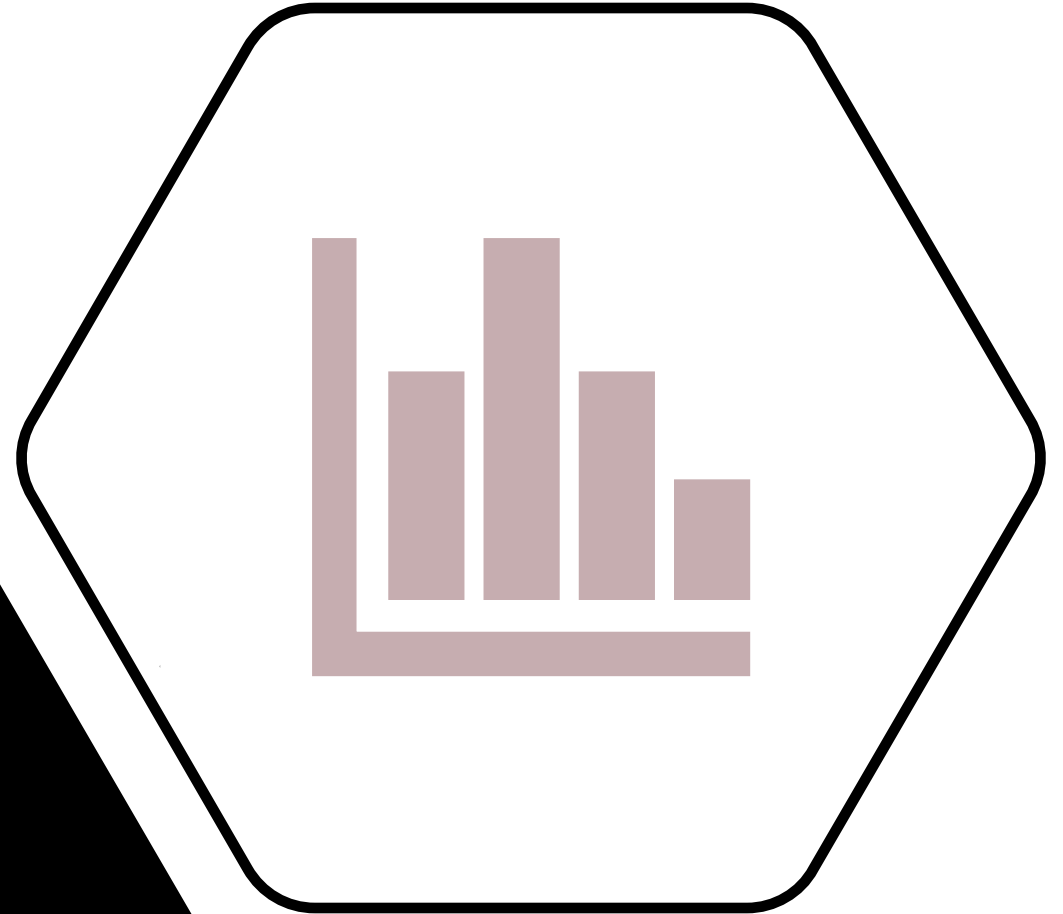
- Number of requests (left)
- Response Time (right)



RESULTS



- Scaling capabilities
- Small vs Medium instance type
- Time for an instance to be operative



CONCLUSIONS

- Aws sandbox limitations
- Different ways to test the services offered by amazon
- Total cost of the application one year is \$1619,76 using the t2.medium instance type for EC2 and the c4.xlarge instance type for EMR



THANKS FOR THE ATTENTION