TOPICS IN NETWORKS TERM PROJECT

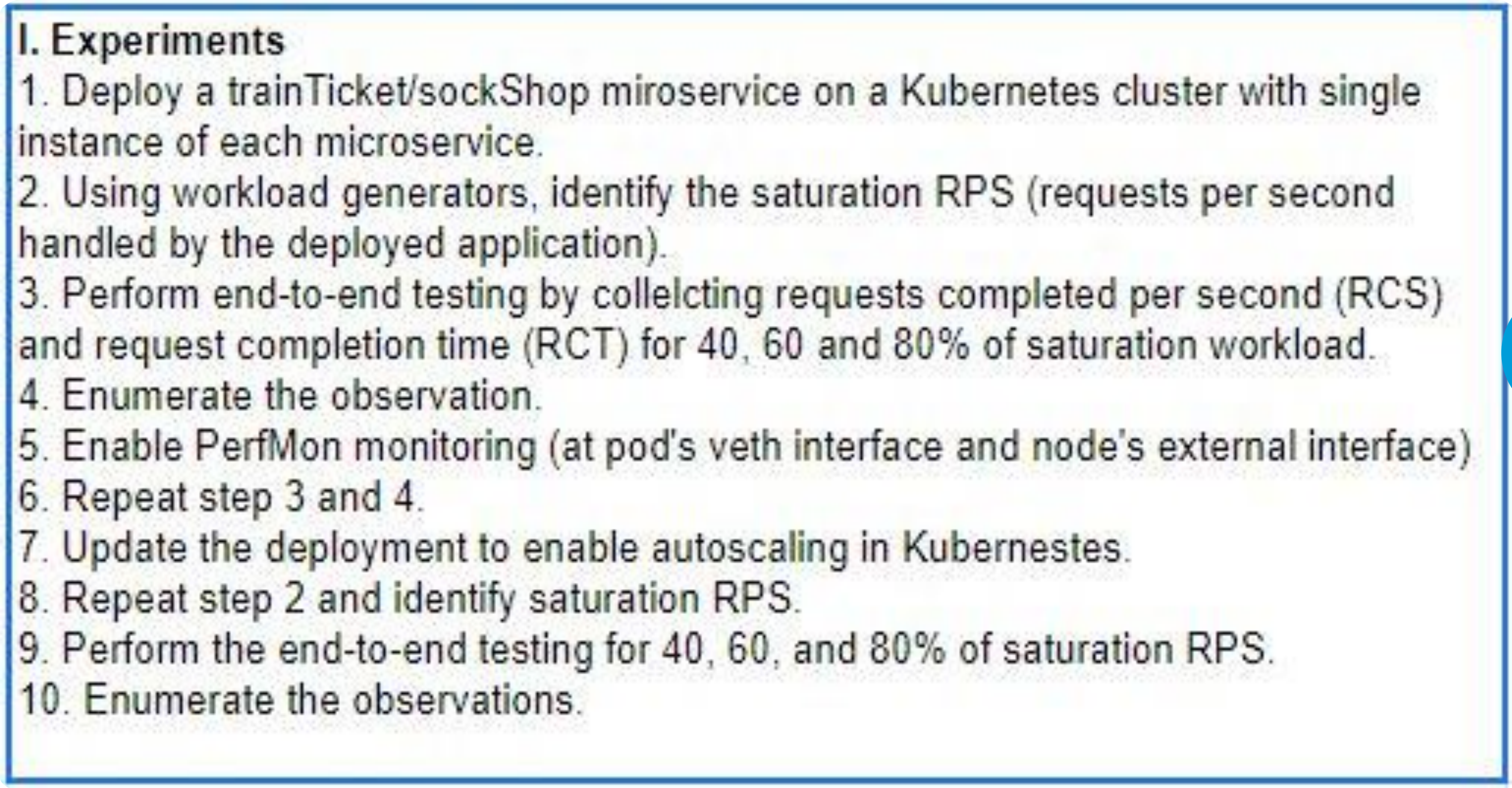
- Group Members:
- 1. Ashutosh Rajput : CS23MTECH11005
- 2. Akshat Gupta : CS23MTECH11001
- 3. Malsawmsanga Sailo : CS23MTECH11010

- Mentors:
- 1. Ranjitha K : CS21RESCH01002
- 2. Ankit Sharma : CS22MTECH12003

- Guided by : Dr. Praveen Tammana

Project Details

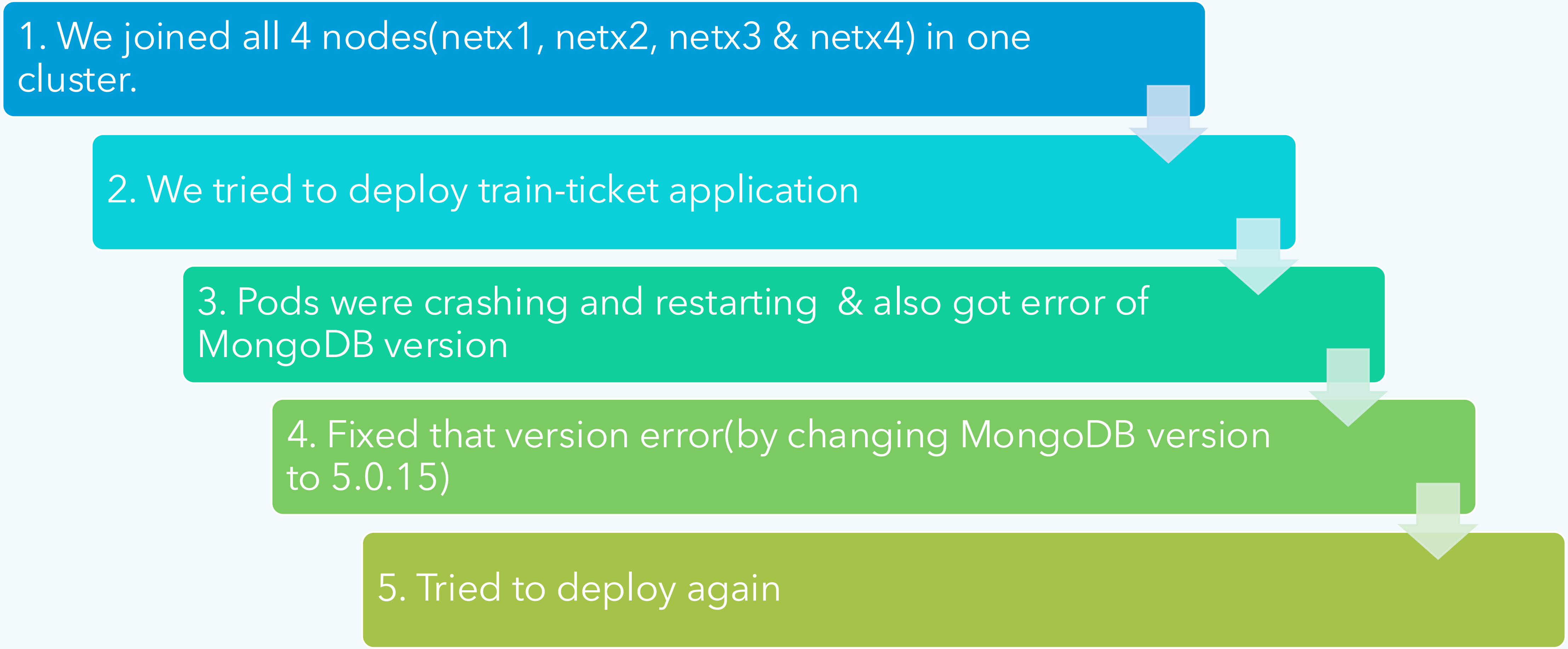
Project Number : P2



I. Experiments

1. Deploy a trainTicket/sockShop miroservice on a Kubernetes cluster with single instance of each microservice.
2. Using workload generators, identify the saturation RPS (requests per second handled by the deployed application).
3. Perform end-to-end testing by collelcting requests completed per second (RCS) and request completion time (RCT) for 40, 60 and 80% of saturation workload.
4. Enumerate the observation.
5. Enable PerfMon monitoring (at pod's veth interface and node's external interface)
6. Repeat step 3 and 4.
7. Update the deployment to enable autoscaling in Kubernestes.
8. Repeat step 2 and identify saturation RPS.
9. Perform the end-to-end testing for 40, 60, and 80% of saturation RPS.
10. Enumerate the observations.

Summary : Work done till last review



1. We joined all 4 nodes(netx1, netx2, netx3 & netx4) in one cluster.

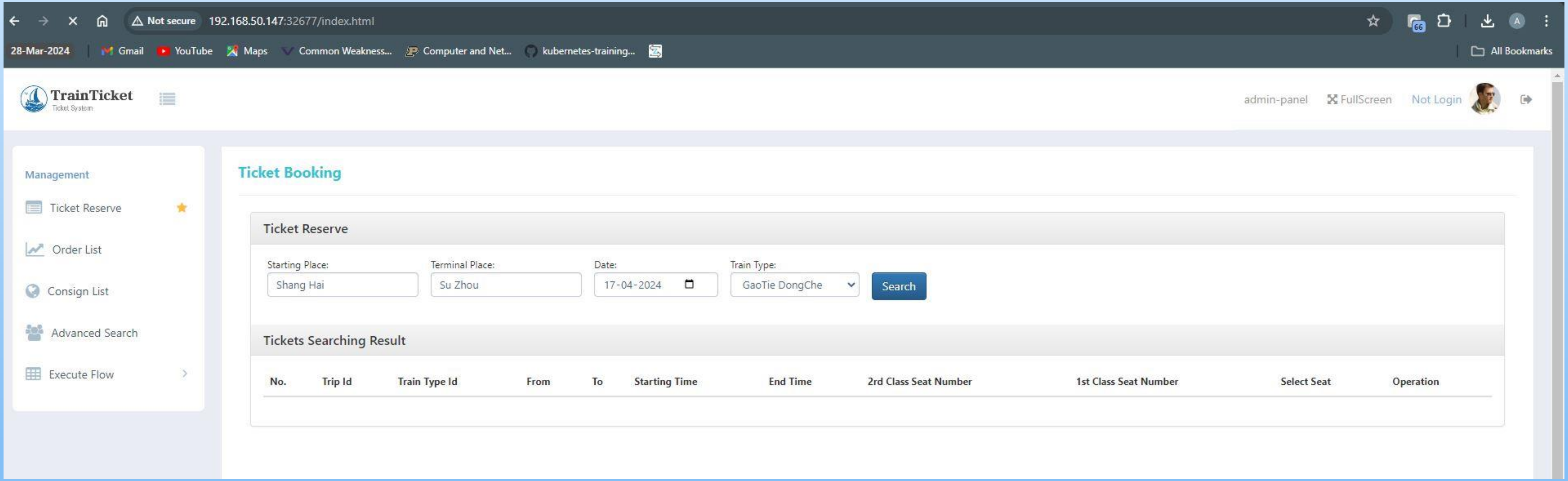
2. We tried to deploy train-ticket application

3. Pods were crashing and restarting & also got error of MongoDB version

4. Fixed that version error(by changing MongoDB version to 5.0.15)

5. Tried to deploy again

Deployed our train-ticket application



Ticket Booking

Ticket Reserve

Starting Place: Shang Hai

Terminal Place: Su Zhou

Date: 17-04-2024

Train Type: GaoTie DongChe

Search

Tickets Searching Result

No.	Trip Id	Train Type Id	From	To	Starting Time	End Time	2rd Class Seat Number	1st Class Seat Number	Select Seat	Operation

Objectives

1. Identify the saturation point(in Requests per second) using work load generators

2. Make script for generating the work load

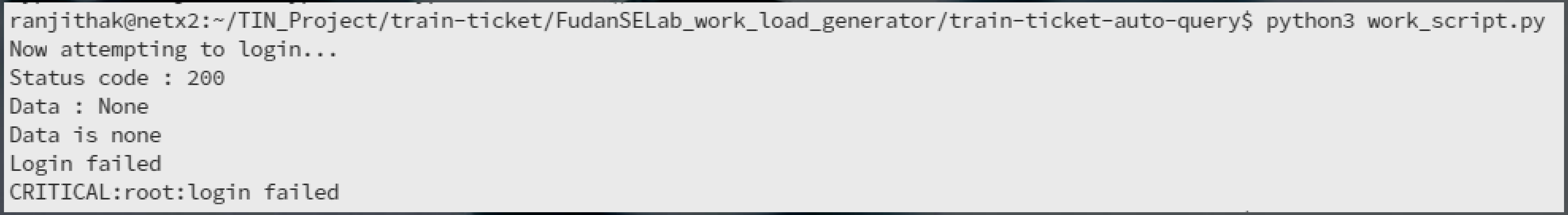
3. The script should hit the Microservices

4. Enable PerfMon monitoring and do end-to-end testing again

5. Update the deployment to use Auto-scaling and do testing again and find the saturation RPS

Week 3

- We used the python script to generate load and tried to run it and got error : login failed.
- Found out that the problem was in
  - Verification code
  - **ts-auth-service** pod



```
ranjithak@netx2:~/TIN_Project/train-ticket/FudanSELab_work_load_generator/train-ticket-auto-query$ python3 work_script.py
Now attempting to login...
Status code : 200
Data : None
Data is none
Login failed
CRITICAL:root:login failed
```

Github Repo : https://github.com/FudanSELab/train-ticket-auto-query

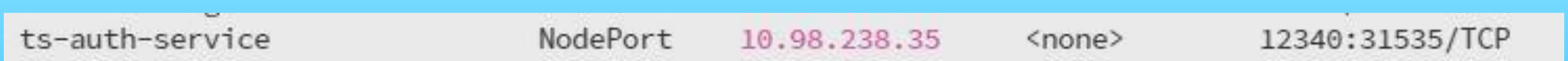
Week 3

Solution :

1. We redeployed the **ts-auth-service** as NodePort
2. We set the verification code to an empty string
3. Parsed data using json

Result : We successfully logged in

But still problems were there in posting and fetching the content



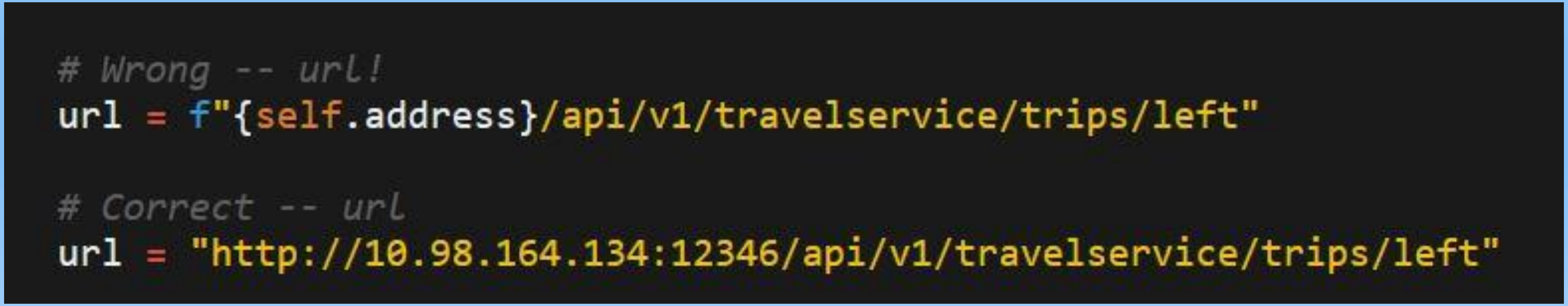
```
ts-auth-service                 NodePort    10.98.238.35     <none>        12340:31535/TCP
```

Week 4

1. WE RUN IT AGAIN BUT WE GOT THE ERROR : **DATA NULL**

2. THEN WE DEBUGGED THE SCRIPT AND FOUND THAT **FOR EVERY SERVICE IT IS HITTING THE AUTH-SERVICE URL**.

3. THEN WE **HARD CODED THE IP AND PORT** FOR EVERY SERVICE.

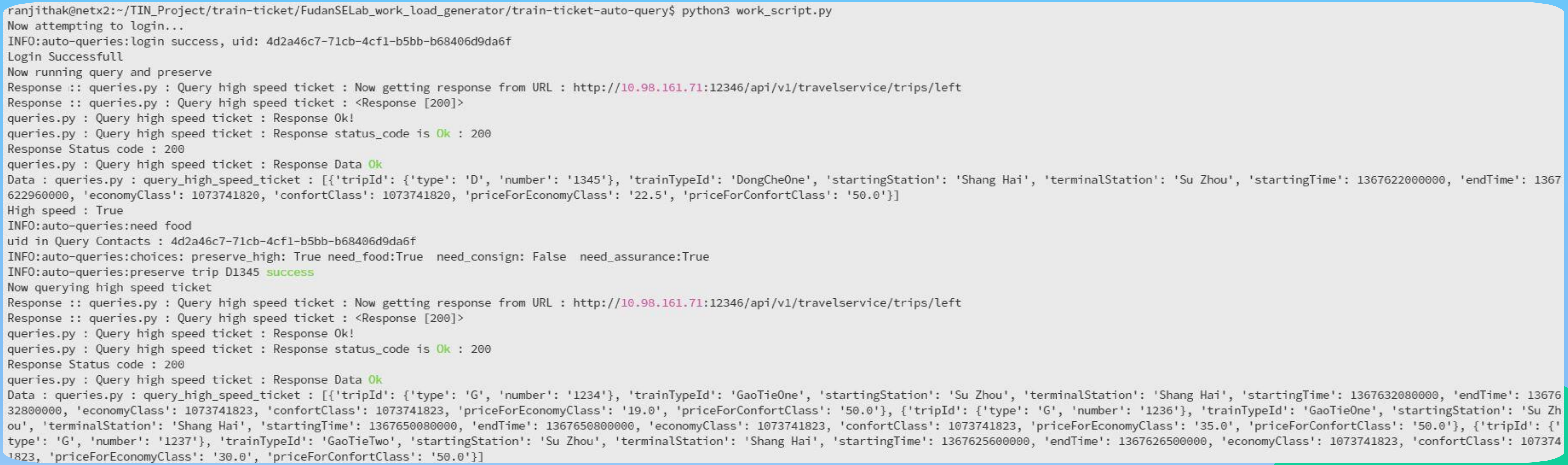


```
# Wrong -- url!
url = f"{self.address}/api/v1/travelservice/trips/left"

# Correct -- url
url = "http://10.98.164.134:12346/api/v1/travelservice/trips/left"
```

Week 4

Then we were able to run our script Successfully



```
ranjithak@netx2:~/TIN_Project/train-ticket/FudanSELab_work_load_generator/train-ticket-auto-query$ python3 work_script.py
Now attempting to login...
INFO:auto-queries:login success, uid: 4d2a46c7-71cb-4cf1-b5bb-b68406d9da6f
Login Successfull
Now running query and preserve
Response :: queries.py : Query high speed ticket : Now getting response from URL : http://10.98.161.71:12346/api/v1/travelservice/trips/left
Response :: queries.py : Query high speed ticket : <Response [200]>
queries.py : Query high speed ticket : Response Ok!
queries.py : Query high speed ticket : Response status_code is Ok : 200
Response Status code : 200
queries.py : Query high speed ticket : Response Data Ok
Data : queries.py : query_high_speed_ticket : [{'tripId': {'type': 'D', 'number': '1345'}, 'trainTypeId': 'DongCheOne', 'startingStation': 'Shang Hai', 'terminalStation': 'Su Zhou', 'startingTime': 1367622000000, 'endTime': 1367
622960000, 'economyClass': 1073741820, 'confortClass': 1073741820, 'priceForEconomyClass': '22.5', 'priceForConfortClass': '50.0'}]
High speed : True
INFO:auto-queries:need food
uid in Query Contacts : 4d2a46c7-71cb-4cf1-b5bb-b68406d9da6f
INFO:auto-queries:choices: preserve_high: True need_food:True  need_consign: False  need_assurance:True
INFO:auto-queries:preserve trip D1345 success
Now querying high speed ticket
Response :: queries.py : Query high speed ticket : Now getting response from URL : http://10.98.161.71:12346/api/v1/travelservice/trips/left
Response :: queries.py : Query high speed ticket : <Response [200]>
queries.py : Query high speed ticket : Response Ok!
queries.py : Query high speed ticket : Response status_code is Ok : 200
Response Status code : 200
queries.py : Query high speed ticket : Response Data Ok
Data : queries.py : query_high_speed_ticket : [{'tripId': {'type': 'G', 'number': '1234'}, 'trainTypeId': 'GaoTieOne', 'startingStation': 'Su Zhou', 'terminalStation': 'Shang Hai', 'startingTime': 1367632080000, 'endTime': 13676
32800000, 'economyClass': 1073741823, 'confortClass': 1073741823, 'priceForEconomyClass': '19.0', 'priceForConfortClass': '50.0'}, {'tripId': {'type': 'G', 'number': '1236'}, 'trainTypeId': 'GaoTieOne', 'startingStation': 'Su Zh
ou', 'terminalStation': 'Shang Hai', 'startingTime': 1367650080000, 'endTime': 1367650800000, 'economyClass': 1073741823, 'confortClass': 1073741823, 'priceForEconomyClass': '35.0', 'priceForConfortClass': '50.0'}, {'tripId': {'
type': 'G', 'number': '1237'}, 'trainTypeId': 'GaoTieTwo', 'startingStation': 'Su Zhou', 'terminalStation': 'Shang Hai', 'startingTime': 1367625600000, 'endTime': 1367626500000, 'economyClass': 1073741823, 'confortClass': 107374
1823, 'priceForEconomyClass': '30.0', 'priceForConfortClass': '50.0'}]
```

Week 4

We tried to generate the work load using wrk2

Command :

```
./wrk -t32 -c50 -d30s -L -s ~/TIN_Project/train-ticket/FudanSELab_work_load_generator/train-ticket-auto-query/work_script.py http://10.97.254.81:12340 -R500
```

We were giving wrk2 Test Parameters as

Threads : 32

Connections : 50

Duration : 30 seconds

Request per second : 500

Result:

Average Request completed : 24-28 request per second

Week 4

- Increased the number of request to 5000
- But our services went down
- We tried to run the script again
  - Got the same error : Data null!
  - Even not able to login
- Then we re-deployed everything

Week 4

01

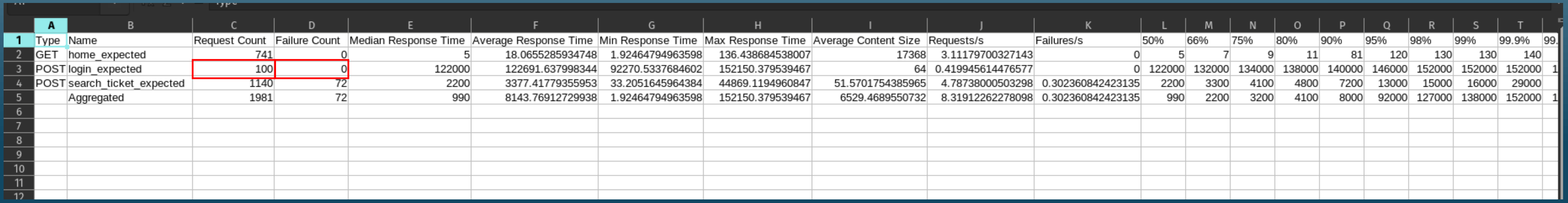
We got the locust python script and fixed all the IP and ports

02

Then we tried to generate the work load with 100 users

03

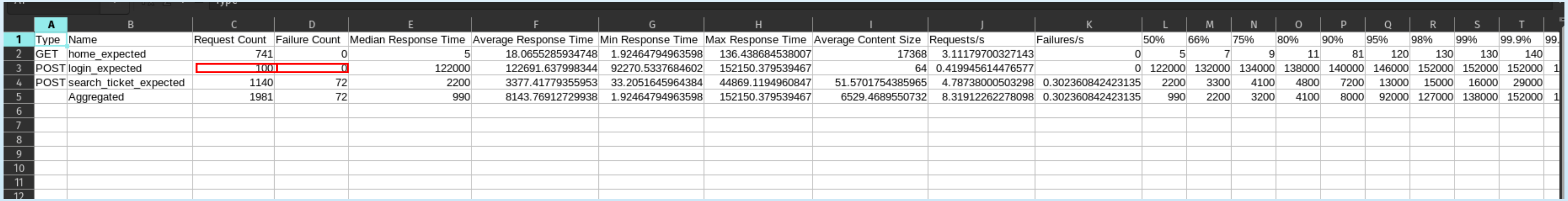
We got all request successful



Type	Name	Request Count	Failure Count	Median Response Time	Average Response Time	Min Response Time	Max Response Time	Average Content Size	Requests/s	Failures/s	50%	66%	75%	80%	90%	95%	98%	99%	99.9%	99.
GET	home_expected	741	0	5	18.0655285934748	1.92464794963598	136.438684538007	17368	3.11179700327143	0	5	7	9	11	81	120	130	130	140	
POST	login_expected	100	0	122000	122691.637998344	92270.5337684602	152150.379539467	64	0.419945614476577	0	122000	132000	134000	138000	140000	146000	152000	152000	152000	1
POST	search_ticket_expected	1140	72	2200	3377.41779355953	33.2051645964384	44869.1194960847	51.5701754385965	4.78738000503298	0.302360842423135	2200	3300	4100	4800	7200	13000	15000	16000	29000	
	Aggregated	1981	72	990	8143.76912729938	1.92464794963598	152150.379539467	6529.4689550732	8.31912262278098	0.302360842423135	990	2200	3200	4100	8000	92000	127000	138000	152000	1

Github repo : https://github.com/rajibhossen/ts-locust-load-generator?tab=readme-ov-file

Week 4



Type	Name	Request Count	Failure Count	Median Response Time	Average Response Time	Min Response Time	Max Response Time	Average Content Size	Requests/s	Failures/s	50%	66%	75%	80%	90%	95%	98%	99%	99.9%	99.
GET	home_expected	741	0	5	18.0655285934748	1.92464794963598	136.438684538007	17368	3.11179700327143	0	5	7	9	11	81	120	130	130	140	
POST	login_expected	100	0	122000	122691.637998344	92270.5337684602	152150.379539467	64	0.419945614476577	0	122000	132000	134000	138000	140000	146000	152000	152000	152000	1
POST	search_ticket_expected	1140	72	2200	3377.41779355953	33.2051645964384	44869.1194960847	51.5701754385965	4.78738000503298	0.302360842423135	2200	3300	4100	4800	7200	13000	15000	16000	29000	
	Aggregated	1981	72	990	8143.76912729938	1.92464794963598	152150.379539467	6529.4689550732	8.31912262278098	0.302360842423135	990	2200	3200	4100	8000	92000	127000	138000	152000	1

Results :

1. Avg response time(50% requests) : 12200 millisecond = 12.2 sec

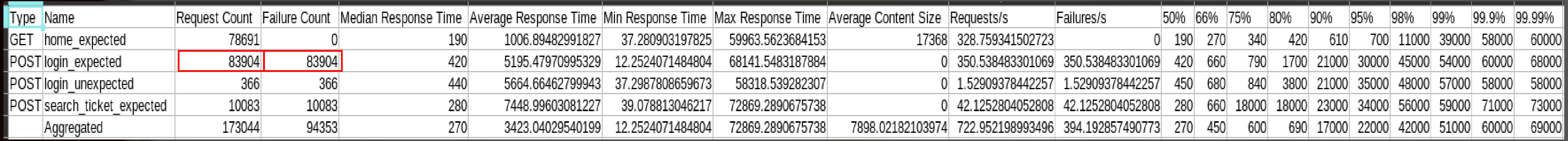
2. 75% requests response time : 13400 millisecond = 13.4 sec

3. 90% requests response time : 14000 millisecond = 14 sec

4. 100% requests response time : 15200 millisecond = 15.2 sec

Week 4

- Then we increased the load to 2000 users
- Similar thing occurred again : The services went down
- No Login was successful



Type	Name	Request Count	Failure Count	Median Response Time	Average Response Time	Min Response Time	Max Response Time	Average Content Size	Requests/s	Failures/s	50%	66%	75%	80%	90%	95%	98%	99%	99.9%	99.99%
GET	home_expected	78691	0	190	1006.89482991827	37.280903197825	59963.5623684153	17368	328.759341502723	0	190	270	340	420	610	700	11000	39000	58000	60000
POST	login_expected	83904	83904	420	5195.47970995329	12.2524071484804	68141.5483187884	0	350.538483301069	350.538483301069	420	660	790	1700	21000	30000	45000	54000	60000	68000
POST	login_unexpected	366	366	440	5664.66462799943	37.2987808659673	58318.539282307	0	1.52909378442257	1.52909378442257	450	680	840	3800	21000	35000	48000	57000	58000	58000
POST	search_ticket_expected	10083	10083	280	7448.99603081227	39.078813046217	72869.2890675738	0	42.1252804052808	42.1252804052808	280	660	18000	18000	23000	34000	56000	59000	71000	73000
	Aggregated	173044	94353	270	3423.04029540199	12.2524071484804	72869.2890675738	7898.02182103974	722.952198993496	394.192857490773	270	450	600	690	17000	22000	42000	51000	60000	69000

Saturation RPS : 100 - 120

loop Pattern

1. WE GENERATE LOW WORKLOAD

2. WE GET RESULTS

3. WE GENERATE HIGH WORKLOAD

4. SERVICES GOES DOWN

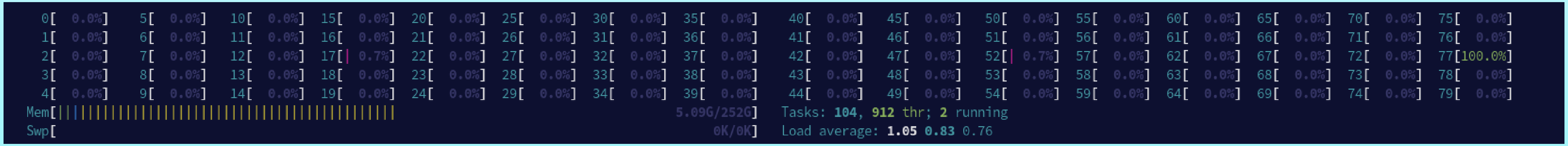
5. WAIT FOR FEW HOURS

REPEAT

Probable Problem

We are generating load from netx2 : 5000 request per second

We can see that the core number 77 is shooting to 100% during work load generation



```
  0[  0.0%]    5[  0.0%]   10[  0.0%]  15[  0.0%]  20[  0.0%]  25[  0.0%]  30[  0.0%]  35[  0.0%]    40[  0.0%]   45[  0.0%]   50[  0.0%]  55[  0.0%]  60[  0.0%]  65[  0.0%]  70[  0.0%]  75[  0.0%]
  1[  0.0%]    6[  0.0%]   11[  0.0%]  16[  0.0%]  21[  0.0%]  26[  0.0%]  31[  0.0%]  36[  0.0%]    41[  0.0%]   46[  0.0%]   51[  0.0%]  56[  0.0%]  61[  0.0%]  66[  0.0%]  71[  0.0%]  76[  0.0%]
  2[  0.0%]    7[  0.0%]   12[  0.0%]  17[| 0.7%]  22[  0.0%]  27[  0.0%]  32[  0.0%]  37[  0.0%]    42[  0.0%]   47[  0.0%]   52[| 0.7%]  57[  0.0%]  62[  0.0%]  67[  0.0%]  72[  0.0%]  77[100.0%]
  3[  0.0%]    8[  0.0%]   13[  0.0%]  18[  0.0%]  23[  0.0%]  28[  0.0%]  33[  0.0%]  38[  0.0%]    43[  0.0%]   48[  0.0%]   53[  0.0%]  58[  0.0%]  63[  0.0%]  68[  0.0%]  73[  0.0%]  78[  0.0%]
  4[  0.0%]    9[  0.0%]   14[  0.0%]  19[  0.0%]  24[  0.0%]  29[  0.0%]  34[  0.0%]  39[  0.0%]    44[  0.0%]   49[  0.0%]   54[  0.0%]  59[  0.0%]  64[  0.0%]  69[  0.0%]  74[  0.0%]  79[  0.0%]
Mem[|||||||||||||||||||||||||||||||||||||||||||||                                5.09G/252G]   Tasks: 104, 912 thr; 2 running
Swp[                                                                                 0K/0K]   Load average: 1.05 0.83 0.76
```

Our Finding : Every process except for a few runs on core 77

Further Work

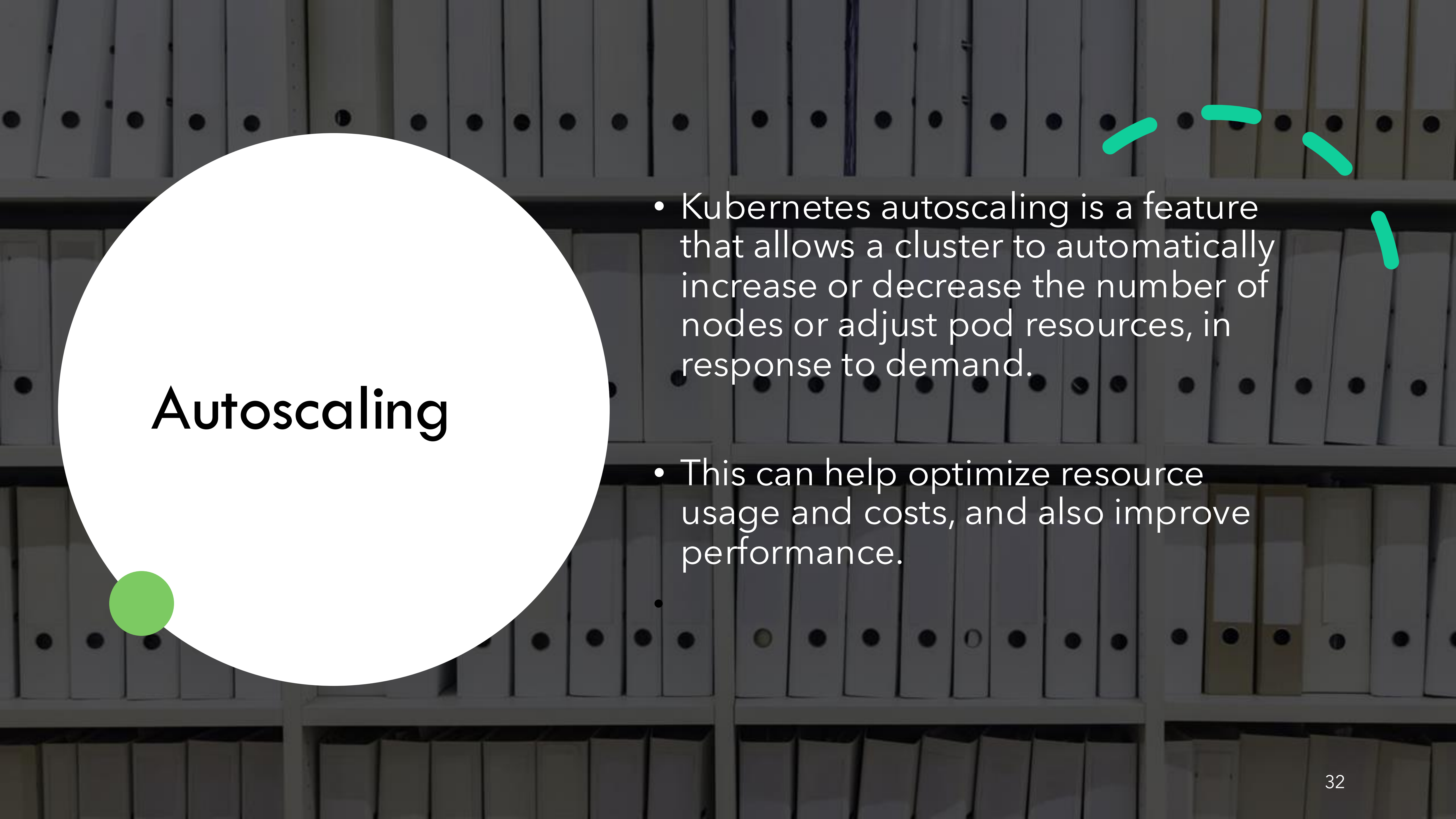
We have to generate **mixed workload** hitting all the services.

We have to enable **Perfmon monitoring** and do end-to-end testing again.

Then we have to update the deployment and enable **auto scaling** and look for the improvement.

Perfmon

- End host Monitoring tool for TCP.
- It **provide** us time spend on root namespace and container namespace.
- Helps us to debug the services like if there is delay why it is?
- Then we also have to study delay caused by Perfmon.

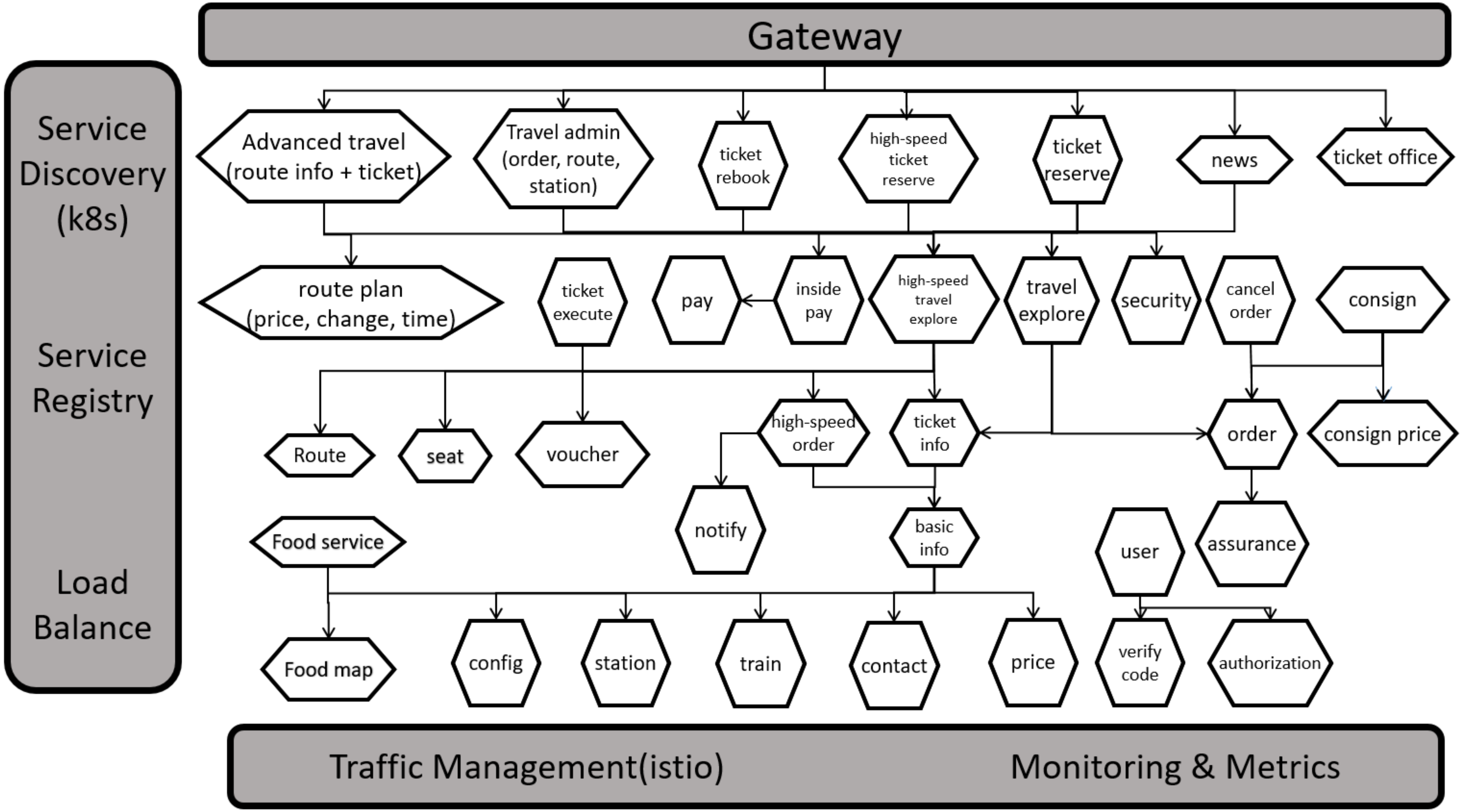


Autoscaling

- Kubernetes autoscaling is a feature that allows a cluster to automatically increase or decrease the number of nodes or adjust pod resources, in response to demand.
- This can help optimize resource usage and costs, and also improve performance.

Thank you

Services


Gateway
Service Discovery (k8s)
Service Registry
Load Balance
Advanced travel (route info + ticket)
Travel admin (order, route, station)
ticket rebook
high-speed ticket reserve
ticket reserve
news
ticket office
route plan (price, change, time)
ticket execute
pay
inside pay
high-speed travel explore
travel explore
security
cancel order
consign
Route
seat
voucher
high-speed order
ticket info
order
consign price
notify
basic info
user
assurance
Food service
Food map
config
station
train
contact
price
verify code
authorization
Traffic Management(istio)
Monitoring & Metrics