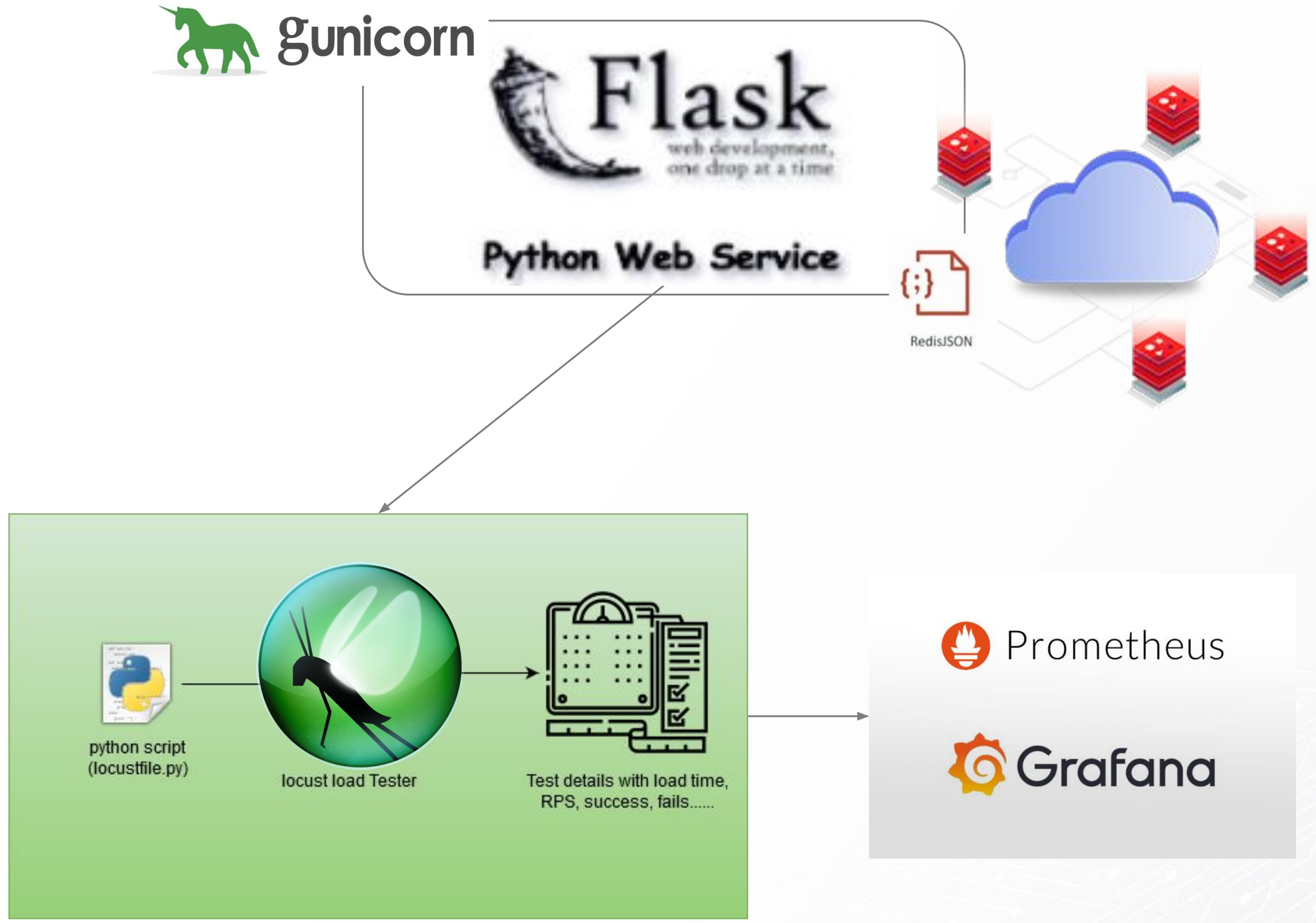
redislabs
HOME OF REDIS

RedisJSON + Locust

Load testing your Redis-enabled Services in a breeze

Demo Architecture


gunicorn
Flask
web development, one drop at a time
Python Web Service
RedisJSON
Prometheus
Grafana
python script (locustfile.py)
locust load Tester
Test details with load time, RPS, success, fails......

- **Flask REST API Services:**
  - RedisJSON 1.0 on Redis Cloud
  - Gunicorn
- **Locust Load testing:**
  - lightweight scripts
  - Easy to group/separate test set
  - Spawn rate
  - Event hook
  - Distributed
- **Visualization:**
  - cAdvisor: API container usage monitoring
  - Scrape some customized metrics from Locust.io via Prometheus
  - Visualize them all on Grafana

redislabs HOME OF REDIS

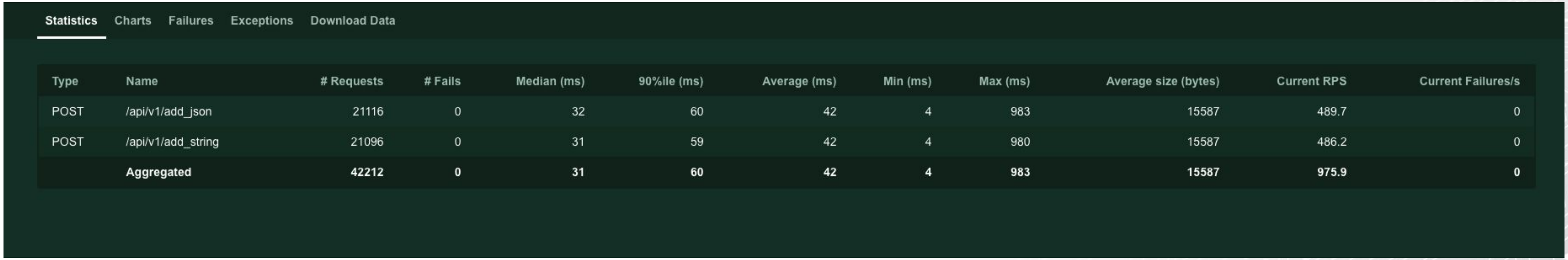
Configuration:

Locust simulation: 100k user; 10000 user/sec spawn rate;
1 master; 4 workers

App server: 32GB/8 cores VM; 20 workers, 2 threads per worker;

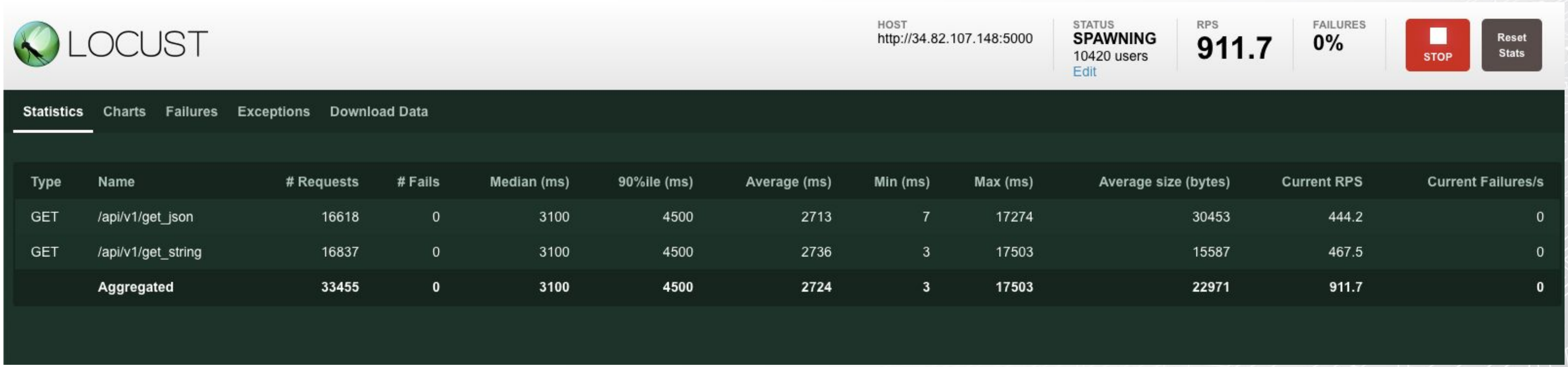
Sample JSON Object: 15K bytes pokemon log

POST 15k+ JSON object



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	/api/v1/add_json	21116	0	32	60	42	4	983	15587	489.7	0
POST	/api/v1/add_string	21096	0	31	59	42	4	980	15587	486.2	0
	Aggregated	**42212**	**0**	**31**	**60**	**42**	**4**	**983**	**15587**	**975.9**	**0**

GET the entire 15k JSON doc



http://34.82.107.148:5000

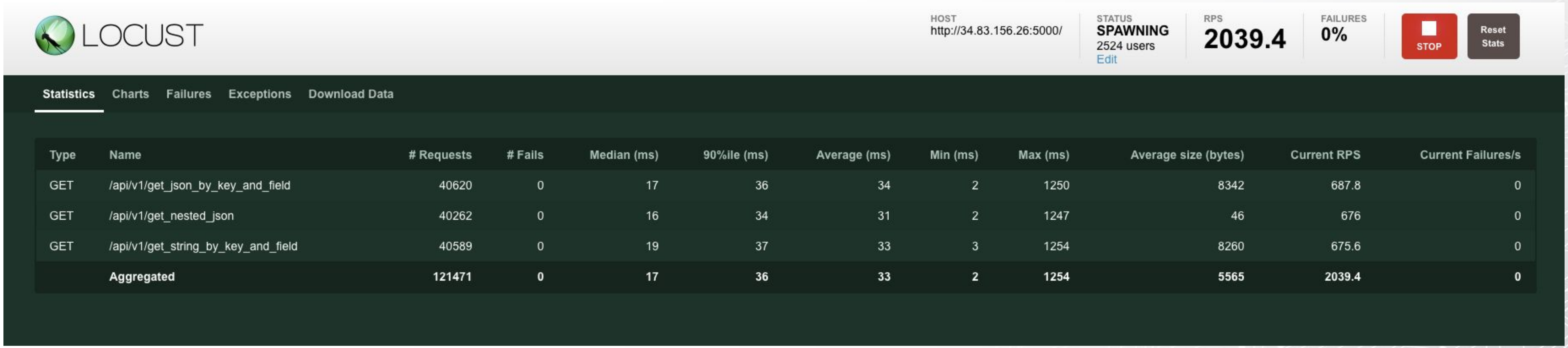
SPAWNING
10420 users

911.7

0%

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/api/v1/get_json	16618	0	3100	4500	2713	7	17274	30453	444.2	0
GET	/api/v1/get_string	16837	0	3100	4500	2736	3	17503	15587	467.5	0
	Aggregated	**33455**	**0**	**3100**	**4500**	**2724**	**3**	**17503**	**22971**	**911.7**	**0**

GET a nested field on 15k JSON object



HOST http://34.83.156.26:5000/

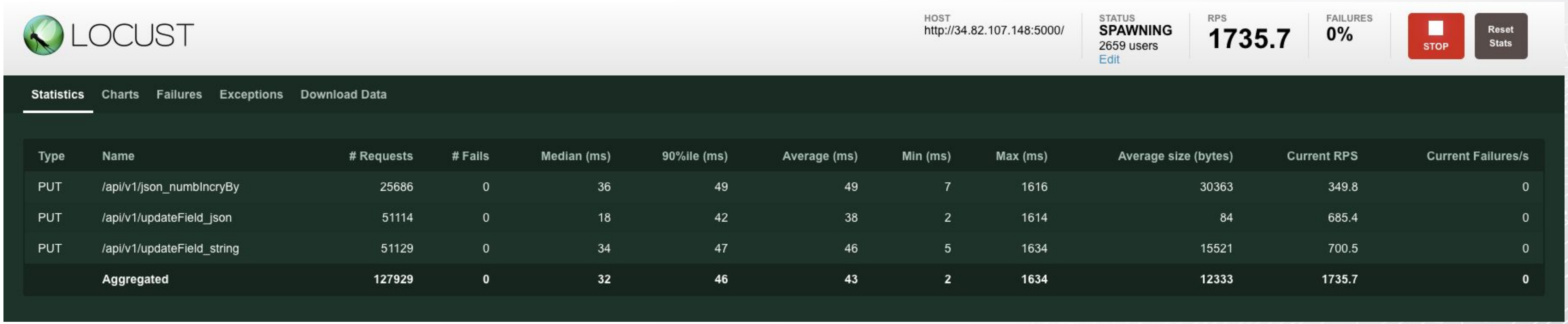
STATUS SPAWNING 2524 users

RPS 2039.4

FAILURES 0%

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/api/v1/get_json_by_key_and_field	40620	0	17	36	34	2	1250	8342	687.8	0
GET	/api/v1/get_nested_json	40262	0	16	34	31	2	1247	46	676	0
GET	/api/v1/get_string_by_key_and_field	40589	0	19	37	33	3	1254	8260	675.6	0
	Aggregated	**121471**	**0**	**17**	**36**	**33**	**2**	**1254**	**5565**	**2039.4**	**0**

UPDATE a nested field on 15K+ JSON Object



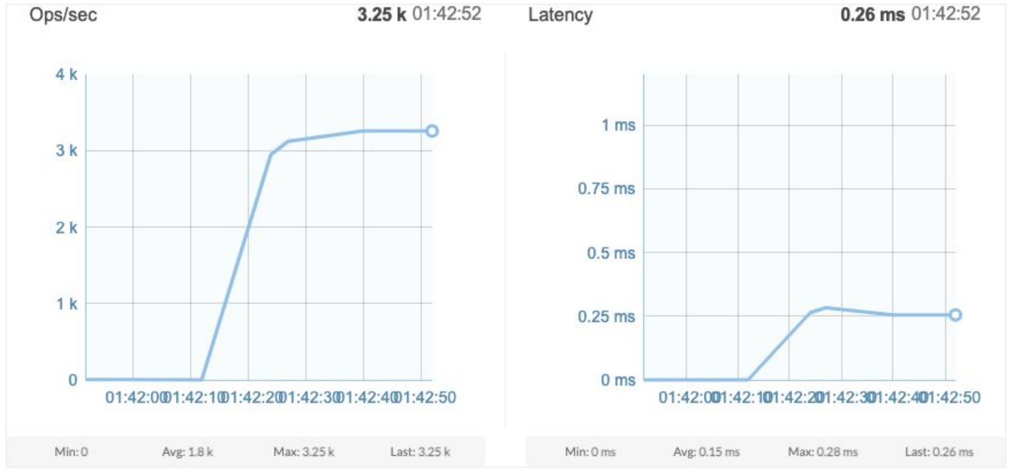
HOST
http://34.82.107.148:5000/

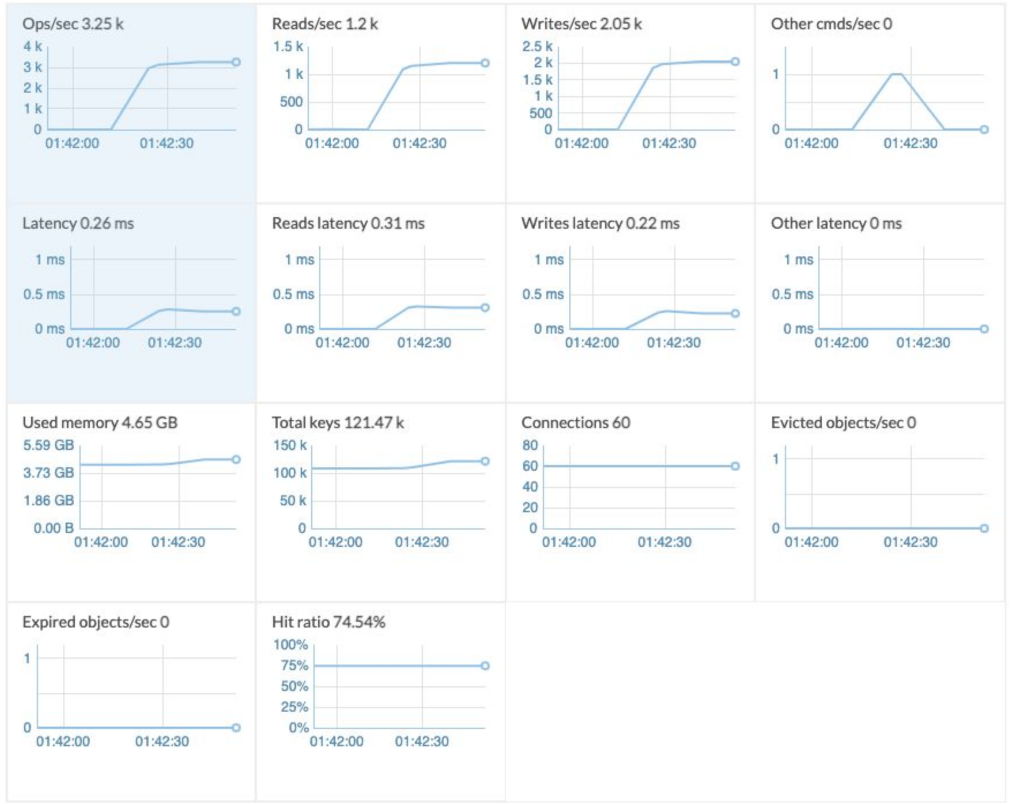
STATUS
SPAWNING
2659 users
Edit

RPS
1735.7

FAILURES
0%

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
PUT	/api/v1/json_numbIncryBy	25686	0	36	49	49	7	1616	30363	349.8	0
PUT	/api/v1/updateField_json	51114	0	18	42	38	2	1614	84	685.4	0
PUT	/api/v1/updateField_string	51129	0	34	47	46	5	1634	15521	700.5	0
	Aggregated	**127929**	**0**	**32**	**46**	**43**	**2**	**1634**	**12333**	**1735.7**	**0**


Ops/sec 3.25 k 01:42:52
Latency 0.26 ms 01:42:52
Min: 0 Avg: 1.8 k Max: 3.25 k Last: 3.25 k
Min: 0 ms Avg: 0.15 ms Max: 0.28 ms Last: 0.26 ms


Ops/sec 3.25 k
Reads/sec 1.2 k
Writes/sec 2.05 k
Other cmds/sec 0
Latency 0.26 ms
Reads latency 0.31 ms
Writes latency 0.22 ms
Other latency 0 ms
Used memory 4.65 GB
Total keys 121.47 k
Connections 60
Evicted objects/sec 0
Expired objects/sec 0
Hit ratio 74.54%

Why Load testing

In summary, load testing helps you to:

- Determine the **throughput** required to support the anticipated peak production load.
- Determine the **adequacy** of a hardware environment.
- Evaluate the adequacy of a load balancer.
- Detect **concurrency** issues.
- Detect **functionality errors under load.**
- Collect data for scalability and capacity-planning purposes.
- Help to determine **how many users** the application can handle before performance is compromised.
- Help to determine **how much load the hardware can handle** before resource utilization limits are exceeded.

Most importantly, show how powerful and reliable Redis/ Redis Modules are under stress testing

What's Locust

- **Define user behaviour in code**

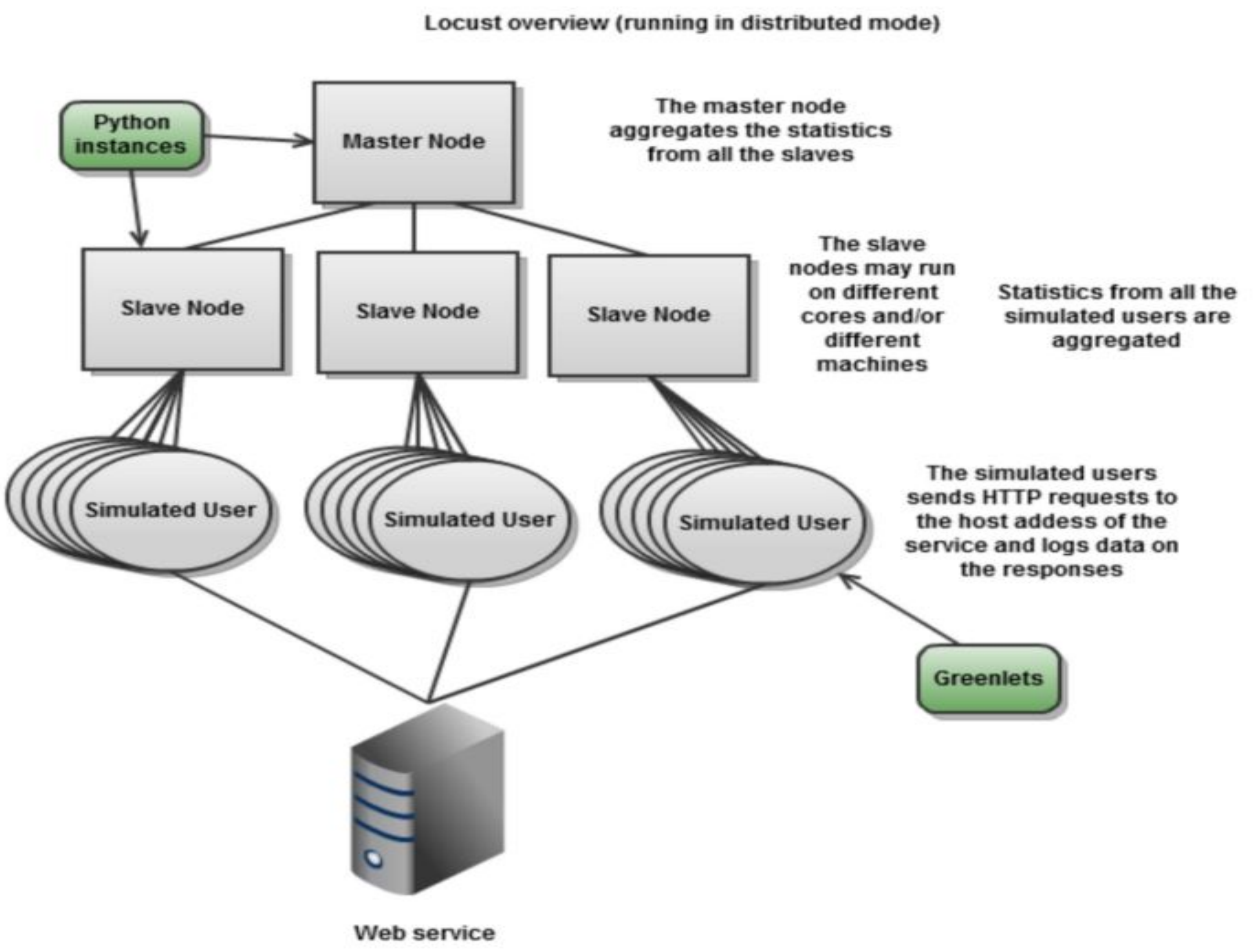
  No need for clunky UIs or bloated XML. Just plain code

```python
""" Build the TaskSet """
class testOnPost(TaskSet):
    @tag('add_random_small_json')
    @task(3)
    def add_random_small_json(self):
        json_doc = {
            'id':    "basicUserJson:" + str(uuid.uuid4()),
            'name': fake.name(),
            'age': fake.random_int(min=0, max=100),
            'location': str(fake.latitude()),
            'address': fake.street_address()
        }
        json_doc = json.dumps(json_doc)

        self.client.post('/api/v1/redisjson',
            data=json_doc,
            headers={'Content-Type': 'application/json'},
            timeout=50,
            name='/api/v1/add_random_small_json')
```

```python
class testOnPut(TaskSet):

    @tag('update_nested_field_json')
    @task(2)
    def update_field_nested(self):
        update = {
            'key': random.choice(AdvancedUserTestSet),
            'field': 'company.name',
            'str': 'Redis Lab'
        }
        self.client.put('/api/v1/redisjson/update',
            data=json.dumps(update),
            headers={'Content-Type': 'application/json'},
            timeout=50,
            name='/api/v1/update_field_nested')
```

redislabs HOME OF REDIS

What's Locust, and what can it do


Locust overview (running in distributed mode)
Python instances
Master Node
The master node aggregates the statistics from all the slaves
Slave Node
Slave Node
Slave Node
The slave nodes may run on different cores and/or different machines
Statistics from all the simulated users are aggregated
Simulated User
Simulated User
Simulated User
The simulated users sends HTTP requests to the host addess of the service and logs data on the responses
Greenlets
Web service

- **Distributed & scalable**

  Locust supports running load tests distributed over multiple machines, and can therefore be used to simulate millions of simultaneous users

source:
https://akshaye-ks.medium.com/locust-an-easy-distributed-load-testing-framework-a4d8aaa9c245

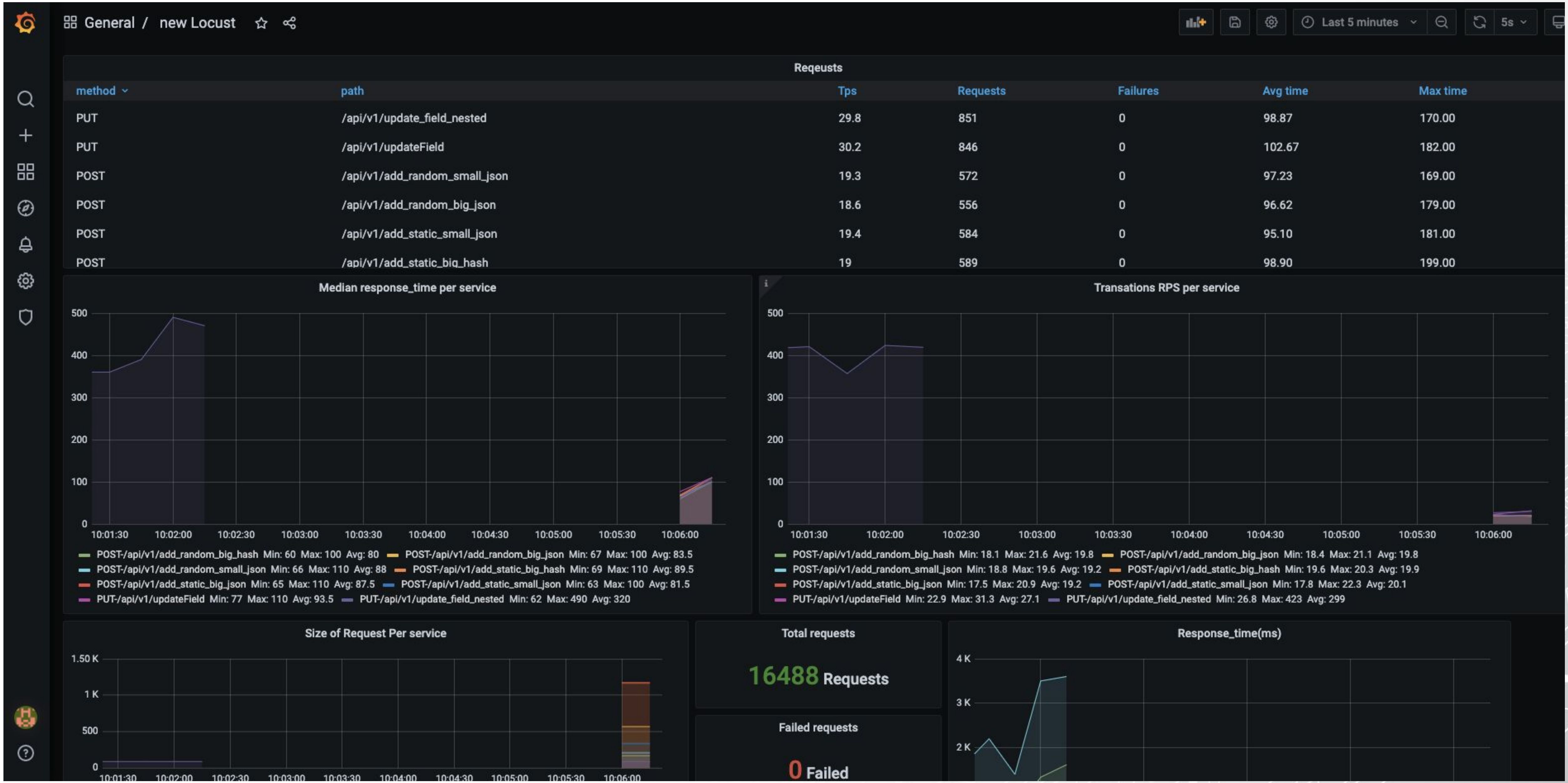
Bonus: How to tune a production-ready Flask app with Gunicorn

Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX. It's a pre-fork worker model. The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy.

https://medium.com/building-the-system/gunicorn-3-means-of-concurrency-efbb547674b7

Take away & Lesson learned & More to come

1. Tuning your web application is crucial
2. Find the right test cases for your audience
3. Monitor the resource usage and allocate resources efficiently
4. Terraform + Cloud deployment



Requests

method	path	Tps	Requests	Failures	Avg time	Max time
PUT	/api/v1/update_field_nested	29.8	851	0	98.87	170.00
PUT	/api/v1/updateField	30.2	846	0	102.67	182.00
POST	/api/v1/add_random_small_json	19.3	572	0	97.23	169.00
POST	/api/v1/add_random_big_json	18.6	556	0	96.62	179.00
POST	/api/v1/add_static_small_json	19.4	584	0	95.10	181.00
POST	/api/v1/add_static_big_hash	19	589	0	98.90	199.00

Median response_time per service

	Series	Min	Max	Avg
🟩	POST-/api/v1/add_random_big_hash	60	100	80
🟨	POST-/api/v1/add_random_big_json	67	100	83.5
🟦	POST-/api/v1/add_random_small_json	66	110	88
🟧	POST-/api/v1/add_static_big_hash	69	110	89.5
🟥	POST-/api/v1/add_static_big_json	65	110	87.5
🟦	POST-/api/v1/add_static_small_json	63	100	81.5
🟪	PUT-/api/v1/updateField	77	110	93.5
🟪	PUT-/api/v1/update_field_nested	62	490	320

Transations RPS per service

	Series	Min	Max	Avg
🟩	POST-/api/v1/add_random_big_hash	18.1	21.6	19.8
🟨	POST-/api/v1/add_random_big_json	18.4	21.1	19.8
🟦	POST-/api/v1/add_random_small_json	18.8	19.6	19.2
🟧	POST-/api/v1/add_static_big_hash	19.6	20.3	19.9
🟥	POST-/api/v1/add_static_big_json	17.5	20.9	19.2
🟦	POST-/api/v1/add_static_small_json	17.8	22.3	20.1
🟪	PUT-/api/v1/updateField	22.9	31.3	27.1
🟪	PUT-/api/v1/update_field_nested	26.8	423	299

Size of Request Per service

Total requests

16488 Requests

Failed requests

0 Failed

Response_time(ms)

redislabs HOME OF REDIS

Questions?

redislabs
HOME OF REDIS