

SISTEMAS OPERATIVOS 1

CLASE 5

LEONEL AGUILAR

SEBASTIÁN SÁNCHEZ

LOCUST

UTILIZANDO HERRAMIENTAS DE LOAD-TESTING

<https://github.com/leoagUILar97/so1-course/tree/main>



PROYECTO 1
Explicación, preguntas,
comentarios.

01

LOAD TESTING
¿Qué es? ¿Para qué?

02

LOCUST
Cómo probar y enviar
tráfico a nuestras APIs

03

¿QUÉ HAREMOS HOY?

04

EJEMPLO PRÁCTICO
Conectando Locust con
nuestra API de timestamps.

05

KAHOOT
Comprobando nuestro
conocimiento de la clase

06

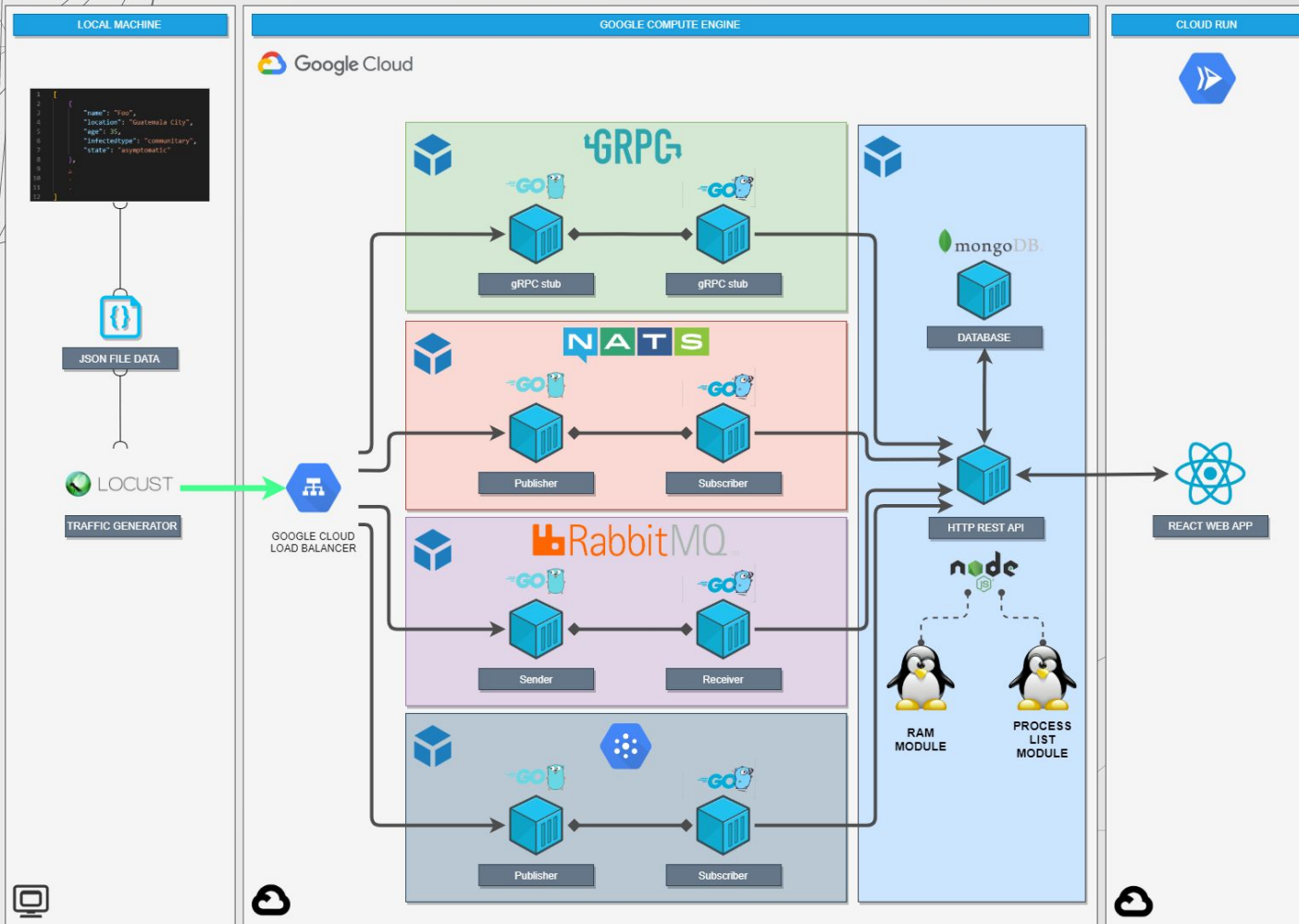
Q&A
Preguntas y comentarios
finales

01

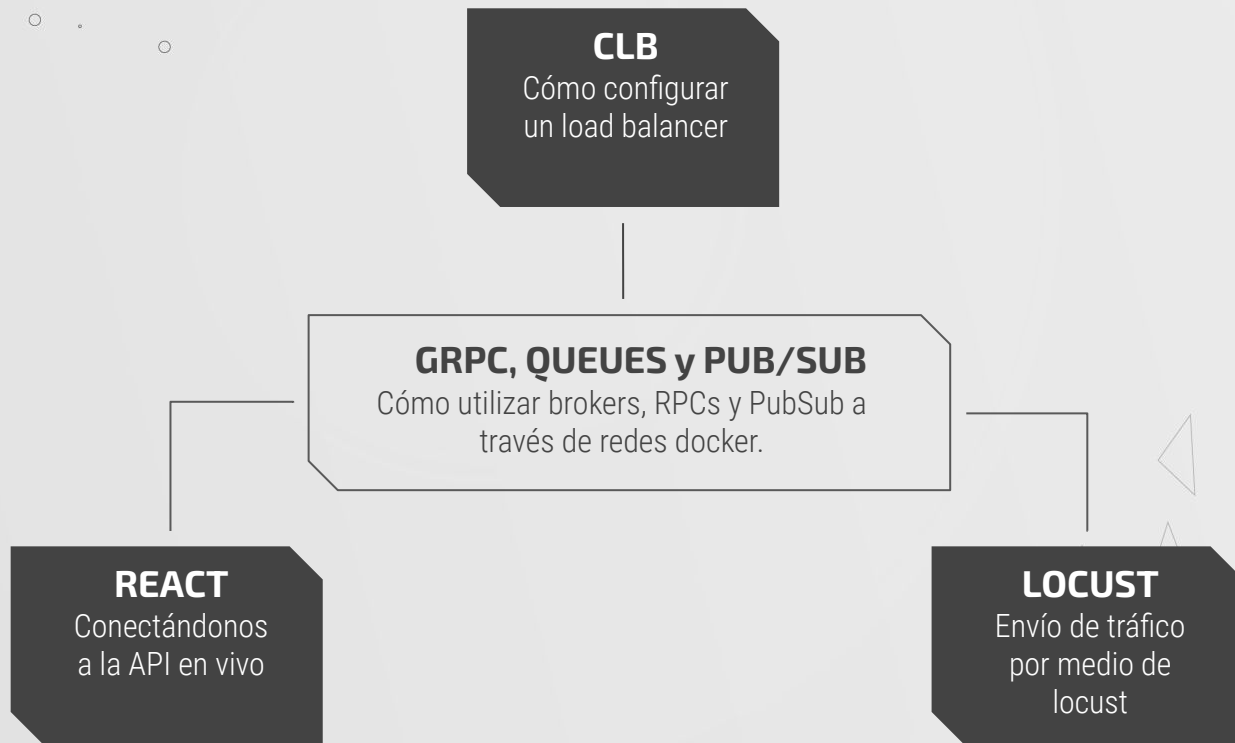
PROYECTO

Una breve explicación, preguntas y comentarios.





¿QUÉ NOS FALTA POR APRENDER?





02

LOCUST

Generando tráfico por la web...

LOAD TESTING



OBSERVAR

Observar cómo se comporta el sistema.

El ambiente de load testing debe ser diferente al de producción.

Analizar el flujo de tráfico de datos y respuestas

ANALIZAR



MEJORAR

Corregir problemas con la latencia, errores que pasaron desapercibidos, etc...



03

LOCUST

¿Qué podemos hacer con Locust?

UTILIZANDO LOCUST...

Utilizado como
framework o como
librería.

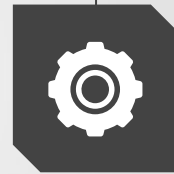
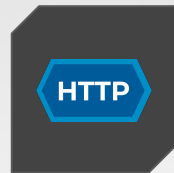
Utiliza la librería
request.py por lo
que es muy familiar.



Utilización con
Python 3.6+



Envío de tráfico a
través de pruebas
específicas



¡Completamente
configurable!



COMANDOS IMPORTANTES



```
# Laboratorio de sistemas operativos 1
```

```
# Instalar locust
```

```
$ pip3 install locust
```

```
# Revisando version
```

```
$ locust -V
```

```
# locust 1.4.3
```

```
# Corriendo un archivo de locust
```

```
$ locust --filename [nombre].py
```



CÓDIGO BÁSICO

```
# Importar HttpUser que es utilizado para crear una clase que pueda usar locust
# El decorador task nos permite definir las tareas del usuario
# between nos permite definir un rango de valores
from locust import HttpUser, task, between

# Creando una clase que derive de HttpUser podemos crear un USUARIO de locust
# Los usuarios ejecutan tareas (descritas con el decorador 'task')
# Las tareas ejecutan código cada vez que son llamadas, generalmente tienen llamadas a la API
# El tiempo de ejecución entre cada tarea depende de la variable wait_time
class SolTest(HttpUser):
    # definiendo wait time como una tarea por cada 0.1, 0.3 segundos
    wait_time = between(0.1, 0.3)


    # definiendo una tarea que realizara una petición http get /
    @task
    def test(self):
        self.client.get('/')

    # Definir una tarea que realizara una petición http post /
    def testPost(self):
        # mandar la petición post a la API
        self.client.post('/', { "data": "¡Hola Mundo!" })

    # Diciendo a locust que en este momento debemos parar las tareas que
    # esta realizando el usuario
    self.stop()
```

VISTA DE CONFIGURACIONES

<http://localhost:8089>



← → ↻ 🏠 localhost:8089 ... 🛡️ ☆

📶 LOCUST HOST STATUS
READY
0 users

Start new load test

Number of total users to simulate

de usuarios a crear

Spawn rate (users spawned/second)

Frecuencia con la que se crearán

Host (e.g. http://www.example.com)

URL de la API que probaremos

Start swarming

VISTA DE TAREAS



LOCUST

HOST
http://localhost:4000

STATUS
RUNNING
0 users
[Edit](#)

RPS
2.5

FAILURES
0%



Reset
Stats

[Statistics](#) [Charts](#) [Failures](#) [Exceptions](#) [Download Data](#)

USUARIOS EJECUTANDO
TAREAS ACTUALMENTE

PARAR LA PRUEBA
O REINICIARLA

PETICIONES QUE HA REALIZADO

TAREA

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	193	0	580	690	587	344	797	143112	1	0
POST	/	200	0	110	160	115	78	266	124	1.5	0
Aggregated		393	0	200	640	346	78	797	70345	2.5	0

04

EJEMPLO!

Conectando Locust con nuestras APIs





KAHOOT TIME



06

¿PREGUNTAS?