Trabajo práctico

75.73 Arquitectura de Software

Situación

La empresa en la que trabajan está planeando modernizar el producto que hacen. Este producto es un gran monolito (una única aplicación muy grande), que se viene haciendo hace muchos años y se volvió difícil de mantener.

La idea no es rehacerlo entero porque llevaría demasiado tiempo y sería muy riesgoso. En el corto plazo, lo que quieren hacer es separar unas pocas partes pequeñas de la gran aplicación a servicios externos, y hacer que la aplicación use esos servicios. Así, ven que cada servicio va a ser más fácil de mantener.

La gran duda ahora es cómo encarar estos nuevos servicios. ¿Qué tecnologías conviene usar? ¿Qué ventajas y desventajas tiene cada una? Por esto, les pidieron que analicen algunas alternativas de tecnologías y configuraciones:

Obligatorias:

- app flask + gunicorn web server (1 worker)
- app node/express (1 worker, sin usar el modo cluster)
- N instancias app (alguna de las anteriores) con load balancer

Opcionales:

- app flask + gunicorn web server (múltiples workers)
- app node/express en modo cluster, con múltiples subprocesos
- N instancias de la app, cada una con varios workers, y un load balancer delante

Ustedes deberán someter a distintos tipos de escenario de carga a un servicio de prueba que hagan en cada una de las configuraciones anteriores y medir diversos aspectos, para así tener resultados representativos de cada configuración en cada caso que les permitan entender las ventajas y desventajas de cada uno.

En función de los datos obtenidos tendrán analizarlos y confeccionar un informe que describa el trabajo realizado junto con los resultados del análisis. Este informe debe contener también gráficos que muestren una serie temporal de las mediciones realizadas y una descripción de las herramientas y escenarios de cargas ejecutados.
Para cada una de las configuraciones, el informe debe incluir un diagrama con una **vista de la Estructura de Deployment** de la arquitectura[1].

[1] *Nota: una Estructura de Deployment es una estructura de tipo Allocation que muestra los procesos que ejecutan el software, las entidades de hardware donde se ejecutan, y las relaciones y vías de comunicación entre ellos.*

Guías

Por un lado, quieren ver cómo se comporta cada tecnología ante endpoints sencillos, como estiman que terminarán siendo algunas de las funcionalidades básicas que tendrán los nuevos servicios. Para ello, y para empezar a familiarizarse con cada una, la idea es hacer un endpoint **rápido y liviano** (tipo “ping” u “hola mundo”).

Por otro lado, saben que Python tiene un modelo de procesamiento sincrónico mientras que Node tiene uno asincrónico, y sospechan que eso puede generar diferencias notables. Para poner la teoría a prueba, pueden probar cómo se comporta cada uno ante carga distintos tipos de endpoints:

- **Endpoint lento y liviano**. Estiman que con un simple sleep/delay pueden simular el impacto que tendría llamar a otro servicio externo (por ejemplo, una base de datos en otro servidor o simplemente otro app server que tenga otra parte de la funcionalidad), es decir que la respuesta final puede demorar pero consumen pocos recursos en ese nodo en procesarlo.
- **Endpoint lento y pesado**. Los endpoints que requieren cálculos más complejos o manipular muchos datos suelen demorar un poco y sí consumen muchos recursos (CPU y/o memoria). En particular, creen que el uso de CPU puede marcar una diferencia notable, con lo que tienen la teoría de que un simple ciclo que demore cierto tiempo sin descansar puede servirles para comparar.

Además, tienen algunas otras ideas que les resultaría interesante probar *(opcionales)*:

- **Contenido estático**. Endpoints que simplemente devuelvan archivos existentes, grandes o chicos
- **Passthrough a otro servicio**. Si bien el sleep/delay es una aproximación simple, les gustaría probar si efectivamente se comporta igual cuando llama a otro servicio.

Links y comentarios útiles

Node.js

- Página oficial: https://nodejs.org/en/. Pueden descargar e instalar node desde ahí. Sugerimos usar la última versión LTS (8.x), pero son libres de usar la que quieran.
- Aquellos que quieran poder manejar distintas versiones de node para distintos proyectos (más allá del tp), recomendamos instalarlo a través de nvm (node version manager): https://github.com/creationix/nvm
- Documentación: https://nodejs.org/dist/latest-v8.x/docs/api/
- Para empezar un proyecto, en una carpeta vacía ejecutar `npm init`
- Para instalar librerias, ejecutar dentro del proyecto `npm install <lib>`
- Para levantar el servidor, ejecutar `node <filename>`.
- Para crear múltiples subprocesos (opcional), ver el módulo `Cluster`: https://nodejs.org/dist/latest-v8.x/docs/api/cluster.html

Express

- Instalar: `npm install express`
- Documentación: http://expressjs.com/en/starter/hello-world.html

Python

- La mayor parte de los sistemas Unix suelen tenerlo instalado, pero de ser necesario, pueden descargarlo desde https://www.python.org/. Pueden usar la versión que quieran, sea tanto 2.x o 3.x.
- Sugerimos usar virtualenv para el proyecto de Python, para evitar que se mezclen las dependencias con otros proyectos de Python que puedan tener: https://virtualenv.pypa.io/en/stable/

Flask

- Instalar: `pip install flask`
- Documentación: http://flask.pocoo.org/docs/0.12/quickstart/#

Gunicorn

- Instalar: `pip install gunicorn`
- Documentación: http://docs.gunicorn.org/en/latest/index.html
- Para levantar el servidor, ejecutar `gunicorn app:<filename_sin_extensión> -w <cantidad_de_workers>`

Nginx

- Documentación: https://nginx.org/en/docs/

Generadores de carga

Hay muchas herramientas distintas, algunas posibilidades son:

- https://httpd.apache.org/docs/2.4/programs/ab.html
- https://www.npmjs.com/package/artillery
- https://jmeter.apache.org/
- https://www.npmjs.com/package/loadtest
- https://www.npmjs.com/package/autocannon
- https://github.com/rakyll/hey