Proyecto final

Materia: Sistemas Distribuidos
Término: 2017 II
Profesor: Cristina Abad

Número de integrantes: 1 a 3
Tema: Arquitectura de micro-servicios con caché para reducir latencia
Fecha entrega propuesta (proyectos autodefinidos): Segundo martes después de examen parcial
Fecha entrega proyecto: Martes 23 de enero, 8h00 (en mi oficina y vía SidWeb); a partir de las 8am puedo llamar a cualquier grupo a que presente su proyecto. Si un estudiante no está presente, tendrá automáticamente 0 en el proyecto. No atenderé a grupos que lleguen tarde.
Sistema operativo: Linux
Lenguaje de prog.: Java, C/C++, Clojure, Go, Python, C#

Descripción

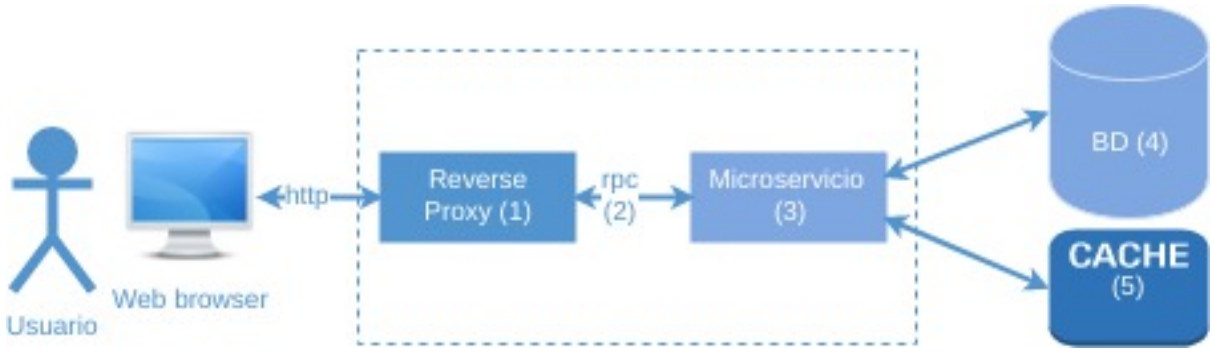
En el proyecto, deberán implementar un sistema web, con una arquitectura de microservicios y una caché para reducir la latencia de acceso a la base de datos. El sistema será uno que permite ver las noticias más populares de un periódico (top 10, en popularidad). Las noticias, con sus contadores de acceso, deberán estar almacenadas en una base de datos. El sistema web deberá contactar a un microservicio que le retorna las top 10 noticias, para su render en HTML al usuario final. Para evitar saturar la base de datos, deberán usar una caché en la cual almacenarán el resultado del query, por ejemplo:

```
SELECT  DISTINCT *
FROM    noticias
ORDER BY
      num_accesos
LIMIT 10
```

Es decir, cuando se invoca el microservicio, ustedes deben:

1) Verificar si el resultado del query ya está cargado en la caché (ej.: get <key> en Redis o Memcached, donde <key> puede ser un string que represente la fecha del día de hoy, dado que el top 10 de noticias debe cambiar cada día).
2) Si el resultado estaba en la caché, retornarlo al usuario.
3) Si el resultado no estaba en la caché, ejecutar el query SQL en la BD, almacenarlo en la caché (para que de ahora en adelante ya esté cacheado), y retornarlo al usuario.

El siguiente gráfico ilustra la arquitectura que deben implementar:



En la sección de Requerimientos se incluyen más detalles con respecto a la arquitectura; específicamente, se aclara lo que se espera con respecto a cada uno de los elementos numerados en la arquitectura.

Aquí hay enlaces a ejemplos con arquitecturas similares:

- http://2.bp.blogspot.com/-EbXflUYFwYM/T6p54bpMNXI/AAAAAAAABCk/cMlR7Cqki_k/s1600/Finagle+Diagram.png
- https://github.com/grpc-ecosystem/grpc-gateway
- https://www.cloudfoundry.org/scaling-real-time-apps-on-cloud-foundry-using-node-js-and-rabbitmq/

Requerimientos

- Indicaciones con respecto a los componentes de la arquitectura:
    1. Pueden escoger el Reverse Proxy / API Gateway que mejor les convenga. Por ejemplo, usar algo genérico como Nginx o algo más especializado como esto: https://github.com/grpc-ecosystem/grpc-gateway . Alternativamente, podrían tener un gateway con node.js, que se comunique con el microservicio directamente.
    2. El API de su microservicio debe ser **binario**, con un **RPC ligero**. Dos opciones comunes son: Apache Thrift o gRPC+protobuf.
    3. El microservicio puede estar implementado en cualquiera de los siguientes lenguajes: Java, C/C++, Clojure, Go, Python. **No se permiten otros lenguajes para la implementación de este componente**.
    4. Para la BD, sugiero usar un RDBMS tradicional, como MySQL, pero se permiten otros productos en caso de que así lo prefieran. La BD la deben cargar con algún dataset de noticias, al que ustedes le pueden añadir un campo de “número de accesos”, y, si el dataset no tiene esta información, ustedes lo pueden llenar con valores aleatorios. Aquí hay un dataset que pueden usar: (newsCorpora.csv en https://archive.ics.uci.edu/ml/datasets/News+Aggregator), o también este: http://mlg.ucd.ie/datasets/bbc.html

5. La caché debe ser Memcached o Redis. Tanto Memcached como Redis son “in-memory key-value stores”, por lo que para leer datos hay que asociarlos a una clave. Para este proyecto, la clave debe ser un string que represente el día en el que se hace la consulta. De esta manera, se cachearían las top 10 noticias **del día**.

- Una arquitectura alterna (en lugar de Nginx+grpc+microservicio) es utilizar Microsoft Orleans. Si optan por esta opción, el lenguaje de programación sería C#. Más detalles en: https://dotnet.github.io/orleans/Tutorials/index.html
- Utilizar Github. Enviarme el enlace al repositorio vía SidWeb, máximo hasta la semana después del examen parcial. **TODOS los miembros del grupo deben tener más de un commit en el repositorio; el primer commit de cada miembro debe tener una fecha máxima de Enero 4 de 2018. Necesito poder saber en qué trabajó cada uno.**
- Middlewares permitidos (en caso de que propongan su propio proyecto):
    1. RPC: gRPC o Apache Thrift ← Solamente para propuestas diferentes a la del servicio de compresión descrito abajo
    2. Publish/subscribe: Redis Pub/Sub, Apache ActiveMQ, Amazon SNS, Mosquitto (IoT)
    3. Message queues: Apache ActiveMQ, RabbitMQ, o Amazon SQS
- Deberán entregar un documento final (2 a 4 páginas, no más) en el que documenten el problema y solución propuesta y decisiones de diseño, así como resultados de los experimentos de rendimiento (latencia y throughput).
    1. Para las pruebas de rendimiento, deben generar pedidos HTML con alguna herramienta. Esto puede ser alguna herramienta que descarguen de Internet, o ustedes mismos programar algo en Python o Bash.
    2. La idea es que hagan dos pruebas: El rendimiento del microservicio usando solamente la BD vs. el rendimiento del microservicio usando la caché+BD. La idea es poder observar que al usar la caché, el rendimiento mejora.
    3. Para cada uno de los dos tipos de experimentos, deben realizar al menos 10 pruebas, y mostrar los resultados usando un box plot (https://en.wikipedia.org/wiki/Box_plot).
- Deployment en la nube: Puede ser en Azure o AWS. AWS otorga créditos gratuitos para estudiantes. Alternativamente, búsquenme en mi oficina y les puedo dar credenciales de acceso a AWS, usando unos créditos que tengo yo. En AWS, el servicio que deben usar es el Elastic Compute Cloud (EC2). **IMPORTANTE:** ESPOL tiene bloqueado el acceso al puerto 22, necesario para hacer SSH a instancias EC2. Esta parte la deben hacer desde su casa. El demo lo podrán mostrar sin problema, ya que ustedes se conectarán al servicio vía HTTP, lo cual no está bloqueado.

Alternativas para el proyecto final

- Auto-definido: Presentar propuesta.
- Investigación: Conversar conmigo si quieren colaborar en un proyecto de investigación.

Rúbricas

Será calificado sobre 30. La nota irá a calificación de proyecto final (2do parcial) **Y** del examen de mejoramiento. Para mejoramiento, es posible presentar un proyecto mejorado, en caso de que así lo deseen. La fecha máxima de entrega para el mejoramiento será el viernes de la semana de exámenes de mejoramiento, entre 8 y 9am, en mi oficina.

- Documento final: 2 puntos
    - Sección de metodología describe y justifica correctamente las decisiones de diseño del proyecto (¿Estructuras de datos usados? ¿Librerías o middlewares utilizados? ¿Manejo de errores? ¿Mecanismos de sincronización usados? ¿Cómo realizaron las pruebas de rendimiento? ¿Qué nube usaron? ¿Otras cosas importantes?): 2 puntos
- Implementación: 28 puntos
    - Web frontend: 2 puntos
    - Microservicio con uso correcto de RPC con formato binario: 6 puntos
    - Acceso correcto a BD: 2 puntos
    - Acceso correcto a Caché, incluyendo inserciones en la caché, cuando no se encuentra lo que se busca: 6 puntos
    - Se ha incluido un Makefile que permite compilar el proyecto en la línea de comando + README que explique lo necesario para compilar/instalar el SW: 1 punto
    - Deployment en la nube: 5 puntos.
    - Pruebas de rendimiento correctamente realizadas y correctamente documentadas (incluye el uso de una herramienta de benchmarking, o la implementación de scripts de prueba): 6 puntos
    - EXTRAS: Hasta 5 puntos, dependiendo de lo implementado.
- Participación en el proyecto: 10 puntos. Esta participación la evaluaré de dos maneras: (1) en la sustentación del proyecto, y (2) vía el log de commits del estudiante en el repositorio del proyecto. Si un estudiante no asiste a la sustentación o no tiene commits en el repositorio Git del proyecto immediatamente tendrá cero (0) en la nota del proyecto.
    - Estudiante demuestra haber participado activamente en el desarrollo del proyecto y demuestra entender su implementación y resultados: 10 puntos
    - Estudiante demuestra haber participado parcialmente el desarrollo del proyecto y demuestra entender parcialmente su implementación y resultados: 7.5 puntos
    - Estudiante demuestra participación limitada en el proyecto y/o no logra comprender los resultados obtenidos: 5 puntos
    - Estudiante demuestra una colaboración mínima en el desarrollo del proyecto: 2.5 puntos
    - Estudiante no colaboró en el desarrollo del proyecto: 0 puntos
- Nota Final: (Nota documento + nota implementación) * (Nota participación en proyecto)/10