White Paper Report

Response Caching in CITI-SENSE to improve WFS query performance

June 2016

Summary

CITI-SENSE is a European project, that set out to empower citizens through Citizens' Observatories, enabling them to contribute to and participate in environmental governance. Citizens will be able to support and influence community and societal priorities as well as other related decision making processes.

CITI-SENSE started in October 2012 and runs for a period of four years, completing at the end of September 2016. Citizens from 9 cities participated in the project forming around 20 citizen observatories in Barcelona, Belgrade, Edinburgh, Haifa, Ljubljana, Oslo, Ostrava, Vienna and Vitoria. Static sensors (fixed sensors) and mobile sensors (personal sensors) were deployed to monitor various environmental components of air quality.

Snowflake Software's contribution within CITI-SENSE was to explore through the project, how best to design and implement a solution to host both sensor and questionnaire information, allowing end users to access that same information over the internet in a secure method, and ensure the Spatial and Environmental Data Services (SEDS) Platform was performant under heavy loads, in a maintained environment. This whitepaper focuses on the issues surrounding cloud hosted platforms which need to be performant, the solutions available and those implemented in the CITI-SENSE project.

SEDS Platform

The SEDS Platform created by Snowflake Software during CITI-SENSE project provides the core interface between data providers (those static sensors or personal sensors providing data or people completing questionnaires) and data consumers (e.g. applications developed against the platform using the raw data for analysis). The SEDS platform allows data providers to upload data collected from all sensors into a centralised data store. It also allows data consumers to access this data using open standards (WFS and REST) web interfaces.

So what's the problem?

Handling data transfers over the internet from a technical implementation perspective can be challenging depending on the volume of data and the number of data requests. Snowflake Software specialise in Web Feature Services (WFS) an open, platform independent web service (request/response) that allows the retrieval of features contained within a remote data store (in this case, a cloud hosted database).

WFS supports a wide range of use cases, from data exchange where users can download data on request, through to supporting decision makers, for example when web applications are developed against a WFS to deliver value-add information to third party users.

During the CITI-SENSE project, a WFS was used to load data from sensor providers into a cloud hosted database. Two WFS services were also set up to facilitate end user request/responses allowing requests to the database that returned data in XML format. The following considerations were identified during the design of the platform to ensure the system is performant; high data volumes over the internet; number of requests made during a certain time frame; speed of response required for third party applications. In order to maintain a quality level of performance, WFS request response caching was implemented, during the CITI-SENSE project.

What is Response Caching?

Response caching is a technique for speeding up the response times for frequently used web requests. A cache sits between a client (such as a widget) and the server (the origin of the data). Caching can be a fairly complex topic as there are several options to store the cached content. In the simplest form the cached content can be stored on the server side or the client side. The technique can be used for anything from caching frequently used webpages to caching response data in XML for WFS requests (as is the case of CITI-SENSE).

Why would you need caching?

When a browser sends a HTTP request to a server, the request needs to be transmitted and received. The server then needs to compute the response and deliver it back to the browser and finally the browser needs to render the response. The process may take anywhere from fractions of a second up to several seconds to complete.

Each of the steps described above can add a time delay. There could be delays due to the network travel time (the time taken for the request to travel from the browser to the server, and back from the server to the browser), or the processing time due to the complexity of the request. By caching the response, the delay can be reduced or eliminated altogether.

Caching options

There are four options for caching the response.

1) Use client cache and do not contact the server at all – Caching on the client side is the fastest in terms of response time. This works best for static content which gets updated very rarely or not at all or in situations where serving out stale data has non-

critical consequences. The downside of this approach is that the server data can be updated without client awareness.

2) Use client cache with server verification – With this approach, the client contacts the server and verifies whether the client cache is current. If the cache is not current, then the server transmits the full response to the client. The downside of this approach is that both the client and the server need to retain cached copies of the response.
3) Use server side caching – This approach is used when caching on the client side is not possible. This approach creates less load on the server as the server has to handle fewer requests.
4) No cache – Generate a new response with each request. There is no caching that needs to be configured but if the request is complex then the response might take a long time.

Cache considerations

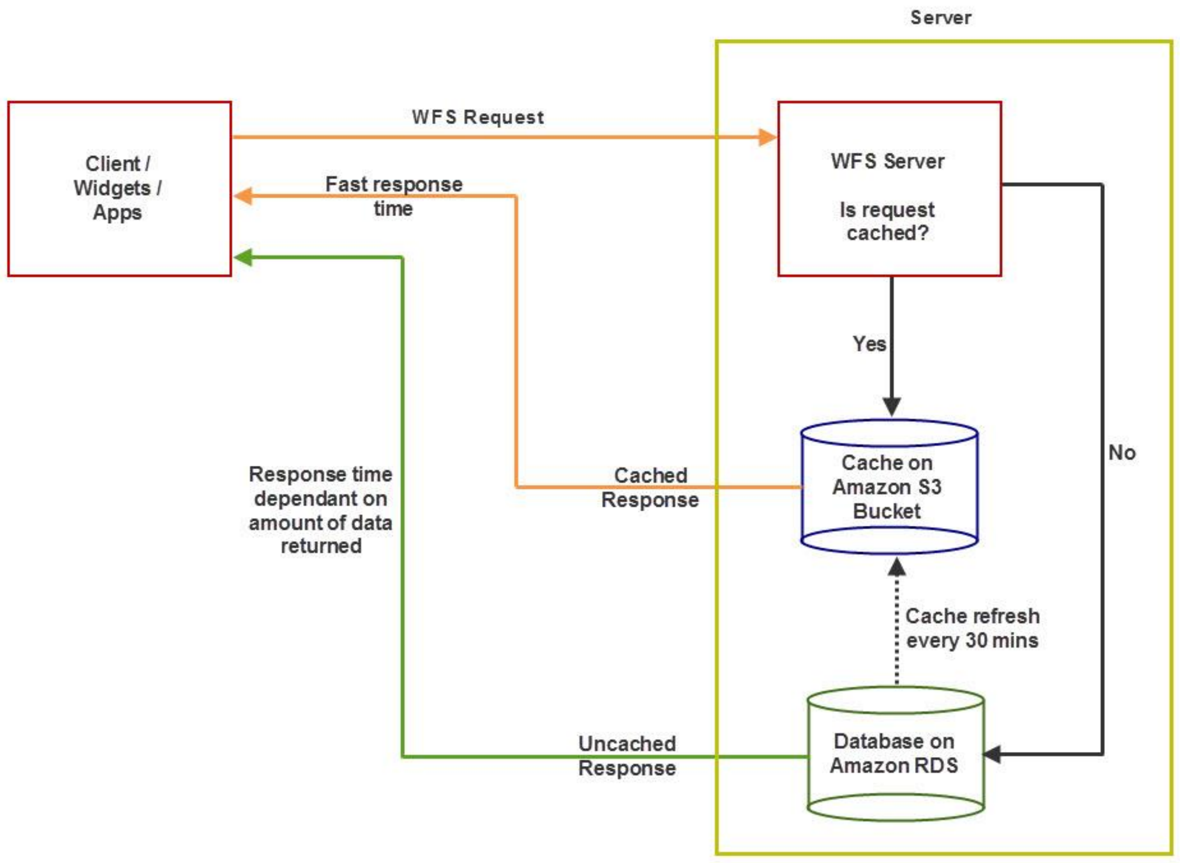
There are a couple of important factors that need to be considered when thinking about caching.

a) Cache lifespan – It is important to know how often the resource to be cached is updated and how often the resource is viewed, as well as what is the acceptable level of data staleness for the resource in question.
b) Memory limits – Server side caching uses memory for every request that is cached. With memory being a limited resource, there can be a limit to the amount of data that can be cached.

Caching in CITI-SENSE

The server side caching approach has been used for the CITI-SENSE project because it yielded substantial performance benefits. It helped reduce the load on the WFS server. 8 WFS requests have their responses cached. These are bounding box requests for 8 pilot cities that retrieve the latest observations from all sensors in those cities. These bounding box requests are the most commonly used requests. The widgets generate these WFS requests when users select the name of a city, to view data for, within the widget.

The response for each WFS request is saved as an XML file and these files are uploaded to an Amazon S3 bucket. The Amazon S3 bucket acts as the cache. Each of the 8 WFS requests gets redirected to the corresponding XML file on Amazon S3 bucket using Nginx configuration. The lifespan for the cache is 30 minutes, i.e. the XML files on the Amazon S3 bucket are refreshed every 30 mins. The diagram below explains the high level architecture used in the server side caching approach.



Result

Using server side caching reduced the response times for the cached WFS requests to under 2 seconds for requests that would take 20-30 secs without caching, ensuring any third-party widgets and/ or applications performed efficiently.

Summary

Through the CITI-SENSE project Snowflake Software explored the use of several caching methods in order to reduce the impact of complex WFS requests on performance of the SEDS platform. The main purpose of which was to reduce impact on widget and applications developed against the WFS component of the SEDS platform. Our findings are that server side caching should be used if the resource being cached changes frequently, and it is not acceptable to serve stale content. Client caching is best used in scenarios where content is rarely updated or serving stale content is acceptable. Response caching can deliver excellent performance gains when configured correctly.