SAP

Agile Developer

Spring 2017 Work Term #2

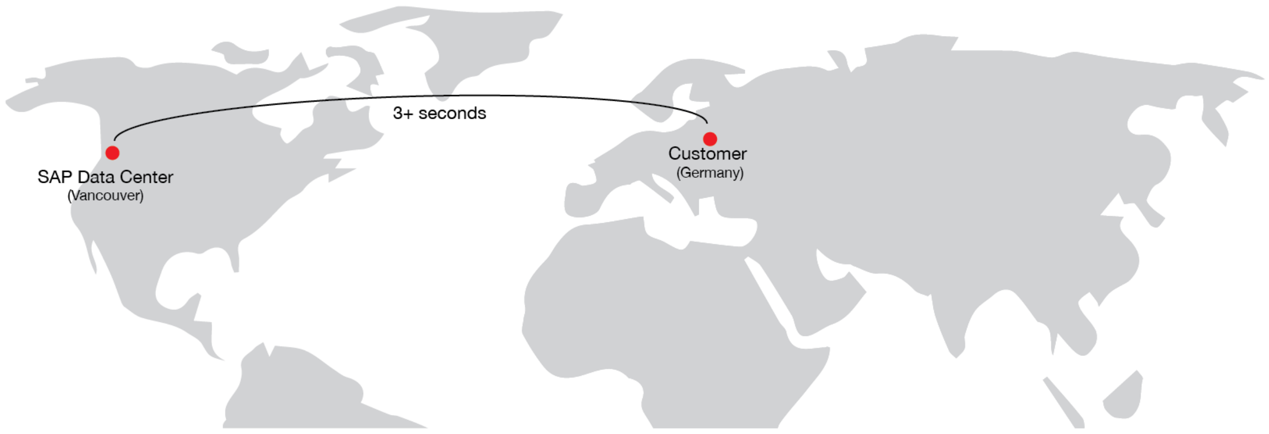
Technical Work Report

July 29, 2017

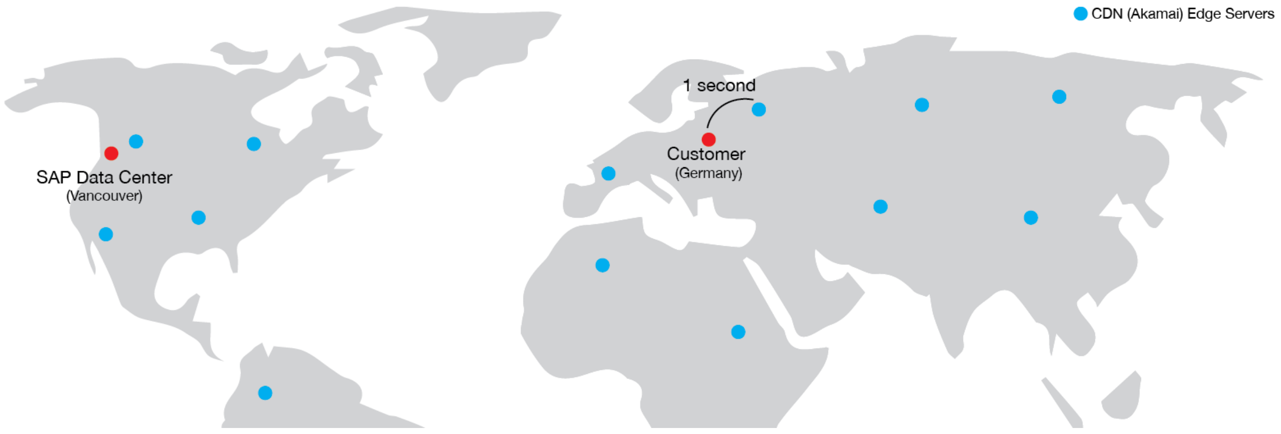
This is my second term working at SAP in the Vancouver office as an Agile Developer. SAP is a software company and one of the largest vendors of ERP (Enterprise Resource Planning). ERP is an efficient way to collect and organize data from various sectors of a company and analyze the performance in real time. I have been working on SAC (SAP Analytics Cloud) which allows for modelling, creating stories, data visualization and predictive capabilities. I have been introduced to many new technologies and technical applications throughout my time at SAP. In particular, this report will focus on a feature I was working on with an intermediate and senior developer. Our task was to move all of our static assets (JavaScript files, images, fonts etc.) to a Content Delivery Network (CDN).

A CDN is a group of servers that are geographically located across the global to deliver web pages and information to the users while maintaining high availability and performance. Our job was to get SAP Analytics Cloud (SAC) ready so that all of the static assets could be moved to the Akamai (one of the leading CDN providers) server in order to host our files so they can be replicated all over their edges servers around the world to increase the performance of our product. The diagram below illustrates how a Content Delivery Network works before and after implementation. For example, if all of our files are located in Vancouver and we have consumers using SAC in Germany then they their request would have to come all the way to Vancouver to retrieve these files and send them back to the consumer to display the web page. Network traffic could also lead to increased loading time for the consumer to use SAC. The Akamai CDN would take our files and replicate our static files used for SAC and cache the content to their own edge servers around the globe. With the CDN in place then the consumer in Germany would not have to fetch the data from Vancouver but instead can retrieve the data from the closest edge server to where the consumer is located. This will in turn improve performance of SAP Analytics Cloud. If content cannot be retrieved from the closest edge server for any reason then it will redirect them to the next closest server. This always keeps our product available for the end user and results in zero down time. CDN's are widely used in the tech industry by companies such as YouTube, Google and Facebook in order to deliver the end user the best possible experience when using their service.

Prior to CDN Implementation



After CDN Implementation



I was fortunate enough to be a part of such a large scale project for SAC and had various tasks to complete throughout the 3 months I contributed to this. A few of my tasks included pre/post Akamai performance testing, displaying an error page when our files cannot be accessed, and implementing a HTTP Server in order to host our files locally so we can still use our local development machine once our files are hosted in Akamai permanently and removed from our HANA database. Implementing the HTTP Server was definitely the most challenging task I've had to date. I had to integrate the server in order to serve up the static assets during our builds and run it asynchronously in order to not affect the build process. This was difficult but I learned how to implement it using Maven. Maven is a build automation tool which define how our files get compiled during our builds. Our XML (Extensible Markup Language) files define how our project gets built, dependencies used and the plugins. I had to create the HTTP Server using Node.js and start it by implementing XML in our POM (Project Object Model) files. These POM files are done in XML are how Maven projects are configured. I had to spawn off a separate process to start the HTTP Server which could then be accessed through our local IP address. I also had to not include this by default as not everyone would need this configuration so I created an environment variable that can be set to 'true' when we want this serve up our files locally. I also needed a way to stop the server once it was running so I implemented a Node Inter Process Communication (IPC) to listen when the server is up and know shut the server down when it receives a request to. I implemented a few requests that could be done by executing them in the terminal including to start the server, stop the server and to enable CORS (Cross Origin Resource Sharing) which would allow us to request resources (ex. fonts) from another domain that's different from where the request is coming from. This task of mine took roughly 2 weeks working on and off (as I got pulled on this task from time to time to work on other issues and responsibilities). Throughout this task I needed to learn how our builds works, using Maven/POM files, learning Node.js, estimating & time-boxing my work, and asking smart questions if an issue arose.

Another technical term I learned about while working on the Content Delivery Network is cache-busting. When browsers download files (ex. test.js) for the first time it can be cached in the browser for long periods of time (sometime even up to one year). Even if that file is updated the browser will still use the previous file already cached to load up the webpage. This is problematic as the browser doesn't know when there's a new version of test.js so the consumer will not see the updated web page. Cache-busting is a way around this problem to tell the browser not to use the cached test.js file but retrieve the new version of test.js that's available. This is done by versioning the file name with a hash key. This has key is a unique identifier appended to the file name to tell the browser that this file is different from the one that's cached. Whenever a file gets updated by a developer and ready for the public you can generate a random hash (ex. abcd1234) and append it to the file (ex. test.abcd1234.js). When this gets sent out, the browser will see this and notice the file names do not match and will retrieve this file and cache it until a newer file arrives.

With the CDN implemented our loading times have decreased. From our initial point of login, my pre Akamai CDN test was around 9-10 seconds for all of our chunks to load and navigation could start and my post Akamai CDN showed 6-7 seconds. With the inclusion of HTTP/2 (a network protocol focusing on performance and decreasing latency) our product will have even better performance. Overall, this was a very enjoyable and rewarding experience working closely with intermediate developer (Jason Park) and senior developer (Forrest Petrie) and contributing to the success of SAP Analytics Cloud.