# Salesforce Architecture - How they Handle 1.3 Billion Transactions a Day

Monday, September 23, 2013 at 8:44AM

*This is a guest post written by Claude Johnson, a Lead Site Reliability Engineer at salesforce.com.*

The following is an architectural overview of salesforce.com's core platform and applications. Other systems such as Heroku's Dyno architecture or the subsystems of other products such as work.com and do.com are specifically not covered by this material, although database.com is. The idea is to share with the technology community some insight about how salesforce.com does what it does. Any mistakes or omissions are mine.

This is by no means comprehensive but if there is interest, the author would be happy to tackle other areas of how salesforce.com works. Salesforce.com is interested in being more open with the technology communities that we have not previously interacted with. Here's to the start of "Opening the Kimono" about how we work.

Since 1999, salesforce.com has been singularly focused on building technologies for business that are delivered over the Internet, displacing traditional enterprise software. Our customers pay via monthly subscription to access our services anywhere, anytime through a web browser. We hope this exploration of the core salesforce.com architecture will be the first of many contributions to the community.

## Definitions

Let's start with some basic salesforce.com terminology:

- **Instance** - a complete set of systems, network and storage infrastructure, both shared and non-shared, that provides the salesforce.com service to a subset of our customers. For example, na14.salesforce.com is an instance.
- **Superpod** - a set of systems, network and storage infrastructure, including outbound proxy servers, load balancers, mail servers, SAN fabric and other infrastructure supporting multiple instances. Superpods provide service isolation within a datacenter so that problems with shared or complex components cannot impact every instance in that datacenter.
- **Org** (a.k.a., organization) - a single customer of the Salesforce application. Every trial started on www.salesforce.com or developer.force.com spawns a new org. An org is highly customizable and can have distinct security settings, record visibility and sharing settings, UI look and feel, workflows, triggers, custom objects, custom fields on standard salesforce.com CRM objects, and even custom REST APIs. An org can

support anywhere from one to millions of licensed individual user accounts, portal user accounts and Force.com Sites user accounts.
- **Sandbox** - an instance of the salesforce.com service that hosts full copies of production orgs for customer application development purposes. Customers using our platform can have full application development lifecycles. These are test environments for customers to do user acceptance testing against their applications before deploying changes into their production org.

## Stats (as of August 2013)

- 17 North America instances, 4 EMEA instances and 2 APAC instances
- 20 sandbox instances
- 1,300,000,000+ daily transactions
- 24,000 database transactions per second at peak (equivalent to a page view on other sites)
- 15,000+ hardware systems
- > 22 PB of raw SAN storage capacity
- > 5K SAN ports

## Software Technologies Employed

- Linux for development and primary production systems
- Solaris 10 w/ ZFS
- Jetty
- Solr
- Memcache
- Apache QPID
- QFS
- Puppet, Razor
- Perl, Python
- Nagios
- Perforce, Git, Subversion

## Hardware/Software Architecture

### Logging in to the salesforce.com service

We maintain a pool of servers to handle login traffic for all instances. A handful of servers from many (but not all) instances accept login requests and redirect the session to the user's home instance. This is what happens when you log in via login.salesforce.com.

Customer traffic starts with our external DNS. Once a lookup has successfully returned the IP address for an instance, standard Internet routing directs it to the appropriate datacenter.

Once the traffic enters our network in that datacenter, it is directed to the load balancer pair on which that IP lives. All of our Internet-facing IPs are VIPs configured on an active/standby pair of load balancers.

## Inside the instance

The load balancer directs the traffic to the application tier of the given instance. At this tier, we service both standard web page traffic as well as our API traffic. API traffic makes up over 60% of the traffic serviced by our application tier overall. Depending on the needs of the customer's request, it will be directed to additional server tiers for various types of backend processing.

## Core app

The core app tier contains anywhere from ten to 40 app servers, depending on the instance. Each server runs a single Hotspot JVM configured with as much as a 14 GiB heap, depending on the server hardware configuration.

The batch server is responsible for running scheduled, automated processes on the database tier. For example, the Weekly Export process which is used to export customer data in a single archive file format as a form of backup.

Salesforce.com offers a number of services including basic and advanced content management. We have a content search server and a content batch server for managing asynchronous processes on the content application tier. The content batch servers schedule processing of content types, including functions such as rendering previews of certain file types and file type conversion.

## Database

The primary data flow occurs between the core app server tier and the database tier. From a software perspective, everything goes through the database so database performance is critical. Each primary instance (e.g. NA, AP or EU instances) uses an 8 node clustered database tier. Customer sandbox (e.g. CS instances) have a 4 node clustered database tier.

Since salesforce.com is such a heavily database-driven system, reducing load on the database is critically important. To reduce load on the database tier, we developed ACS -- API Cursor Server. This was a solution to 2 problems which enabled us to improve our core database performance significantly. First, we used to store cursors in the database but the deletes were impacting performance. Second, after moving to using database tables to hold cursors, the DDL overhead became a negative impact. Thus was born the ACS. ACS is a cursor cache running on a pair of servers, providing a method to offload cursor processing from the database tier.

## Search

Our search tier runs on commodity Linux hosts, each of which is augmented with a 640 GiB PCI-E flash drive which serves as a caching layer for search requests. These hosts get their data from a shared SAN array via an NFS file system. Search indexes are stored on the flash drive to enable greater performance for search throughput.

Search indexing currently occurs on translation servers which mount LUNs from storage arrays via Fibre Channel SANs. Those LUNs make up a QFS file system which allows single writer but multi-reader access. Like most other critical systems, we run these in active/passive with the passive node doing some low priority search indexing work. It then ships its results to the active partner to write into the QFS file system.

The translation occurs when these same LUNs are mounted read-only from a group of four NFS servers running Solaris 10 on SPARC. These SAN mounted file systems then are shared via NFS to the search tier previously described.

### Fileforce

We maintain a tier of servers that provide object storage, similar in concept to Amazon's S3 or OpenStacks' Swift project. This system, Fileforce, was developed internally to reduce the load on our DB tier. Prior to the introduction of Fileforce, all Binary Large Objects (BLOBs) were stored directly in the database. Once Fileforce came online, all BLOBs larger than 32 KiB were migrated into it. BLOBs smaller than 32 KiB in size continue living in the database. All BLOBs in Fileforce have a reference in the database so in order to restore Fileforce data from backups, we have to start a database instance based on a database backup from the same restore point.

Fileforce includes a bundler function, developed to reduce the disk seek load on the Fileforce servers. If 100+ objects smaller than 32 KB are stored in the database, a process runs on the app servers to bundle those objects into a single file. A reference to the bundled file remains in the database along with a seek offset into the bundle. This is similar to Facebook's Haystack image storage system but built into an object storage system.

### Support

Each instance contains various other servers for support roles such as debugging application servers and "Hammer testing" app servers in the app tier, hub servers which monitor each instance for health and monitor servers running Nagios. Outside of the instance itself reside supporting servers like storage management, database management, log aggregation, production access authentication and other functions.

## Future Directions

On the database tier, we're carefully examining several options for data storage systems. We're also evaluating higher speed, lower latency interconnects such as Infiniband for our cluster interconnects with our existing database solution.

In the future, we expect to make the search hosts do all reads and writes. We are in the process of rolling out Apache Solr for search indexing. Solr runs locally on our existing search hosts. The SAN, NFS servers and SPARC based indexer hosts will all go away which will lead to a dramatic reduction in complexity of the entire search tier.

Our application container was previously Resin but over the last year, we've been migrating to Jetty. We’re also in the midst of a hardware refresh of our application tier hardware which will increase RAM sizes anywhere from 30% - 266% and introduce Sandy Bridge processors into our stack.

I hope this overview of the salesforce.com technology architecture and stack has been interesting and informative. Please leave a comment if you’re interested in additional guest posts deep diving into other parts of our expansive technology infrastructure. Thanks for reading!

## Related Articles

- The Internal Design of Force.com’s Multi-Tenant Architecture
- Salesforce.com Architecture
- Cell Architectures