Architecting High-Volume Reads in Salesforce

Alpine Code
October 18, 2020

SHARE:

One of the first lessons that architects learn when working with Salesforce is the impact of governor limits on performance. The multi tenant architecture of Salesforce affords tremendous flexibility and scalability, but its shared nature requires all of us to be good neighbours. Depending on how we are integrating with Salesforce, different techniques can be used to best ensure we provide the scale that we need.

As API demand increases, the risk of reliability problems grows, and this can have consequences for the business. It is important to understand the business needs around all requests for data to ensure that the requests are responded to in a timely manner. "Timely" can have differing meanings, as some business processes may require data in near real time whereas others can afford greater latency. Once you clearly understand the business needs for the data, you can employ different strategies to meet those needs.

This post explores issues to consider when the system you are architecting needs to make a large volume of API calls to read data from the Salesforce Platform. As the considerations are slightly different between high-volume reads and high-volume writes, high-volume writes will be explored separately in a future post.

Building Trust Through Scalability

When technology is unable to keep up with the data demands of the business, problems inevitably arise. Most commonly this is experienced as slow-downs, with users experiencing long delays as data is retrieved. For example, in a customer service center, poor data performance leads to longer call handling times. Beyond this though, even the quality of the data can be affected. Data synchronisation processes that aren't fast enough can lead to distributed systems having incorrect or even corrupted data.

As these problems grow, confidence and trust in the system deteriorates, which can affect the entire business.
The resulting confusion and miscommunication can undermine the confidence of business users and customers alike. The mounting frustration can lead to lost customers, low employee morale, and higher turnover.

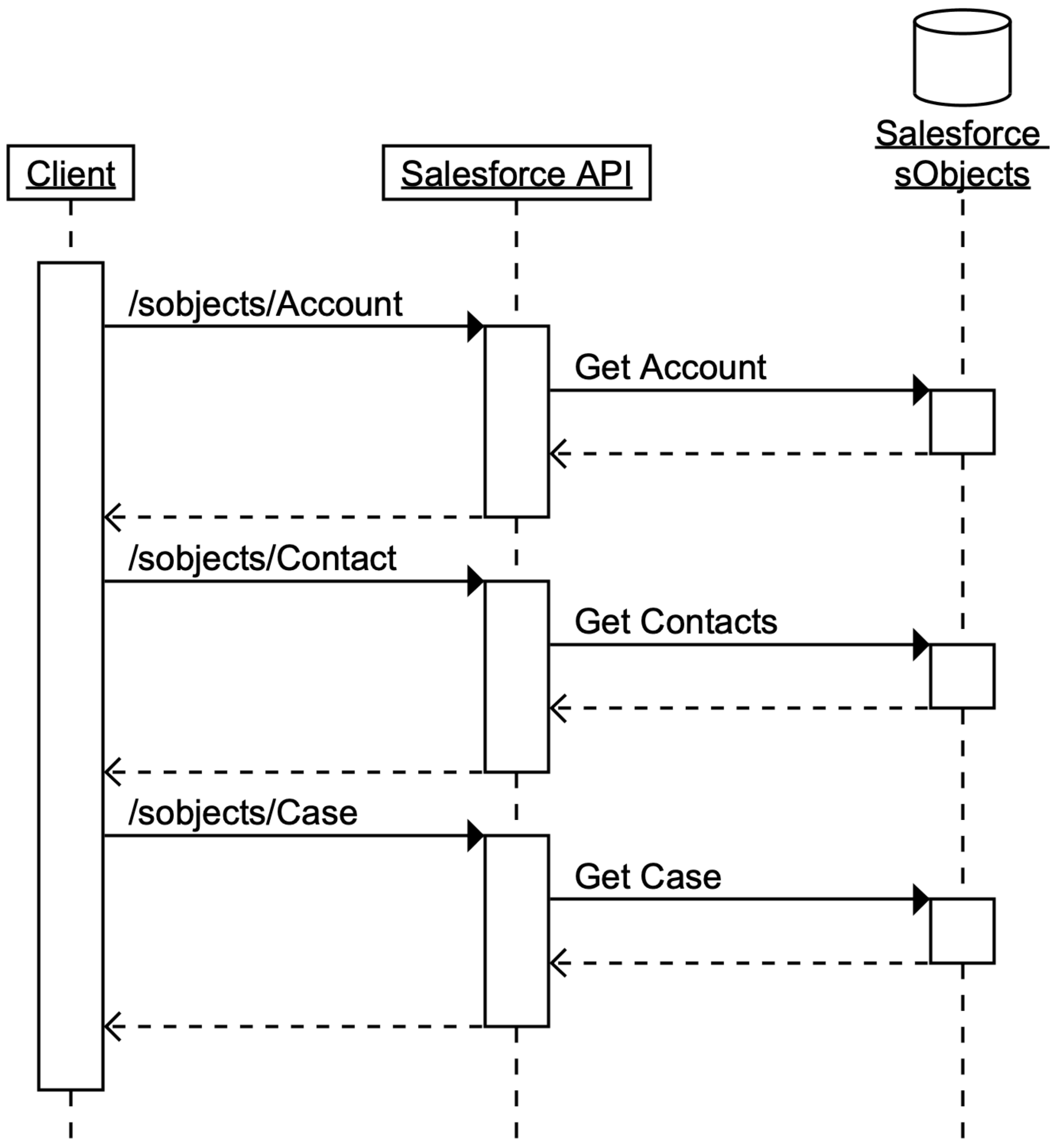
To build trust, you must address scalability issues that are affecting platform performance. Before addressing these issues, you must first fully understand the business need. This understanding will enable you to make informed decisions when tradeoffs — in which performance in one area is sacrificed to improve performance in another — are necessary.

Scaling Salesforce APIs

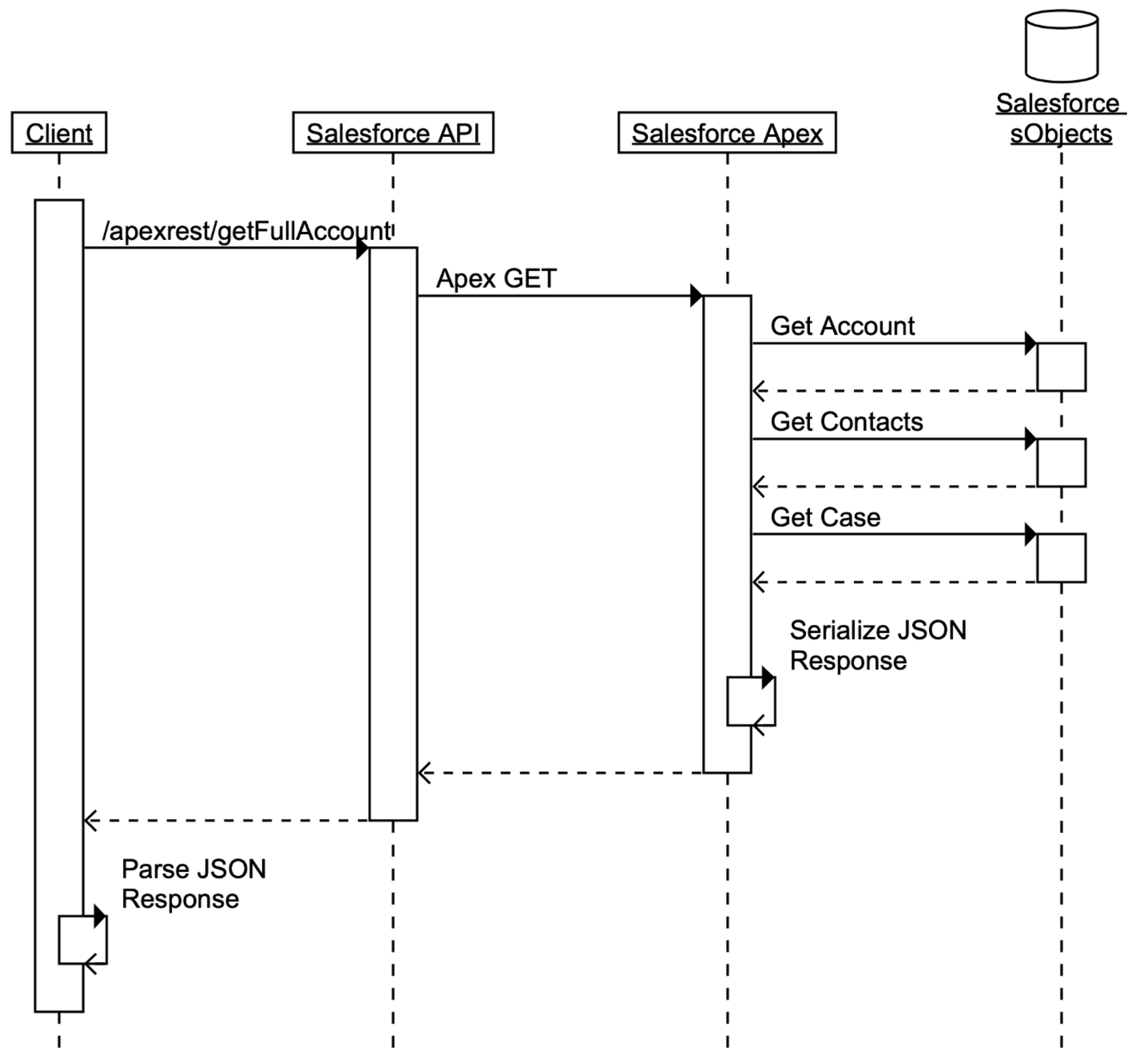
When building API integrations with Salesforce, the most obvious place to start is with the native Salesforce REST and SOAP APIs. Both APIs support high-volume reads, subject to governor limits. These limits have changed over time, so before designing a scalable solution be sure to review the latest developer documentation. For APIs, the two key limits are *concurrent API requests* and *total API requests*. You can work around those limits through changes to your design, but these tradeoffs can lead to other limits becoming a factor.

For example, rather than make multiple independent API requests, you can use custom Apex REST APIs to consolidate a request. Requests for several different related elements of data can be written as a single request. This tradeoff reduces the risk of hitting the limit for the total number of API requests but increases the risk of other limits such as concurrent API request limits, Apex CPU time limits, and Apex heap size limits.

Here is a simple example. Three API calls are made in sequence against native Salesforce REST APIs. If you begin to run into the limit on *total API requests*, with this integration you could redesign it to use a single custom REST API call.



The *getFullAccount* API call orchestrates the Salesforce data, before returning it to the client. This reduces three API calls down to one, helping to avoid the limit on total API requests. However, you can expect this one call will require more time to execute, potentially risking the *concurrent API request limit.*



You can also use the Composite API to consolidate multiple related requests into one call. This approach simplifies how you build your calls, and reduces the risk of hitting the limit for total number of API requests. And as of the Winter '21 release, you can now use the Composite Graph API to package a complicated series of sub-requests into one call, allowing you to process up to 500 sub-requests in a single payload, with the assurance that if any part of the operation fails within a given graph, the associated transaction is completely rolled back.

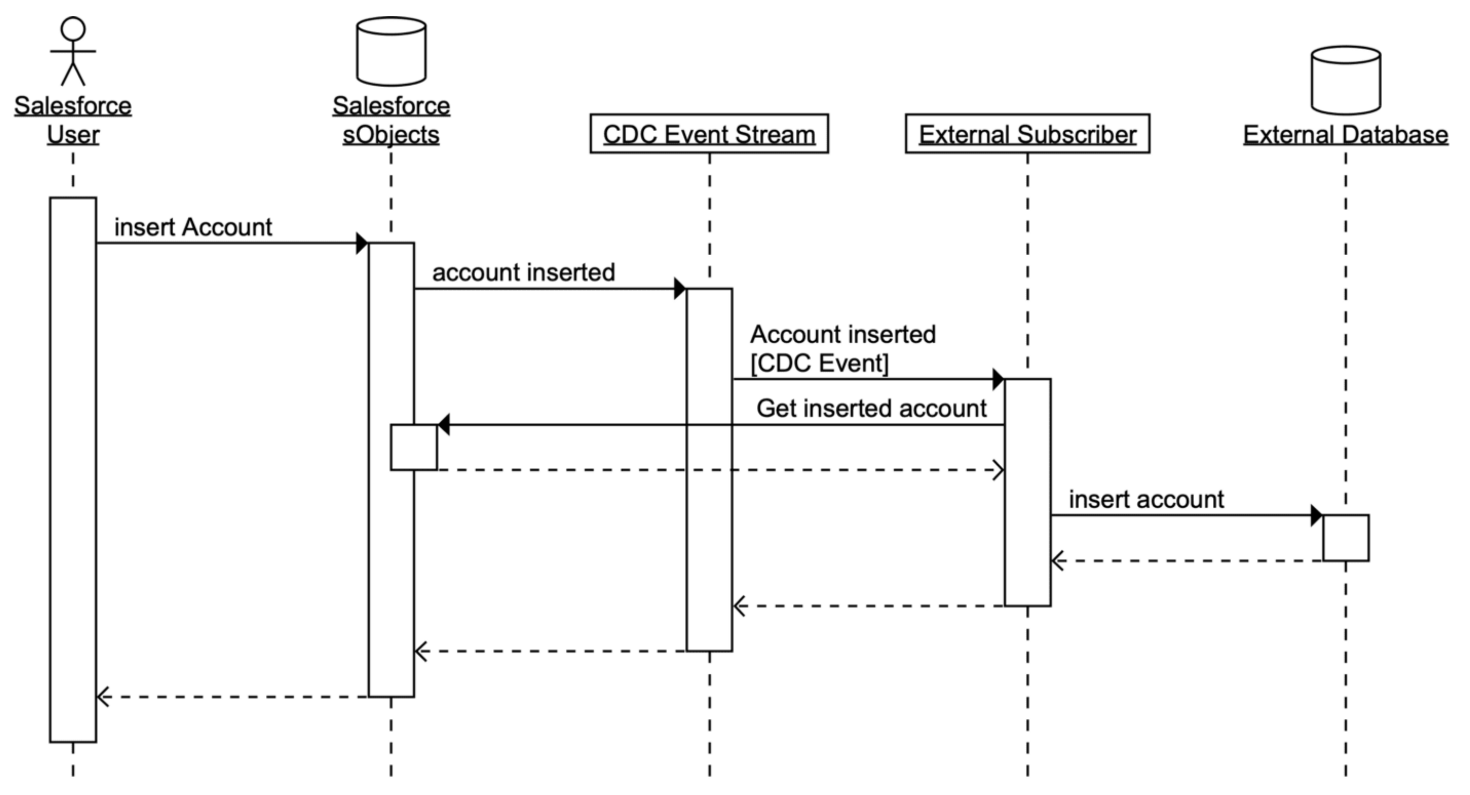
The flexibility to balance between different governor limits can achieve greater scale, but only up to a point. If you're receiving enough API traffic even a very short transaction can lead to *concurrent API request* limit issues. To achieve the next level of scale it you'll need to use a different architecture, leveraging other aspects of the Salesforce Platform or expanding beyond the platform.

Salesforce Streaming Events

Salesforce provides a streaming event architecture that provides a different approach for handling high-volume data. Rather than synchronously requesting data from Salesforce, data can be pushed out from Salesforce to other systems. PushTopic, Change Data Capture (CDC), platform, and generic events all provide slightly different capabilities for streaming data. The event type you choose will be dependent on your specific use case, but the overall architectural pattern is similar between them.

For example, Changed Data Capture (CDC) events provide a way to notify external systems of data changes in Salesforce as changes occur. Because CDC events are fundamentally asynchronous, there's no guarantee that any given change will be immediately available on the external system. However, by supplying data to external systems and using those systems to handle data requests, you can reduce the need to read large volumes of data directly from Salesforce.

In this example, an Account record is inserted into Salesforce. Upon insertion, a CDC event is created and subscribers to that event can react to it. In this case, an external subscriber gets the latest copy of the event, and inserts that event into an external database.

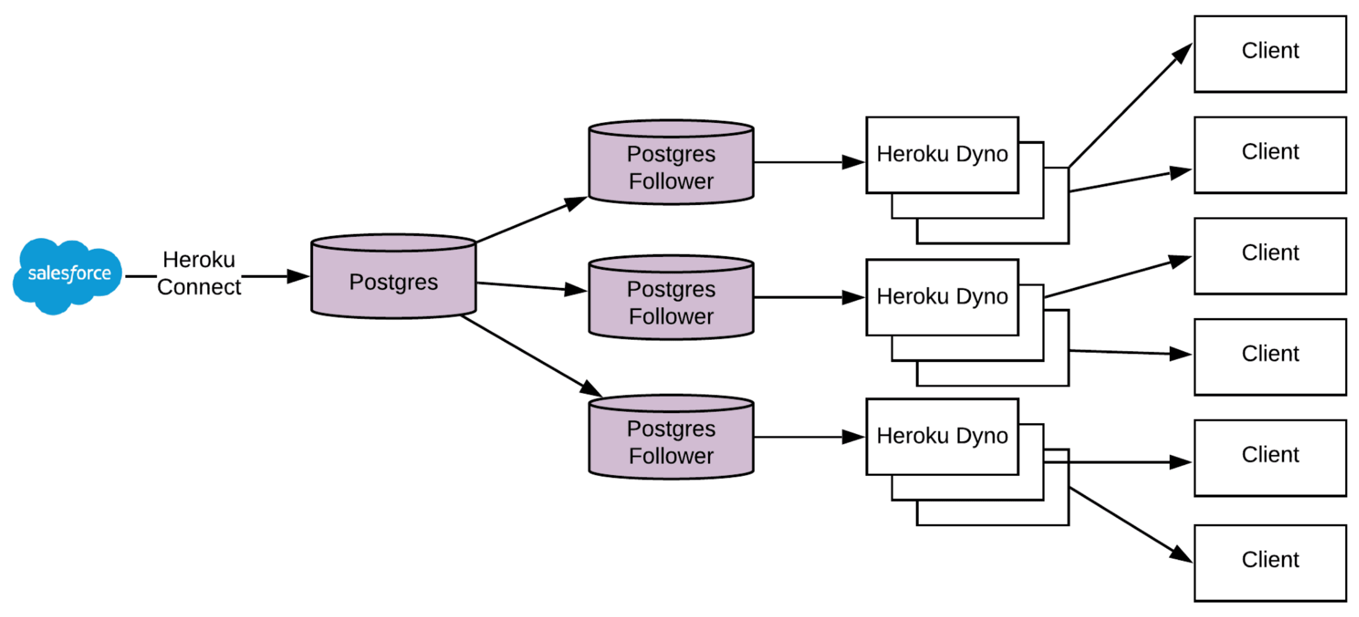


With this approach, other systems can query for the account data against the external database without needing to directly access Salesforce.

This scalability advantage does come with some complexity. Message delivery isn't always guaranteed, so in rare cases, a change in Salesforce could be lost, leading to synchronization issues with the external system. It's important to have a process to reconcile any such data synchronization issues that may develop over time. Even with a reconciliation process in place, streaming events cannot provide the same guarantees as a synchronous transactional approach. However, if your business need can be fulfilled given these limitations, platform events offer tremendous gains in scalability.

Extend With Heroku

As part of the Salesforce Platform, Heroku is well suited to handling large volumes of API requests and is frequently used to improve scalability. For example, a common pattern is to use Heroku Connect to enable synchronization between Salesforce and Heroku Postgres.



Here, a large number of client systems can access Heroku to retrieve data that is kept in sync with Salesforce. As the clients scale, the number of Postgres databases and the number of Heroku dynos can be scaled to meet the increased demand. Meanwhile, the demand on Salesforce itself remains unaffected.

As with platform events, this scalability comes with some added complexity. External clients will either need to query the Postgres database or connect through a custom API implemented on Heroku web dynos. Heroku Connect has strong management tools, but you'll need to consider how sandbox refreshes work with your integration testing environment. Also, as with platform events there is some risk that data can get out of sync in rare circumstances.

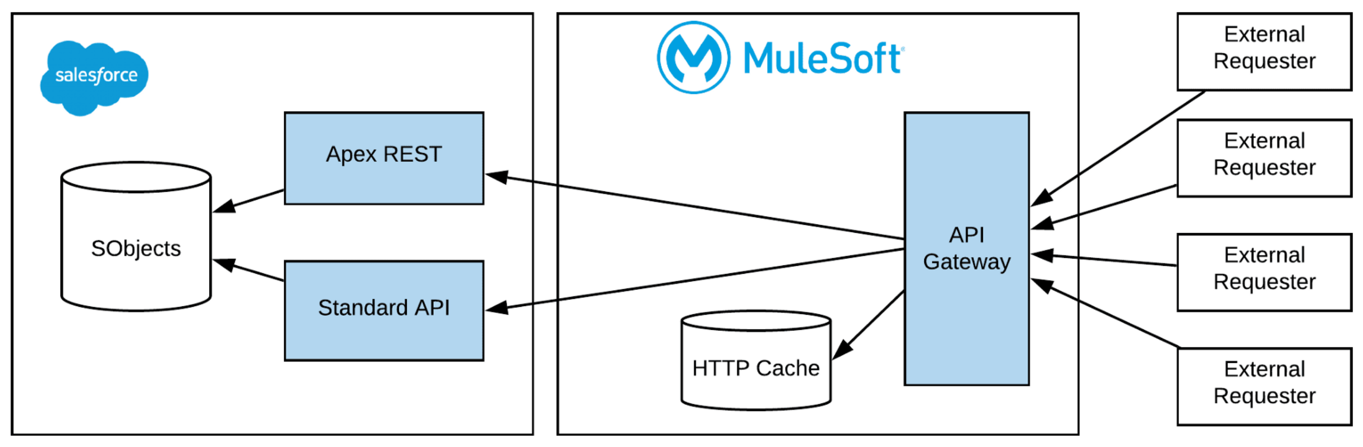
Security is another factor to consider because the ownership structures in Salesforce are not carried into Heroku. If you have complex security requirements, you may face additional complexity in the Heroku implementation.

For many businesses, this approach offers the best of both worlds. Salesforce provides a flexible base architecture, with the steady scale that a business needs day to day, and Heroku complements this with the ability to scale up to meet the needs of high-volume processes that would be affected by governor limits if run directly against Salesforce.

MuleSoft

MuleSoft's Anypoint Platform (along with CloudHub) supports highly scalable integrations with an architecture similar to the Heroku architecture just described. Cloud-based workers can take on the demands of incoming read requests and scale dynamically as needed. Additionally, the tooling of the Anypoint platform simplifies much of the work to configure these integrations.

For example, to support high-volume reads, MuleSoft Anypoint can act as an API gateway in front of Salesforce data. On its own, this doesn't address the concerns of high-volume reads, however, the Anypoint platform provides caching capabilities as well. Depending on the nature of the data, this cache capability can greatly reduce the demand on Salesforce while minimizing custom code.



GraphQL

In Salesforce, if you wanted to get the data in the example above, you'd need to do multiple requests to the REST API, and there are limits on the number of API calls you can make a day. With GraphQL support, we can avoid this problem and make integrations more efficient by getting all this related data in a single request.

GraphSQL is a free package that extends the Salesforce REST API to include a new REST endpoint in your org, where you can send any standard GraphQL query, and using only one REST API call, get data from several objects back.

Conclusion

Salesforce provides tremendous scalability out-of-the-box, but being a shared system, governor limits will always create a performance ceiling. For business needs that require high-volume reads you may want to consider an architecture that includes Heroku, MuleSoft, or external systems updated via Salesforce streaming events.