Getting Data Out of Salesforce

by Patrick Connelly posted on August 13, 2016

If you've been working with Salesforce for a while or you're a larger company you'll eventually want to get data out of Salesforce and into another system. So let's take some time to break down the options that are available for getting data out of Salesforce in varying degrees of realtime.

On Platform Options

I'm calling these "on platform" because they do not require a 3rd party system to package up the data and send it. You will need a 3rd party system to process the data (or to send it to) but you won't need one to retrieve the data.

Outbound Messaging

Outbound messaging is something I have talked about previously in regards to using it an JBoss to roll your own ESB. The simplest explanation of outbound messaging is that it creates an XML message of anywhere from 1 to 100 messages that contains the data you generate and then makes a POST call to an endpoint. The endpoint that the POST is made should process the XML and then return an ACK back saying that the message was process correctly.

Features

- **Automatic Retry –** One of the biggest selling points in my opinion for outbound messaging is that the messages are requeued and retried if the delivery fails. This could mean that the endpoint is down, too busy to handle the request or a similar reason. If the message fails, it is tried again later.
- **No Code Required –** The way you fire off an outbound message is typically via a workflow. This means that it is done all declaratively and does not require any development on the Salesforce platform.

Limitations

- **No Guaranteed Order –** Because these requests are asynchronous in nature and they could be retried, there's no guarantee what order they'll end up in your system. If you are only sending these messages on create, then this isn't a big deal but if you are relying on updates coming over and the order of those updates matter it could be.
- **No Guaranteed Timeframe –** Asynchronous calls in Salesforce are guaranteed to happen *sometime* in the future. That's the only guarantee. There are no SLAs around how fast the queue will be processed. So if you need near-realtime performance, you could spend minutes or even hours waiting for messages to be sent. *(This time is highly dependent on the load of the pod as well as the number of messages in the queue)*
- **Not Great with Large Data Volumes –** Remember above when I said that it'll send up to 100 messages to your endpoint at a time? Well if you're creating messages faster than they can be processed then you're queue is going to stay backed up and could take even longer to process.
- **Can't Dig into Related Data –** Want to send account name from a case? Don't have it as a formula field on the object? Well too bad! Outbound messaging only supports sending fields that are directly on the object you are creating the message for.
- **Lack of Strong Security –** While you can us SSL with outbound messaging, this is your only real option for security. You cannot set any sort of credentials on your request and you cannot verify who you say your are via something like mutual SSL. Your only option is to whitelist the IP address(es) for Salesforce and hope for the best.

Conclusion

I'm still a big fan of Outbound Messaging, I like the simplicity of doing the work declaratively. It's very easy to see what the status of your messages are in the Setup UI. I think for smaller instances and implementations where order of operations isn't critical, it could be a decent solution since it requires almost no work inside of Salesforce.

Callouts

For callouts I'm going to be talking about using an @future call, Batchable Apex, Scheduled Apex or Queueable Apex to make a remote callout. All of these methods can make an Apex callout to send data in XML/JSON/CSV/etc to any endpoint. Checkout the Apex REST Callouts unit of the Apex Integration Services Trailhead module to see it in action.

Features

- **Flexible Data Types –** Unlike outbound messaging, callouts can GET/PUT/POST/PATCH pretty much any data type you want. JSON easier for you to understand, do that! Hate yourself and want to use XML, go for it!
- **More Security Options –** Since you are formulating your HTTP request, callouts give you the flexibility to do lots of different types of security. You can do your basic auth, stored oAuth and even mix in some mutual (two-way) SSL for fun.

Limitations

- **No Guaranteed Order –** Similar to the outbound messaging, callouts are asynchronous calls that can happen whenever they darn well want. If you were to make multiple @future calls there's no guarantee what order they'll execute. This can be mitigated a little by using batch, scheduled or queuable apex by determining the order in them but they introduce other problems.
- **Not Guaranteed Timeframe –** Again, they'll happen sometime in the future. Even your scheduled Apex will happen "sometime after the scheduled time" not right on it. This can be most problematic when the async queue becomes bogged down and your calls wait. Personally, I've seen the queue get backed up by more than an hour and a half regularly (under high volume). This can wreak havoc if you are trying to schedule apex on a very tight timeline.
- **No Retry –** If your callout fails in a future call (let's say your endpoint is down) that's it, there's no retrying. You cannot call a future call from a future call so you can't kick it off later.
- **Difficult to Log Failures –** It's really tough to know of callouts are failing. Since the logging is only stored in the normal debug logs, you have to be watching them to know that they are failing. You could log them to a 3rd party such as Loggly or Splunk, but who's to say if your not failing, or just failing to log your failures.
- **Governor Limits –** There are some governor limits around callouts. For the most part any org of any size won't hit the number of callouts in a 24hr period but you could very easily hit the time, size or number per transaction. Also, there's a really nasty one regarding not being able to do DML → callout → DML that will leave you very frustrated

Conclusion

I've written several articles that include callouts to external webservices and for some use cases it works well. Where I think callouts really fail is when it comes to dealing with data at scale. The lack of order and the potential loss of sent data make it difficult to consider.

Off Platform Options

These options require you to have some sort of 3rd party system to gather the data and then do something with it. Obviously this could be done on a PaaS like Openshift or Heroku or on system hosted internally. One of the biggest upsides to these systems is that they typically do not require that an endpoint be publicly exposed to the internet.

Heroku Connect

Heroku Connect is a product that was announced a couple of Dreamforces ago and it provides bi-directional data synchronization between Salesforce and a postgres database running on Heroku. There's a neat quick start Trailhead project for Heroku Connect to see it in action.

Features

- **Bi-directional Data Sync –** Any data in Salesforce gets copied to the postgres database and any data created / updated in the postgres database gets updated in Salesforce. Behind the covers it's using the Streaming API so it's pretty close to realtime

Limitations

- **Cost –** Let's get this one out of the way right off the bat. It's not cheap. In addition to the fee for the product, you also have to pay for the postgres storage on Heroku.

- **Could Be a Bit Much –** If all you're looking at doing is capturing some data from your Salesforce org, Heroku Connect could be like going after a fly with an elephant gun. Ideally, you'd want to interact directly with the postgres database. But if you need this data in an existing system, then you'd have to pull the data from the postgres and dump it into your other system.

Conclusion

I think the Heroku Connect is a really interesting option. Given an unlimited amount of money and a brand new system to work with, it could be really great. Delegate all of your users to Salesforce and then have your customers or visitors use a website running on Heroku connecting to your postgres database. But I think the total cost of it is pretty prohibitive and you're still left with the problem of getting the data out of a Heroku postgres db and in house.

Streaming API (Push Topics)

Streaming API is an interesting thing because unlike the other solutions* you subscribe to a published push topic and then interact with the data.

- Technically Heroku Connect uses the streaming API but since it's behind the scenes, I don't know if I'd really count it.

Features

- **True Realtime –** This is one of the few options that offers true realtime updates for you topics. There are some caveats here but it should be as close to realtime as you can really get.
- **Message Reliability –** Prior to the API 37.0 release, if you weren't always listening to a stream (like your connection or application died) you could miss messages. In the most recent release, Salesforce added message reliability so that you can play back missed messages.
- **Message Durability –** As part of the API 37.0 release, messages can now be played back up to 24hrs later to catch up on any missed.

Limitations

- **Lot of Work –** This subscription type of model isn't that well known and can be kinda difficult to deal with. If you're doing a client side web app, then Streaming API may be a perfect fit since you don't have to deal with the reliability and durability. But if you're trying to use streaming API to store all updates for an object then you have to take into account what happens if you lose connection and you have to reply all missed messages.
- **Governor Limits Can Be Tricky –** While most of the limits for the streaming API may seem like enough, when you hit them, you've got very few options. For backing up everything as a single service user, the number of subscribers shouldn't be a problem. But if you're using it as a client facing app as well, that number quickly dwindles. Also, the maximum length of the SOQL might seem like plenty, but if you have an object with a bunch of fields (especially if they are long names) that number of characters runs up quick.

Conclusion

I really really want to like the streaming API. It's got all the bits to make real-time communication happen. But I think since the message reliability and durability are still in their infancy (only one release old) it may not be tested enough to recommend. If you're going to do some client side work that requires real-time data then I think this is a perfect fit.

Polling (REST / SOAP APIs)

Polling Salesforce (via the standard REST/SOAP APIs or custom REST/SOAP APIs) is probably the most common way to pull data out of Salesforce. And there's a good reason behind it. It works.

Features

- **Get Exactly What You Want –** And get it when you want it. Unlike the other options, you control when you poll into Salesforce to get your data. Want to dump it every night? Want to get updates every minute? You can do it. (Limitations apply, see store for details)
- **Bundle Multiple Queries –** One of the things that I think gets overlooked when talking about polling is that you can write your own REST / SOAP endpoints and poll them. And as part of this you can combine multiple objects into one result. You have to be weary of governor limits in the Apex, but this is one great way to reduce the number of API calls you make
- **You Can Get a Bunch of Data –** By using the Bulk API, you can pull down 50 Million records in a rolling 24hr period. (See full limits)

Limitations

- **API Call Limits –** The biggest is going to be the number of API calls you can make to your org in a rolling 24hr period. This is determined by the number of liscense you have in your org.
- **Number of Records –** You are limited to 2,000 records per query call. To get more than this you will need to segement your call and make a new query. This can be excacerbated by the next limit.
- **Datetimes Store to Second –** One of the more common ways to split your queries would be to get the last CreatedDate and just query for anything created after that. Well if you do a large dataloader job and end up with a bunch of CreatedDates that are the same, you could either end up in a loop where you keep processing the same 2,000 records over again, or you end up skipping a bunch of records. So to handle this, you have to split on more than just the CreatedDate. Not a terrible limitation but one to note.
- **Frequency –** No matter how tight you make the calls, it just won't be real-time. You should wait for your existing query to finish to then make a new one so you don't grab duplicate data and so you don't end up hitting the concurrent Apex limit.

Conclusion

Overall, this polling is the "sanctioned" solution from Salesforce to consistently pull data out of Salesforce. While it's not without it's flaws (and it's not nearly as sexy as any of the other options) it's the tried and true option. And it's that because it works the most.

Final Thoughts

I'm sure I missed some features and limitations for these options. And I'm sure I've missed some options for getting data out*. If it were up to me, I would want Salesforce to update outbound messaging and make it a more viable option. But in my opinion, it's been on the platform for so long without any substantial updates that I'm not holding my breath for anything to happen.