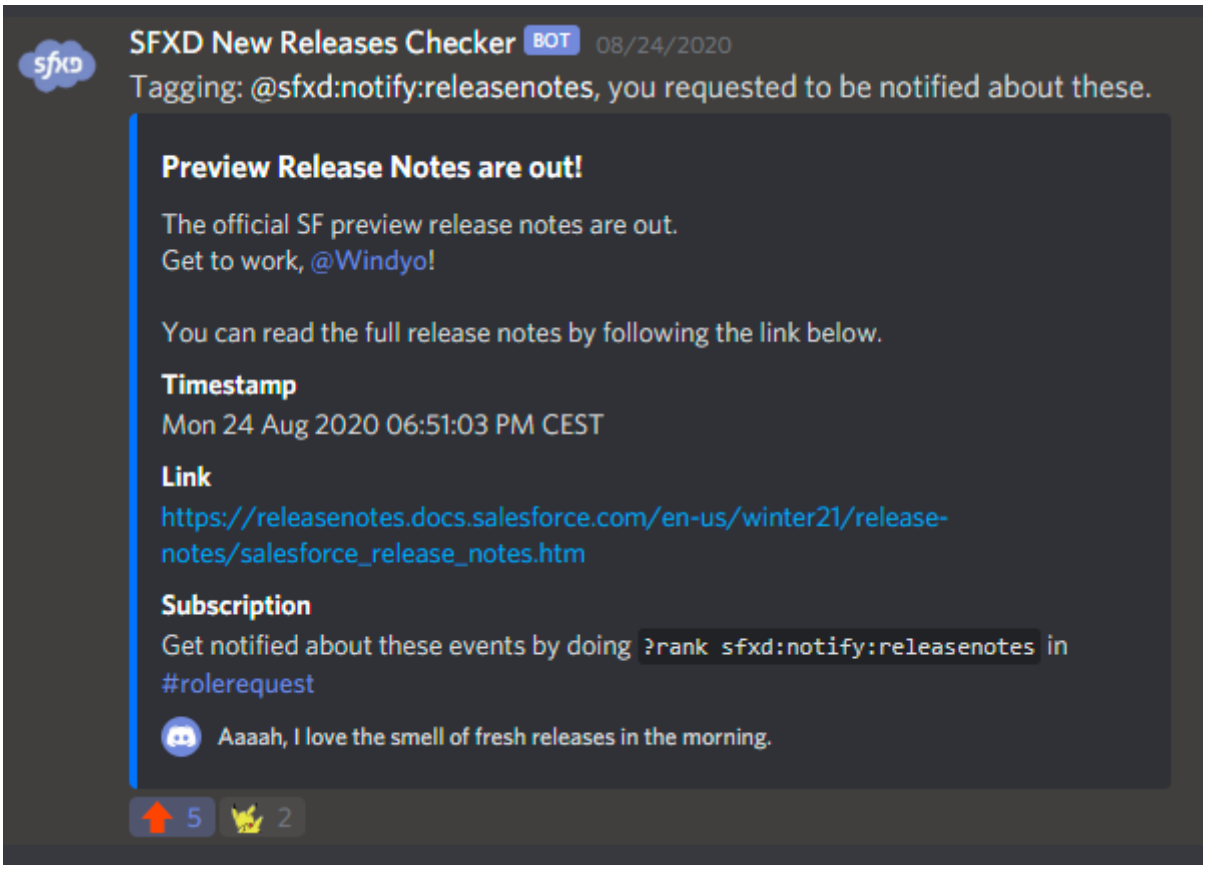
SFXD Presents - Parsing the new Help site is horrible

(Like really)
(Please don't break my parser)
(Plllllllllllllllllllllllllllllllleeeeeeeeeeeeeeeeeeeeeeeease)

INTRODUCTION

One thing that I provide SFXD members (and myself, honestly), is a couple of bots.
Those bots do various things, but one of my favourites is my Notifications bot, which isn't really a bot, it's a shell script. A 10-line shell script.
This courageous little fellow, based on the current time of year as well as the actual year, constructs the next release (summer21, as of writing, or release 232), and then queries both the Release Notes page and the Preview Organization form - and if they go live (HTTP code 2XX), it pushes a nice notification in Discord. It also monitors the release notes for changes. All of this was easily done by querying the HTML of said pages.



Now obviously the script isn't perfect:

- Salesforce has a tendency to do weird things with its pages when nearing a release, making the pages come live and then not again, which results in bad notifications,
- using cURL to get the HTML means I was quite limited in processing unless I wanted to depend on pyton or another utility for parsing.

So I had been thinking of updating it to something more robust for a while.

Then, Salesforce decided it would move the Release Notes from
http://releasenotes.docs.salesforce.com/ to https://help.salesforce.com/.
The new website was slightly prettier, and didn't lose function, so I saw this mainly as an great excuse to finally change my notifier bot to something better.

... it didn't go so well.

THE RIGHT TOOL FOR THE JOB

So now that I had decided I wanted a better notification system I looked into various tools.

For those who know me, I'm somewhat of a self-hosting freak.
If I can self-host it, I will - it's funny, teaches me about server management, and that way I have the guarantee my tool won't disappear into the aether based on the whims of a company or

individual.

So I looked into Statuswatch, changedetection.io, and other such equivalents, before settling on Huginn. I, at the time, thought that it was WAY overkill but had the potential to be useful for other subprojects.
It turned out to be the PERFECT tool for the job, and I do not think this would have been doable without some heavy scripting if I had gone with another tool.

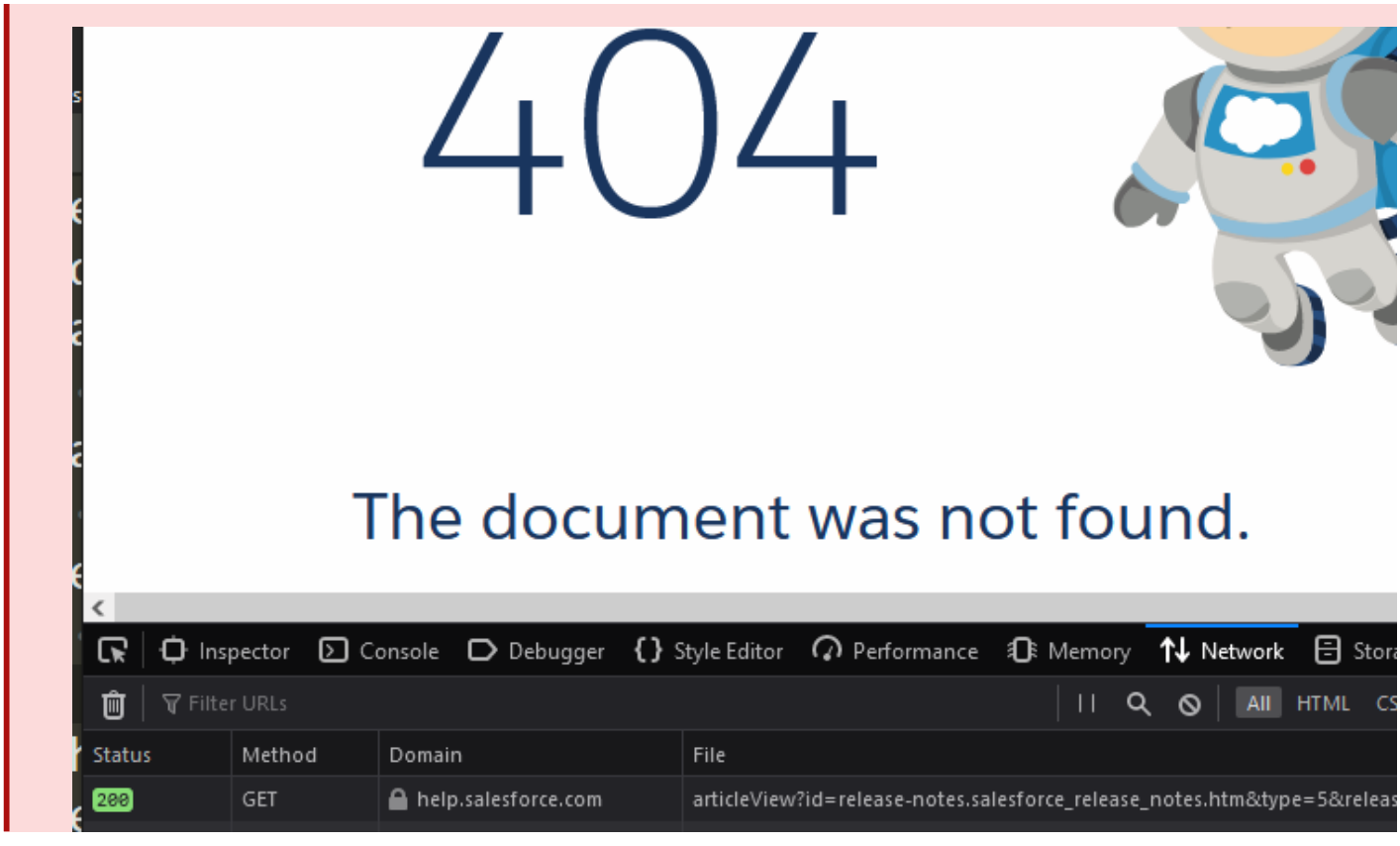
Yes, I am saying you need an entire events-stream based automation tool to parse release notes. We'll see why later.

GETTING THE RELEASE NOTES VERSION

In order to get the release notes, I need a few things:

- determine which release we currently are
- determine if a new version is coming
- determine if the webpage for the new version is up yet.

My original plan was to query `https://help.salesforce.com/articleView?id=release-notes.salesforce_release_notes.htm&type=5&release={{version}}`, much like I was doing previously, and trigger notifications if I had a confirmed (lasts more than a minute) `HTTP2XX` code - the old webpage returned a `302` if the version wasn't published yet.
A first problem arose: the Help site ALWAYS answers with `200`. Even if the version just doesn't exist at all. Try it: https://help.salesforce.com/articleView?id=release-notes.salesforce_release_notes.htm&type=5&release=96 the page *shows* you 404, but actually tells the browser "**YUP HERE'S YOUR PAGE.**"



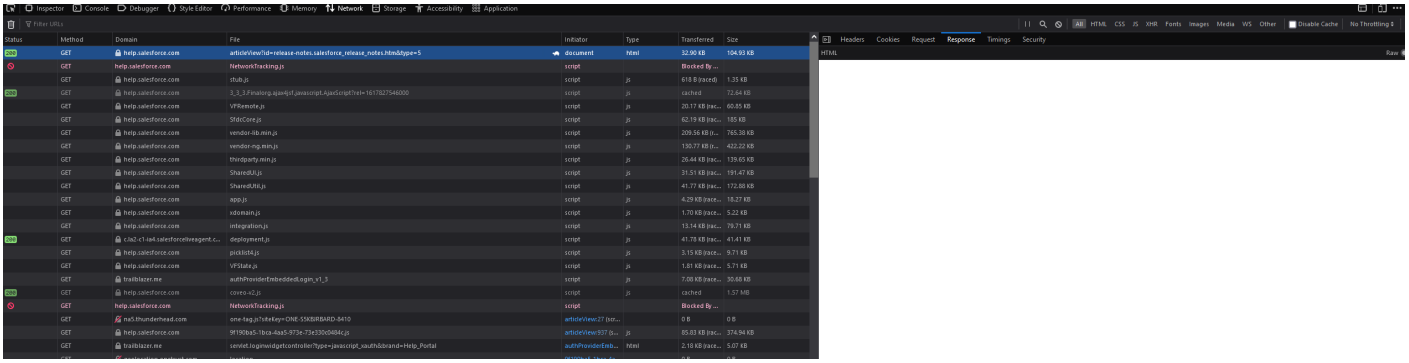
> Yes I know Help pages are mostly cost centers but this is just bad design

"No matter", I thought. "I'll just query https://help.salesforce.com/articleView?id=release-notes.salesforce_release_notes.htm&type=5, see which version number it returns, and check if `{{version +2}}` is a valid parameter". (version numbers increment 2 by 2 in Salesforce releases).
As you can guess, this doesn't work either. Funnily enough if you ask for the NEXT version, **Salesforce Help will still answer HTTP2XX in the rendered page**... but show you the current version. So 404 for invalid sessions, but 200 for next versions, OK, cool.

The next solution was then to get the current page, parse the version number, add 2, query that page, and check if versions differ.
And that's where the pain began.

GETTING THE RELEASE NOTES

Implementing the webpage fetcher was one of the most annoying things for this entire project. My original assumption was that I could just use a Huginn Website Agent (basically still a cURL command) to fetch the HTML, and then parse it using one of the many parsers Huginn offers. This would have worked if the Salesforce Help page worked like a normal web page. It does not.



such response, much wow

The Salesforce Help first answers any query with a wrapper, containing limited HTML and a LOT of Javascript but no other content.
This javascript then implements a Visualforce Remoting call, which queries Salesforce to display what seems to be KnowledgeArticles and Content on the page.
The result of those calls determine what you see afterwards.

The problem therefore was that Huginn, as well as cURL, sees this wrapper when doing their query. And they can't execute the JS calls themselves.

I fired up the Network inspector of Firefox, desperately looking for something that would avoid me having to fire up Selenium or something just as heavy-handed just to get the sweet text-based release notes I wanted.
LOTS of searching after, I found... nothing. Literally everything was handled through JS and fed

through VFremoting calls.
I naturally took the mature option and bitched about it to SFXD members, and @psb was kind enough to highlight that ONE call made on that page - the release notes Version picklist - was not to `apexremote` but to a static URI: `https://help.salesforce.com/services/apexrest/Help_TOCService?language=en_us&multiTocId=release-notes.salesforce_release_notes_toc&release={{version}}`

I hopefully queried it, and got back something I could work with: Raw JSON indicating the release number, the link to the PDF... and the HTML body embedded in an XML wrapper for some reason.

From there, the logic was simple:

- get `https://help.salesforce.com/services/apexrest/Help_TOCService?language=en_us&multiTocId=release-notes.salesforce_release_notes_toc`
- parse the version number, assuming it defaults to last stable release
- increment version number by two
- get `https://help.salesforce.com/services/apexrest/Help_TOCService?language=en_us&multiTocId=release-notes.salesforce_release_notes_toc&release={{stableversion+2}}`
- check if the release number in this subsequent call is equal to `{{stableversion+2}}`
- if yes, construct the release url `https://help.salesforce.com/articleView?id=release-notes.salesforce_release_notes.htm&type=5&release={{stableversion+2}}` send a notification with that URL, the PDF link, and the version number
- stop further notifications until the release number increments again.

This worked! (Except I can't know if it actually does until Salesforce releases the new preview so I can test it out, but at least the event stream worked.)

... but that wasn't the end.

GETTING THE RELEASE NOTE CHANGES

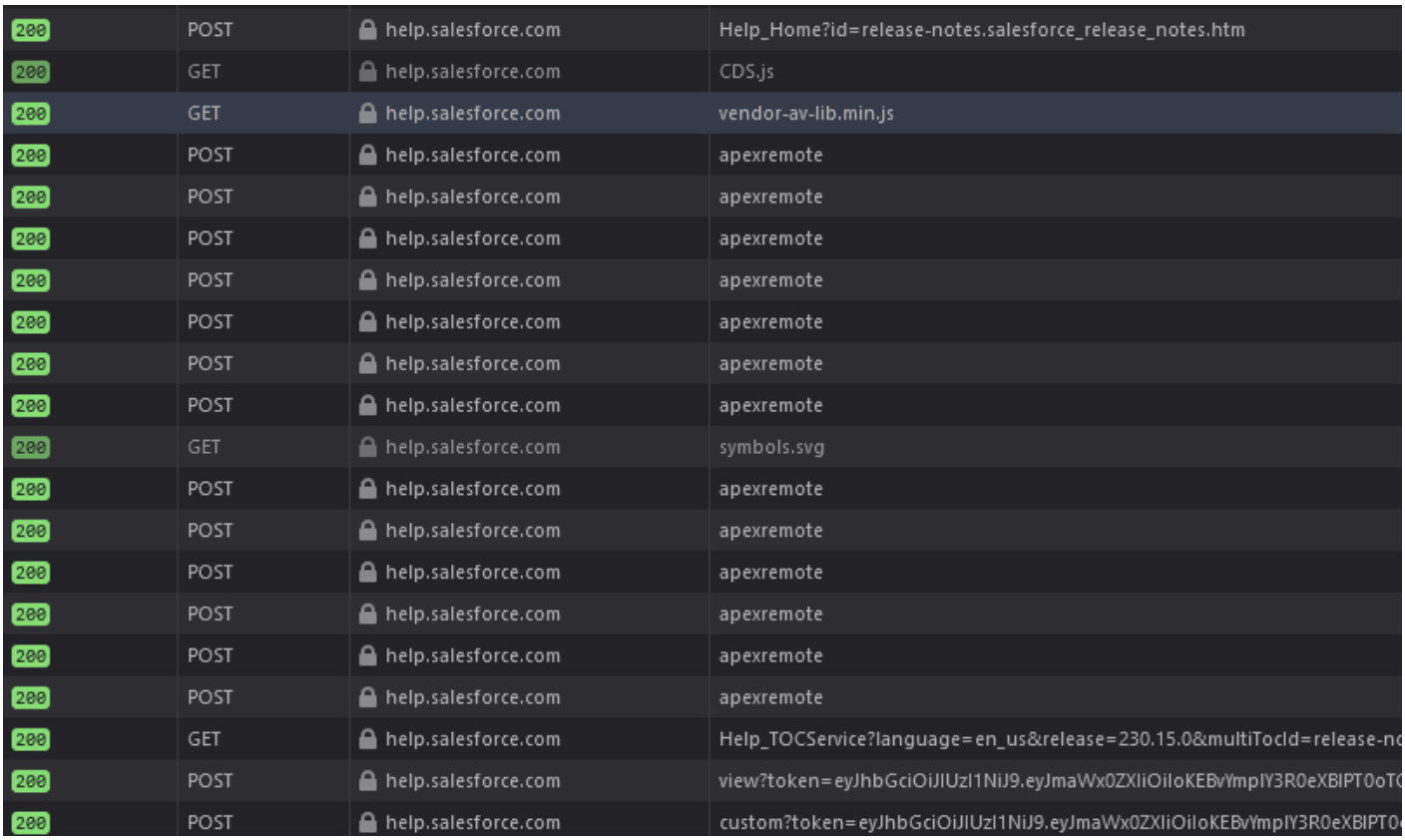
I had forgotten this part of it.
For those who don't know - the release notes, when not final, get modified. A LOT.

https://help.salesforce.com/articleView?id=release-notes.rn_change_log.htm&type=5&release=230
So for people like me who read them early, you gotta take the time to keep up to date. There's no schedule for these small updates, so using a change notifier was easier.

"*But you get the entire Release Notes body in some weird XML thing! You're fine!*" yes, that's what I thought. I hadn't checked said XML. Turns out it's just the headers for the top sections of the release notes, and I have no way of querying the changes to the release page.

So I went right back to the drawing board.

I was honestly pretty close to just paying for a PhantomJS cloud parser which could get all the page details for me. And then I decided "I'm already 6 hours in this project, it's 1AM, I might as well jsut go full ham". So instead, I decided to inspect every piece of information I got from every call, and try to get an APEX remoting call working via cURL.
After all, why not.



After checking all the calls done via `/apexremote/,` I found one called `getArticleData` which seemed to answer with the details I wanted in JSON format.

I used ReqBin to test a POST to `https://help.salesforce.com/apexremote`, using the same headers and payload as the browser. That yielded the same answer - cool. At least the authentication isn't a cookie.

Playing a bit with the headers and content made me realize a few things:

- I'll need to fake the Referer, Origin, and User Agent if I want `apexremote` to answer
- I'll need to send pretty much the same payload as the browser for it to resolve.
- This payload includes a VersionId, which I'm assuming I can get from somewhere... and a csrf token, because you wouldn't want people to browse your release notes from a non-Salesforce website, right. Gotta be secure.

The very first response I get from the Help Site as I said is mostly Javascript with no content. The JS yields some interesting information.

- one, there's a lot of parameters that seem to be some old stuff that just never got refactored (Hiiii org62 feedback button label)
- two, Visualforce.remoting.Manager is the class orchestrating everything and it seems to do some 143 calls before rendering the page. If you wonder why the original load is slow, that's why. Clicking on an article only does a VFremoting call for that one article - yay - but navigating to another menu item reloads everything. Which is sad considering it seems to load 135 of these elements on page load.
- three, the parameters for the VFRemoting manager are fed as inline JSON to this class. This inline JSON contains: the vfid, a list of actions including `getarticledata` AND a csrf token per action.

I had a way forward: I could now get that first page, parse the JS, parse the JSON within that JS, and then try to construct a POST that would pass verification based on the results therefrom. Not clean by any means, but what's a geek to do...

The good news is that getting said JS was easy enough. I just had to select all Script tags, emit those as individual events, and then filter on the one containing the Remoting Manager. I treated the resulting Javascript-which-contained-JSON as text, using regex to fetch the parameters the POST to `apexremote` needs. Dirty, but efficient.

The POST itself also went well. Turns out once you lie about where you are, which browser you are, which version you want and how the data is encoded, Salesforce answers happily to VF remoting calls done via scripts. I had a few issues with Huginn and its expectaiton that `application/json` be written simply `json` instead, which resulted in some weird parsing, but nothing major.

The data answered by the VF remoting call though is frankly horrible. It's a structured JSON, which contains a subobject which contains an XML wrapper for an HTML page which contains the changes to the release notes.
Due to some frustration with Huginn (mostly due to my lack of experience, but compounded by the fact that it did not allow me to view the raw data) I spent a LOT more time than I should have trying to parse this, but in the end I managed to get the holy grail - the html of the release notes changes page.

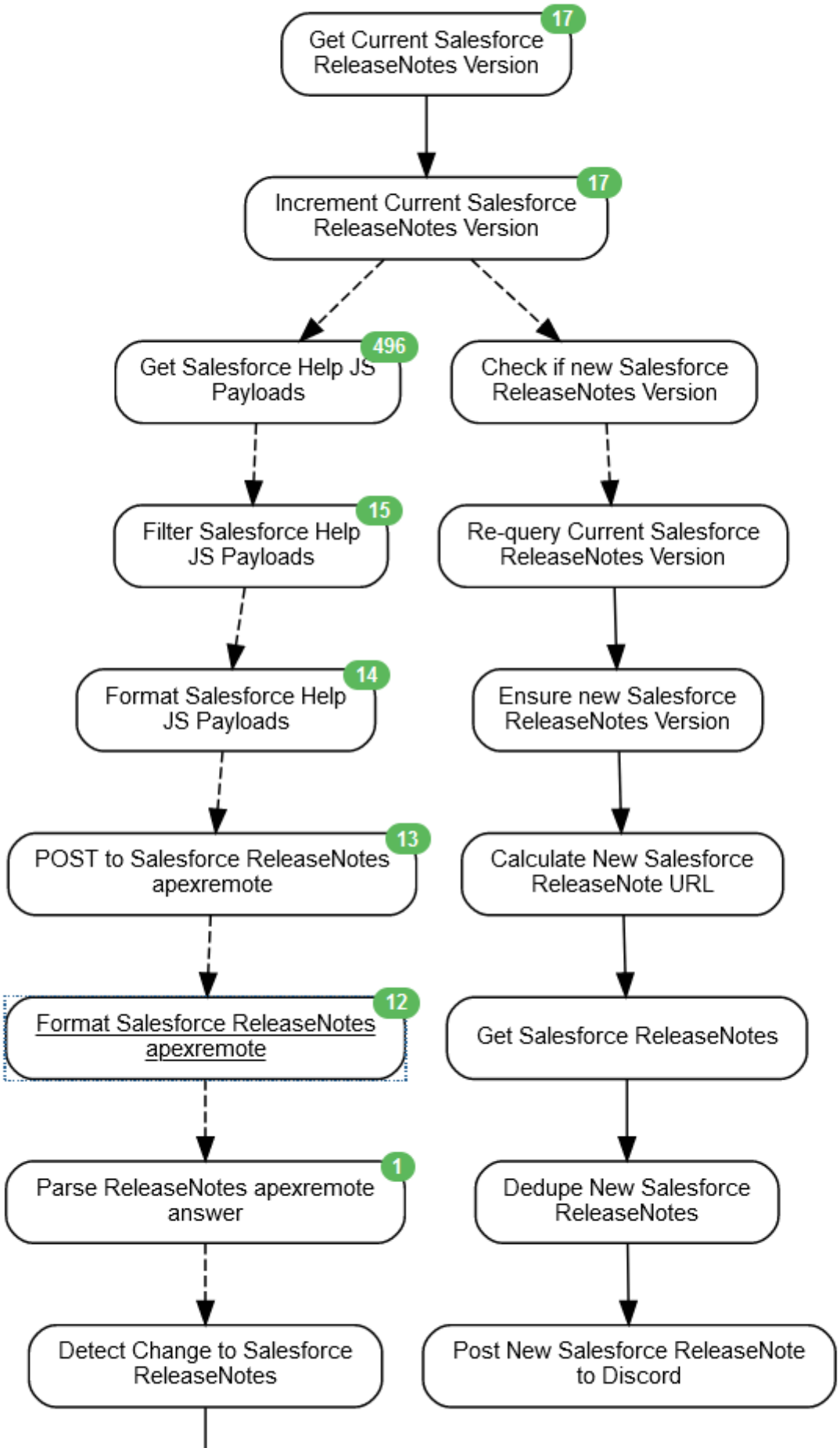
I then simply fed that to a change detector agent, which if it does detect changes, pings me on Discord.

TL;DR

The new SF help website isn't script-friendly, makes subscribing to changes REALLY hard, and its function is honestly weird - I do not see why we are hdiing release notes behind remoting calls protected via CSRF.
But well, I have a funcitonning notifier and change detector now.
Pic related

Agent Event Flow

Back



Revision #2
Created 2021-04-08 08:29:48 UTC by Windyo
Updated 2021-04-08 10:34:23 UTC by Windyo