**The Data Engineer’s Guide to Salesforce**

How to perform a Python-based data migration into Salesforce CRM

After spending this past year living my life in the AWS Cloud, I was introduced to this impressive, powerful, and (for me) foreign tool. As a data engineer, and not a Salesforce professional, there was a lot to learn. I’ve written this guide to be “the standard repository for all knowledge and wisdom” — or, rather, a comprehensive introduction — to Salesforce for my fellow data engineers.



Image courtesy of <https://hipwallpaper.com/hitchhikers-guide-to-the-galaxy-wallpapers/>

**Salesforce from a Data Engineer’s Perspective**

Salesforce is a SaaS-based customer relationship management (CRM) tool. Earlier in 2020, it was announced by International Data Corporation’s Worldwide Semiannual Software Tracker that Salesforce was the #1 CRM provider for the seventh year in a row. In 2019, Salesforce increased its market share by more percentage points than the other top 13 CRM vendors *combined*. The numbers don’t lie:

Chart, line chart

Description automatically generated

Salesforce market share growth. Courtesy of: <https://www.salesforce.com/company/news-press/press-releases/2020/05/salesforce-idc/>

Basically, if you haven’t yet heard of Salesforce, it is likely that you soon will.

Until recently, I had only heard of this CRM powerhouse, but I hadn’t properly worked with it. Data engineers don’t often get the chance to work with Salesforce, although I anticipate that will change in the near future. All of that growth in market share is indicative of migrations, and migrations point to (you guessed it) DATA!

**Migrating to Salesforce: Build or Buy?**

For a one-time migration, you may grimace at the price tag associated with purchasing a product for the job. Fortunately, there is an alternative: it is quite feasible to do a code-based migration into Salesforce.

While there are a number of ready-to-use tools which can aid in moving data into Salesforce, these are not always the best choice for your data migration. This decision ultimately comes down to time and money.

Migrations typically start with a discovery phase where data to be moved into the new system is identified, order of migration steps is established and timelines become clearer. After discovery, everyone has a better idea of they’re *really* working with.

If there is a well-defined dataset to be migrated which maps well into Salesforce’s data objects, the migration is well-suited to a code-based migration. “Well-defined” refers to the object mappings themselves, the volume of data, and how often the data is updated. However, if the data is ambiguous or additional discovery is required for the data itself, it may be worth looking into tools which can do this as part of the migration.

The next logical step is to determine how to actually get everything from the old system to the new one. If data can be easily extracted from the legacy system via APIs, especially leveraging bulk APIs, it is feasible to use code for the migration.

Alternatively, if it is decided that a tool should be purchased in order to do the migration, an evaluation phase typically follows the discovery phase. This timeline may vary, but it will likely be weeks of discussions with sales representatives, product demos, internal discussions with key stakeholders, and financial deliberation.

Ultimately, there are a few important decisions which may determine the suitability of a migration for being code-based or tool-based:

1. How well the data is understood, as well as how well-defined the dependencies within the data are understood and mapped.
2. The feasibility of programmatically extracting the data from the source system and mapping it (as well as loading it) into Salesforce.
3. Whether the migration is a one-time migration, or will be undergone for a significant time period (we’re talking several months, or even years) into the future.

If the data and its dependencies are well understood, and you are solving for a one-time migration use case, programmatically moving the data out of legacy systems and into Salesforce, leveraging code instead of purchasing a tool, is the better option.

This is where data engineers come into play. The team necessary for this migration will ideally include one or more Salesforce SMEs and one or more data engineers. The former will be key in guiding the migration (and the data engineer), and the latter will be responsible for the movement of that data.

As a data engineer (and definitely *not*a Salesforce SME), I have put together this introductory guide for other data engineers working with Salesforce. This will make life much easier both for you and the Salesforce SME you are working with.

**Reference Use Case**

The code-based migration I was a part of involved migrating data from Zendesk Support into Salesforce Lightning. Zendesk offers an incremental API for certain data types, and this was extremely useful in egressing data from the system in an efficient manner.

There were tens of thousands of each type of object to be migrated. However, among the objects being migrated, there were comments and attachments — which were not compatible with the Zendesk API. Throw in the large volume of data associated with these object, we had to identify another approach. Therefore, threading was used for these objects, which I describe later on in the guide.

This migration was performed over the course of a weekend, from Friday night to Sunday evening. Before the cut-over itself, Salesforce Sandbox instances were heavily relied upon to test the migrations of each object. The Zendesk system was “frozen” a few hours prior to the data migration’s start.

**What to Know Before Embarking on a Code-Based Salesforce Migration**

Let’s start with an overview of Salesforce’s data model, review some tools to make the job easier, and share lessons I learned to save you hours of pain.

· [**Salesforce Basic Object Types**](https://medium.com/slalom-build/the-data-engineers-guide-to-salesforce-78772d2a627b#a686)  
· [**Salesforce Record IDs**](https://medium.com/slalom-build/the-data-engineers-guide-to-salesforce-78772d2a627b#3cda)  
· [**External IDs**](https://medium.com/slalom-build/the-data-engineers-guide-to-salesforce-78772d2a627b#72eb)  
· [**Importing and Exporting Data**](https://medium.com/slalom-build/the-data-engineers-guide-to-salesforce-78772d2a627b#3a69)  
· [**Additional Migration Tips & Tricks**](https://medium.com/slalom-build/the-data-engineers-guide-to-salesforce-78772d2a627b#6d40)

**Salesforce Basic Object Types**

Salesforce stores its data in what it calls “objects”, which are physically represented as relational tables containing attributes. The two most common object types are:

1. **Standard objects:** come “out-of-the-box” with Salesforce
2. **Custom objects:** created by the business for specific needs

This article will focus on the first type of object, Standard Objects, as they are fundamental to anything you do in Salesforce. The following table is a list of the most common Standard Objects in Salesforce.

**Users**

The people within your Salesforce organization. In order for someone to have access to Salesforce, they will need a User set up, allowing them to login and, well, use it.

**Accounts**

Typically companies that your organization does business with, but can represent people too (think B2C). There are two different types of Accounts: Business and Person. I will explain Person Accounts lower down, as they will make more sense after explaining Contacts. Business Accounts represent organizations whose employees are associated with the Business Account. They typically have an associated street address, but no email address; their employees would have email addresses as part of their Account record.

**Contacts**

Employees of a Business Account. They are associated to the Business Account via a standard ID field (else they are only visible to the person who created the Contact) and Salesforce provides the capability to add custom external IDs (we’ll get into this shortly). Contacts may have an email address, phone number, or any other type of information with which you can contact them.

**Person Accounts**

Created by Salesforce after the creation of Business Accounts and Contacts, and are effectively a combination of the two. Remember the point above about a Contact needing to be associated with a Business Account? Person Accounts exist because at some point Salesforce realized that some of their customers needed the flexibility to track people independent of a Business Account. Picture this: you work for a pharmaceutical company. Many of your customers are pharmacies (Business Accounts). However, some of your customers are also individual doctors. These B2C (Business-to-Customer) relationships are prime use cases for leveraging Person Accounts.

Almost every migration will require both Business and Person Accounts, and Contacts to be migrated. Other objects typically belong to these objects (or at the very least, link to them.)

**Salesforce Record IDs**

Each record in a Salesforce organization is uniquely identified by a record ID. These IDs are 15 or 18 characters long, and consist of the following format:

A picture containing text, clipart

Description automatically generated

**QQQ:** the key prefix unique to the entity type. *Example:*001 is an Account

**PP:** Pod Identifier. *Example:* The na2 instance will use 30

**R**: Reserved for future system use

**NNNNNNNNN:** Unique alpha numeric identifier

**SSS:** Optional 3-character suffix to make IDs case-insensitive

The most useful part of the ID from a migration perspective is the first three characters, represented by “QQQ” in the above legend. These are the key prefix unique to the entity type. This enabled me to specify a record type, which was especially useful for exporting data from Salesforce.

Some of the common record types’ key prefixes are included in the table below.

Graphical user interface, application

Description automatically generated

Common Salesforce Object Type Key Prefixes, courtesy of <http://www.fishofprey.com/2011/09/obscure-salesforce-object-key-prefixes.html>

**15 vs 18 Characters**

An older version of the Salesforce API used case-sensitive 15-character record IDs. Later on, Salesforce introduced 18-character case-insensitive IDs (“SSS” in the above legend). By default, Salesforce uses 15-character IDs in the UI and 18-character IDs in the API. When you build a report in Salesforce and include the ID column, it will give you the 15-character ID. I never ran into issues with IDs, but this can get confusing; you can read more on Salesforce IDs in [this Salesforce Help article.](https://help.salesforce.com/articleView?id=000324087&type=1&mode=1)

**API Field Names**

When loading data into or extracting data from Salesforce, there is a good chance that you will be working with the API field names, and not necessarily the field labels as they appear on Salesforce page layouts. For information on where to find these field names, check out [this link.](https://help.salesforce.com/articleView?id=000327123&type=1&mode=1)

There exist some tools which make it easier for users to find the API names, instead of looking for the individual objects’ API names within Salesforce itself.

One of these tools is Workbench. Workbench is a web-based toolset which enables users to “interact with Salsforce.com organizations via the Force.com APIs”. Basically, Workbench is a streamlined means by which users can interact with Salesforce data via their web browser.

Another tool is a browser extension called Salesforce Inspector. This productivity tool is available for both Chrome and Firefox, and can be used directly in the Salesforce UI to inspect data and metadata.

**External IDs**

Salesforce describes an external ID as a field that “may be used to reference an ID from another, external system.” External IDs are custom, indexed fields that you can create in your Salesforce organization per object. It is highly recommended that you use them to form relationships between connected entities from your old system in Salesforce.

For example, if you have a Contact belonging to a certain Business Account from your legacy system, you may assign an external ID for the Business Account to both the Contact and the Account. If you search the Business Account’s external ID in Salesforce, the results will include both the Contact and Business Account. Note — external IDs are not case-sensitive.

Not only are external IDs searchable, but you can also refer to the external ID (and avoid using the record ID) while upserting records!

Diagram

Description automatically generated

How External IDs work, courtesy of: <https://douglascayers.com/2016/02/07/salesforce-power-of-upsert-and-external-ids-part-1/>

Let me explain why this is so exciting by describing two scenarios: one without using external IDs, and one using external IDs. In this example we are migrating Business Accounts, Person Accounts, Contacts, and Cases from our old system into Salesforce.

**Scenario #1 (Without External IDs):**

1. Load Business Accounts into our Salesforce organization as there are other objects (Contacts) which depend on these existing.
2. In order to migrate our Contacts from the old system into Salesforce, we need to identify the Business Accounts to which we would like to link our Contacts. Even though these entities may be linked in the old system, they use the old system’s IDs, which are meaningless in Salesforce. Therefore, we need to export a list of Business Accounts with their record IDs from Salesforce.
3. We need to perform a lookup to cross-reference the existing Salesforce Business Account record ID which should be associated with each incoming Contact from the old system.
4. Load the Contacts with their associated Salesforce Business Account record IDs into Salesforce.
5. Load Person Accounts into the Salesforce organization.
6. Cases need to be linked to a Contact or Person Account. This step is similar to step 2, except we need to export all Contacts and Person Accounts with their associated record IDs from Salesforce.
7. As was done in step #3, we also need to perform a lookup to cross-reference the existing Salesforce Contact or Person Account record ID that should be associated with each incoming Case from the old system.
8. Load the Cases with their associated Contact or Person Account record ID into Salesforce.

**Scenario #2 (With External IDs):**

1. Create an external ID field on Business Accounts, Contacts, Person Accounts, and Cases.
2. Load Business Accounts into our Salesforce organization, and map the old system’s Business Account ID to our external ID field in Salesforce.
3. Load Contacts into our Salesforce organization, and map the old system’s Business Account ID to the Salesforce Business Account external ID as well as the old system’s Contact ID to the Salesforce Contact external ID.
4. Load Person Accounts into our Salesforce organization, and map the old system’s Person Account ID to the Salesforce Person Account external ID field.
5. Load Cases into the Salesforce organization; if the Case was requested by a Contact, map the old system’s requester ID to the Salesforce Contact external ID field. If the Case was requested by a Person Account, map the old system’s requester ID to the Salesforce Person Account external ID field. It is also useful to populate the Case external ID field with the old system’s Case ID, as it may have dependencies (ie: Case Comments) that can be linked via this field.

Not only does Scenario #2 require fewer steps, but the steps involved are also much simpler than those in Scenario #1. Trust me, external IDs will make your life a lot easier than if you choose not to use them.

Follow [this link](http://winshuttle-help.s3.amazonaws.com/studio/en/connect-salesforce/help/add-external-id-field.htm#gsc.tab=0) for a walkthrough of creating an external ID for Accounts, which can be applied to any object you desire.

**Importing and Exporting Data**

Exporting data may seem like a strange thing to discuss considering this guide talks about migrations *to*Salesforce, and not egressing it. However, depending on what’s being migrated, it is sometimes useful to export data from Salesforce in order to create necessary links prior to loading data for import. Scenario #1 in the previous section provides an example of where exporting data is useful, but hopefully you can avoid having to do so many exports and lookups by leveraging external IDs.

For some migrations, however, exporting and linking existing Salesforce data to new data being prepped for import is necessary. For example, if you want to import Cases and assign them to existing Salesforce Users upon import, it is necessary to export the Salesforce organization’s Users with their record IDs and create those links in advance. My experience involved Python (and specifically, Pandas), but however you choose to create those relationships is up to you.

There are four main ways to export data from Salesforce. Tools 1 and 2 in the list below can also be used to load data into Salesforce. Please always follow security best practices when exporting Personally Identifiable Information (PII).

1. Salesforce API (programmatic)
2. Data Loader (application)
3. Salesforce Reports (internal feature)
4. Workbench (external website)

**Salesforce API (programmatic)**

Salesforce provides a few different options in terms of APIs with which you can move Salesforce data. They provide a useful guide to these options in the Help section on their [website](https://help.salesforce.com/articleView?id=integrate_what_is_api.htm&type=5).

My preferred means of exporting data from Salesforce is using the Salesforce REST API with Python, and a library called Simple Salesforce.

**[simple-salesforce/simple-salesforce](https://github.com/simple-salesforce/simple-salesforce" \t "_blank)**

[Simple Salesforce is a basic Salesforce.com REST API client built for Python 3.5, 3.6, 3.7 and 3.8. The goal is to…](https://github.com/simple-salesforce/simple-salesforce" \t "_blank)

[github.com](https://github.com/simple-salesforce/simple-salesforce" \t "_blank)

**Data Loader (application)**

Salesforce Data Loader is an application enabling bulk imports and exports of data into and out of Salesforce. Data Loader is downloadable for free, and can be used via an interface or Windows command line. Data Loader operates using CSV files for both importing and exporting.

My experience with Data Loader was via the interface, which was very easy to use. I mainly used Data Loader to import records, and less so to export. With that being said, exporting data from Salesforce via Data Loader can be helpful in order to get an idea of the size and shape of data in a quick and easy manner.

Graphical user interface, text, application, chat or text message

Description automatically generated

Salesforce Data Loader interface “home” screen

While my gut instinct was to only use the programmatic API for loading data into Salesforce, the team of Salesforce experts I was working with convinced me to use the Data Loader. Like most tools, Data Loader has pros and cons, however, ultimately I was grateful to my Salesforce colleagues. Here’s why.

Data Loader has built-in error handling, and creates both success and error CSV files on every run. In addition, if your import gets stopped at a certain row in your dataset, Data Loader saves that row number as the starting point for the next import operation.

I typically used Upsert for imports as I was leveraging external IDs for the various record types. Upserts require either the Salesforce record ID or the external ID to be used as part of the import.

[The configuration of Data Loader](https://help.salesforce.com/articleView?id=configuring_the_data_loader.htm&type=5) is crucial. It is worth developing an understanding of the various settings, as well as what makes sense for your organization. Some settings worth noting are “Batch size” and “Use bulk API”.

**A Note About the Bulk API**

Be wary of the bulk API limits of your organization in the case that you select this. The bulk API may reach its limit before expected due to background processes in an import as well as other processes happening in your Salesforce organization. If possible, disable or create bypasses for these background processes, while importing data into your Salesforce environment. Examples of background process include triggers, process builds, and workflows. Disabling or bypassing these processes means that you don’t need to worry (as much) about API limits or any sort of throttling during your import. However, if this is not feasible, you will need to be mindful while using the bulk API.

During the migration I worked on, we were unable to exclude record processing during import. Therefore, I found it was safe to use the bulk API when loading only a few thousand records, with a batch size of 2000. However, for a volume of more than a few thousand records, I would encounter a high number of errors or a complete job failure. Check your organization’s bulk API limits before using this.

If you don’t plan to use the bulk API, it makes sense to use a smaller batch size. Without the bulk API, I recommend starting the batch size at 100. This may result in a high number of failed records, which is fine. My recommendation for that is to keep feeding the new error files (with only the desired fields selected) into the Data Loader, until there are no more error files. If you find your Data Loader would perform better at a different non-bulk API batch size, feel free to use whatever works best. I encourage you to “play around” with this, in order to get a better idea of the optimal value.

For instructions on installing Data Loader, view [this link](https://help.salesforce.com/articleView?id=loader_install_general.htm&type=5) and select the article for your operating system.

**Salesforce Reports (internal feature)**

Salesforce Reports are an extremely useful tool for getting a quick idea of the record counts as well as the presence of certain attributes for a given record type, such as an external ID. Reports offer a variety of combinations of fields and metrics for exploring data and make it easy to share this information within Salesforce and via exports.

Graphical user interface

Description automatically generated

Sample Salesforce Report courtesy of <https://trailhead.salesforce.com/en/content/learn/modules/lex_implementation_reports_dashboards/lex_implementation_reports_dashboards_report_formats>

Salesforce Reports can be saved and re-run with the same reporting parameters repeatedly.

**Workbench (external website)**

Workbench enables users to query data in their Salesforce organization. It’s similar to Salesforce’s [Developer Console](https://help.salesforce.com/articleView?id=code_dev_console.htm&type=5), but allows users to export the results of their queries.

Graphical user interface, text, application, email

Description automatically generated

Developer Workbench homepage

Note, this is not an official Salesforce tool but can be useful when working on data migrations.

**Additional Migration Tips & Tricks**

This section contains a few additional pointers I picked up during the migration. Some of them are directly related to Salesforce, while others are more about the migration itself.

**Walk Before Running; Sandbox Before Production**

It’s typically bad practice to “do it live” when it comes to data migrations. (By “typically”, I mean NEVER DO THIS.) Salesforce Sandboxes are copies of an organization’s Production environment, including all metadata; sometimes, this includes at least a portion of Production data as well. Below is a comparison of the Sandbox types and their features.

Table

Description automatically generated

Comparison of Sandbox features, courtesy of <https://help.salesforce.com/articleView?id=sf.data_sandbox_environments.htm&type=5>

Sandbox environments are critical for testing migrations, settings, and other items due for Production. The different license types have different Sandbox types and quantities available for each. Sandbox licenses are hierarchical, so it is important to determine what needs to be done in a Sandbox environment before purchasing a license. If you purchase a license with insufficient Sandbox functionality, you’ll have to go through the hassle of changing your license type. For more information on Sandbox Types, check out [this Salesforce Help article.](https://help.salesforce.com/articleView?id=sf.create_test_instance.htm&type=5)

Table

Description automatically generated

Sandboxes by license type, courtesy of: <https://help.salesforce.com/articleView?id=sf.data_sandbox_environments.htm&type=5>

All of my ad-hoc, per-object testing was done using a Developer Sandbox. In order to ensure adequate space for the imported data, I deleted all imported records after each test. The cut-over rehearsal was done using a Full Sandbox.

**For a High Volume of Network Calls, Use Threading**

One of the objects I was responsible for migrating was Case Attachments. This involved downloading attachments from the old system via a non-incremental API. Without threading, this would have taken approximately 24 hours to do; this was not remotely feasible for cutover. Therefore, I was forced to use threading (and I’m so glad I was). This cut the time for this portion of the migration down to a few hours, instead of over a day in itself.

**[An Intro to Threading in Python - Real Python](https://realpython.com/intro-to-python-threading/" \t "_blank)**

[Watch Now This tutorial has a related video course created by the Real Python team. Watch it together with the written…](https://realpython.com/intro-to-python-threading/" \t "_blank)

[realpython.com](https://realpython.com/intro-to-python-threading/" \t "_blank)

An additional “hack” I leveraged was downloading all of the attachments a few days before cutover, and then only downloading net new attachments during cutover.

One critical note on threading: do not forget to use a global lock when writing contents from a thread to a file. If you forget this key step, expect to have gibberish written to your file. You’ve been warned.

**Handling Comment Body Limits**

One irritating aspect of Salesforce is the Case Comment character limit. The limit itself is quite low at 4000 characters, but this does not account for the space taken up by special characters and comment formatting. In order to capture complete comments with no data loss, but still display them in such a way that they read top to bottom required some tricky logic.

A screenshot of a computer

Description automatically generated with medium confidence

Logic to print oversized comments in readable (top-to-bottom) format

The above code creates a list of comments from the original comment field having over 4000 characters, with each listed comment containing 3500 characters. This includes a safety factor of 500 characters which isn’t strictly necessary. A variable called “total\_seconds” is also created which is equivalent to the length of the list of comments.

The code then loops through the list of comments, starting from the last one. We do this because we need to print the last comment first and therefore at the latest timestamp, so that it appears at the bottom while scrolling through Case Comments. In order to ensure that this list item has the latest “CreatedDate” the current “total\_seconds” value is added to the original create date associated with the non-split comment.

The relevant fields are mapped for this particular comment in the comment list, and then write it to our CSV (to be imported via Data Loader). After writing the data to the CSV, the code subtracts 1 second from the total seconds, and repeats this process while working backwards through the list.

Note that the above screenshot uses global locking for threading. This ensures the data is cleanly written to the destination file (“sfCaseComments.csv”, in this case).

**Leverage “Audit” Fields for Importing Data**

It is often useful to maintain date and user information on imported records via audit fields. Audit fields in Salesforce are Created By, Created Date, Last Modified By ID, and Last Modified Date. These fields can be specified using Data Loader upon import or insert; it is not possible to set these fields on update. In order to leverage this feature, “Create Audit Fields” must be enabled. For more information on setting audit field values, see [this Salesforce Knowledge Article.](https://help.salesforce.com/articleView?id=000328426&type=1&mode=1)

**Turn Off Email Before Doing Anything in Salesforce**

The final tip is the simplest, but arguably the most important. If you’re making changes in your Salesforce organization, you do not want to accidentally send emails to your customers as a result of these changes. Get into the habit of turning off email deliverability before importing or exporting any data to or from your Salesforce organization.

Graphical user interface, text, application, email

Description automatically generated

Turning off email deliverability, courtesy of <https://i.stack.imgur.com/oIyot.png>

This can be done via the following steps:

1. Go to Setup.
2. In the Quick Find box, type “Email Administration” and open.
3. Click on “Deliverability”.
4. Under “Access to Send Email”, select your desired setting.
5. Click “Save”.

Just as it is important that you turn off email before performing any activities which could inadvertently trigger emails to your customers, it is also important to turn email back on following completion of said changes. Do not forget to click “Save” in either case!

**Sign-Off**

It goes without saying that Salesforce is a massive CRM ecosystem, and this is barely the tip of the iceberg. I was incredibly fortunate to work with Slalom’s Salesforce rockstars, who provided me with guidance and support (and no shortage of patience!) along my learning journey.

Hopefully this article helps you develop at least a foundational understanding of Salesforce, from one data engineer, to another. If you have any questions, comments, issues, etc., please leave a comment below!

*A special thank-you to my Slalom teammate, Kristin Flewelling, for her Salesforce wisdom and expertise in both helping me learn this amazing platform and keeping me honest in this article!*