metazoa

Breaking Your Org into Packages with Salesforce DX and the Metadata API

Most Salesforce orgs contain a sea of unpackaged metadata. Some customizations are made by hand with the Setup Menu, others are created with Change Sets or the Metadata API. Unpackaged customizations can pile up over time and make the org difficult to manage. Too many customizations can result in slow performance, reduced agility, sluggish adoption, and added complexity. Moving unpackaged assets into packages can solve this problem and add much needed structure to your Salesforce account.

This blog discusses techniques that can help you break your Salesforce org into packages. Moving forward with Salesforce DX and Second-Generation Packaging, there are significant benefits to adopting this technology. We discuss the costs and benefits of packaging, the different types of packages that are available, various techniques to identify connected assets, and the technical methods for creating new packages. In conclusion, we discuss a practical roadmap and the best practices for breaking your org into packages.

Packaged or Unpackaged?

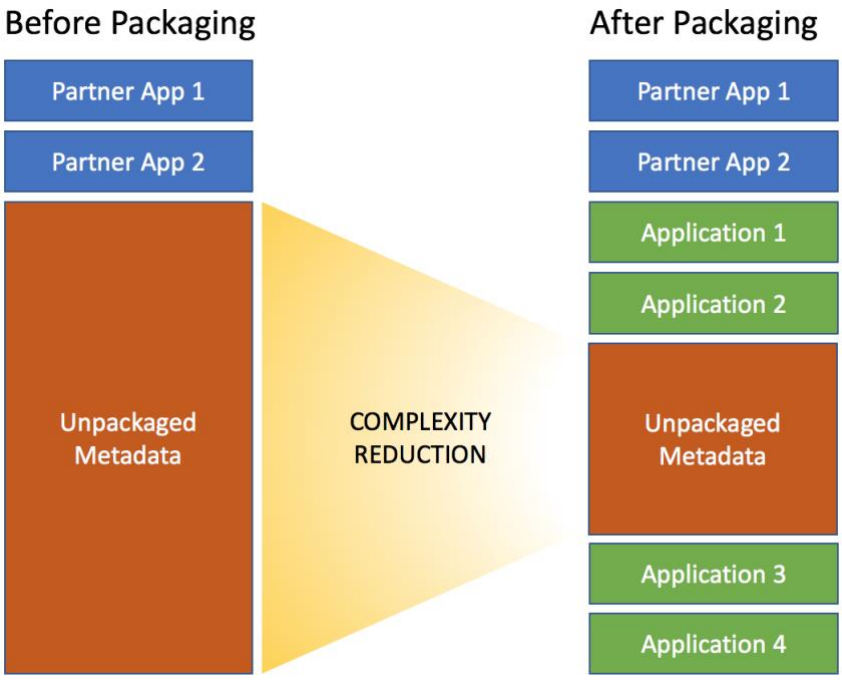
Just to be clear, all of those unpackaged assets in your org aren't going to magically disappear. The standard objects, tabs, applications, and other metadata provided by Salesforce will live on as unpackaged assets. Standard objects can't be moved into packages, anyway. All of the reflected customizations from managed packages will appear among the unpackaged assets with a namespace. Unmanaged and unlocked packages will also pour assets into the sea of unpackaged metadata. If the unpackaged assets aren't going away, then what is the point of organizing them into packages?

Here is an analogy that might prove helpful. In the software development world, source code is compiled into object code. Unpackaged assets are kind of like object code. Understanding the architecture and debugging problems at the object code level is difficult because all of the unpackaged assets are mixed together in a flat folder, and Apex Classes from managed packages are obfuscated. Our goal is to select related assets from the unpackaged area and organize them into packages. Packages are kind of like source code. They group related assets together and make everything easier to understand.

Costs and Benefits

There are many advantages to adopting packages. Bugs and problems are isolated to the package instead of being spread across the entire org. When a project needs to be upgraded or replaced, the package makes entangled assets easier to identify. Packages are the foundation for agile development, enabling smaller groups of developers to focus on more isolated sections of code. Packages can be individually tested in a Scratch Org or developer account. Modular packages can be used to enable multiple orgs and partners. Packages lower cost, increase flexibility, reduce complexity, and improve time to production.

But breaking your org into packages is not for everyone. Smaller orgs can often be managed effectively with nothing more than the Setup Menu. Companies that are not customizing Salesforce in a significant way may not feel any pressure to reduce complexity. Refactoring various metadata assets and Apex code into modular packages will be a significant undertaking, and companies need to be sure that the cost is worth the benefit. This is not an all or nothing proposition. The transition to modular packages can take place gradually and should become one of the priorities in your change and release management process.



Managed or Unlocked?

Second-Generation Packages are deeply integrated with Salesforce DX and have some powerful new features. Under the circumstances, enterprise customers may decide to adopt both technologies. However, there is nothing wrong with using old fashioned unmanaged packages for breaking up an org. Old fashioned managed packages are probably not the right choice, they have special capabilities for ISV deployment on the AppExchange.

A major benefit of Second-Generation Managed Packages is that you can have multiple packages with the same namespace. Extension packages can simply use the same namespace, allowing you to easily share Apex code. You can use public Apex classes across packages rather than global Apex classes, which are exposed to everyone.

But once a managed package is released, you can't change some component attributes of its metadata in subsequent versions. Also, Salesforce recommends that you work with a single namespace to avoid complexity managing components. In that case, you can only see the difference between packaged and unpackaged data. You can't identify individual packages. When you install an ISV product in your org, the unique namespace prevents collision with other assets. But if you are breaking your own org into packages, there are other ways to avoid namespace collisions.

Taking all of these considerations into account, breaking an org into unlocked packages is probably the best choice for most companies. When you migrate metadata from your org into a package without a namespace, the API names don't change. If the API names do change, you will need to remap data records and fix other problems. Adopting unlocked or unmanaged packaging prevents broken dependencies during the transition and gives developers more control over how to organize and distribute the parts of an application.

Old fashioned unmanaged packages are being replaced by Second-Generation Unlocked Packages. These packages are integrated with Salesforce DX and can optionally have a namespace. Unlocked packages with a namespace provide a middle ground with some of the advantages of managed packages but more flexibility like an unmanaged package. At some future date we may see Second-Generation Locked Packages as well. These are like the unlocked packages but prevent end user tinkering when deployed.

Salesforce DX

Salesforce DX is deeply integrated with Second-Generation Packages. You can create package versions, install managed packages, pull source from a Scratch Org, and convert metadata folders into projects. A Salesforce DX project is a file and folder format that defines the source code for a Salesforce org on the local machine. This technology provides source-driven tools for building packages, syncing them with repositories, and deploying them into Scratch Orgs, Sandboxes, and Production accounts.

Here's a thought experiment. Let's say you break your org up into a bunch of unmanaged or unlocked packages. This is easy to do, just go into the HTML interface and move various assets into unmanaged packages. Without namespaces, most customizations will continue working as before. But how do you know that each package can stand alone, is independent of the other packages, and suitable for continued software development? The only way to do this is to deploy and test each package independently, and this is where Salesforce DX comes in.

Each Salesforce DX project can be tested in isolation with a variety of environments. The **org:shape:create** command can spin up Scratch Orgs with different characteristics, and even mimic your Production org. The **source:pull** and **source:push** commands can synchronize the local project with the Scratch Org. The **data:tree:import** command can populate that org with test data. When the project is working correctly, you can create Second-Generation Packages and Package Versions in your Dev Hub, and these packages can be installed into Developer Editions, Sandboxes, or Production accounts as needed.

Mapping Dependencies

One extremely useful technique is to retrieve a metadata image of your org and look for connected assets. For example, if you have a Custom Object that belongs to a distinct project then go search the metadata for this object by name and you will probably uncover dozens of other dependencies, including Custom Tabs and Applications, Permission Sets and Profiles, Workflows, other Custom Objects, and Apex Classes. Gather these assets together, and you have an awesome starting point to build a package.

When you attempt to create a package, the Metadata API may provide error messages about additional assets that must be included in the deployment. Change Sets will also suggest related assets that are required. Sometimes the dependencies can snowball, demanding more and more of your org for the package to be viable. In this case, you need to look again at how you are factoring the assets and see if some references can be simplified.

Package Organization

One natural metadata asset to organize each package around is the Custom Application. Each Application will likely have some Custom Tabs and Objects. Starting from there, you will find connected Translations, Workflows, Page Layouts, Apex Classes, and other assets. Lightning Bundles and Apex Pages are also useful starting points to build a package.

Second-Generation Packages are often used to organize Apex Classes. This scenario closely matches traditional software development, where source code repositories, continuous integration, and library management are common practices. Apex Class packages can be installed in test orgs and Production environments. They can be organized by function, layer, purpose, namespace, team, or whatever scheme makes the most sense. Public Apex Classes can make calls across packages, so they can all work together.

You should be able to explain each package in a single descriptive sentence. The assets in the package should have related names. Unique key words and other naming conventions are a good way to associate the assets, and aids in the metadata search process described above. The goal is a logical package that hangs together and implements a specific project or function. Avoid obsessive code refactoring. A few duplicate scripts can be preferable to lots of entanglements between packages.

Converting Packages

To divide an org into packages you will need the ability to convert between unpackaged metadata, managed packages, unlocked packages, and Salesforce DX projects. Some of the most useful conversions are listed below, and then discussed individually.

- Downloading related metadata assets into metadata folders
- Converting between metadata folders and Salesforce DX projects
- Converting a managed package into an unmanaged package

Downloading related metadata assets in your org into a metadata folder on your desktop is fairly straightforward. To do this, use the Metadata API to retrieve the related assets. Salesforce DX has some commands that simplify downloading metadata. If you want a few packages from your org, or you have the desired assets listed in a Package.xml file, then use **mdapi:retrieve** command. You can also construct a custom package in a Sandbox with the Salesforce HTML interface and retrieve that the same way.

You can turn a metadata folder on your desktop into a package. Change the folder name from "Unpackaged" to the new package name and add the **fullName** parameter to the Package.xml file. Now those unpackaged assets look just like a downloaded package. You can use the Metadata API to deploy those assets to another org as a package for testing.

Converting a metadata folder into a Salesforce DX project and vice versa is also easy. The Salesforce DX command **mdapi:convert** will translate a metadata folder into a Salesforce DX project. Likewise the command **source:convert** will translate a Salesforce DX project back into a metadata folder. The Salesforce DX project folder is the right foundation for package testing, deployment, and archive.

If you need to make extensive edits to an existing managed package, then converting it to an unmanaged or unlocked package is a handy trick. This technique avoids the editing limitations that managed packages have. The package can be rebuilt with a new namespace after all your edits. As long as other companies or partners are not dependent on the package, an enterprise developer can install the package with a new namespace. Don't forget that data records may need to be remapped, and that other dependencies on the package namespace can break.

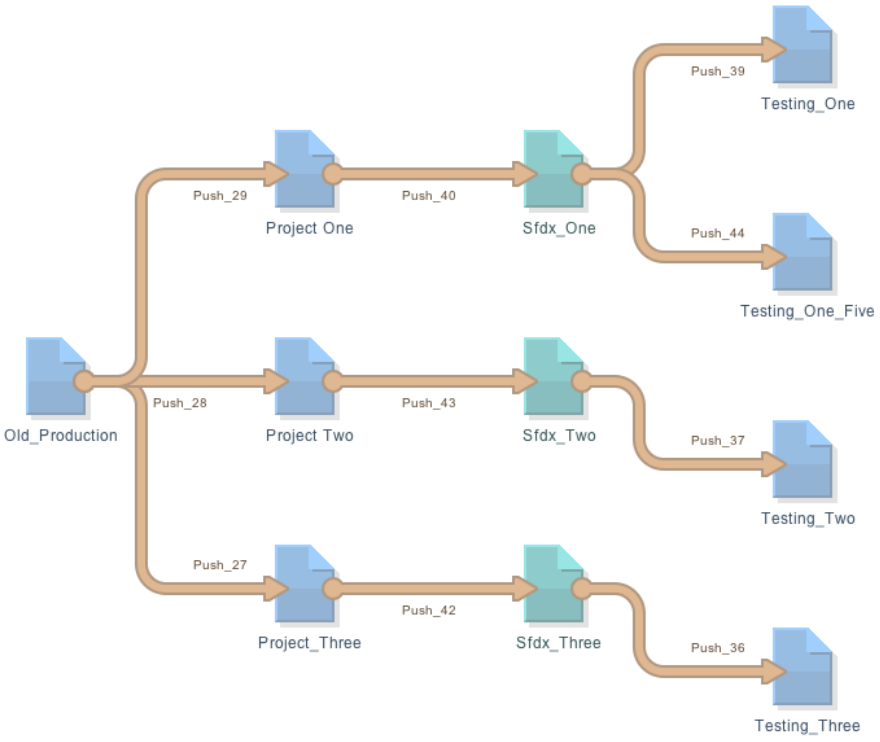
To convert a managed package to an unmanaged one, retrieve the managed package with the Metadata API. The retrieved assets will not have a namespace, so they can be deployed as an unmanaged package to a new org. Alternatively, you can grab the reflected assets and strip the namespace prefix. One problem migrating managed packages is that Apex Classes are obfuscated, they have to be refreshed from source. Second-Generation Managed Packages may solve this problem moving forward.

The Big Picture

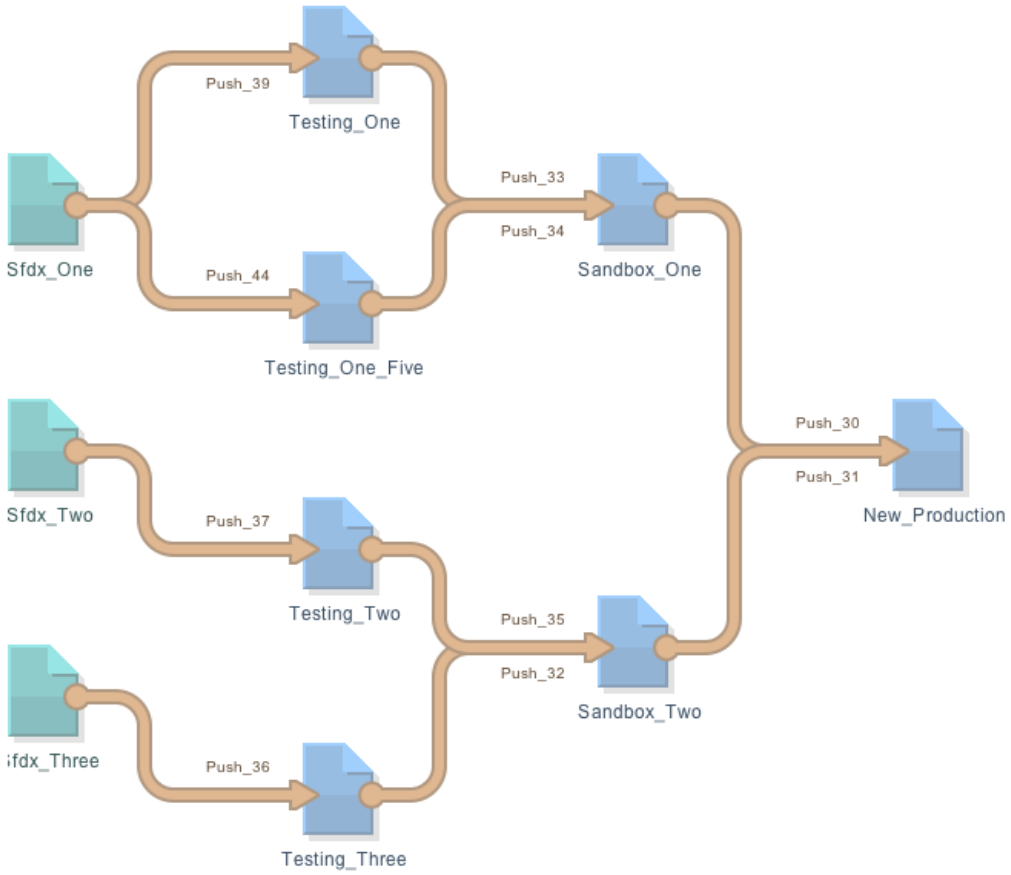
Taking a step back to look at the big picture, here are the practical steps for breaking your org into packages, and installing the new packages into Production:

- Identify related metadata assets that would make good packages
- Convert these related metadata assets into Salesforce DX Projects
- Test the Salesforce DX Projects with appropriately configured Scratch Orgs
- Create Second-Generation Packages and Package Versions in your Dev Hub
- Install these Packages into Sandboxes for testing or Production for deployment

Breaking an org into packages can be done all at once or take place over time as part of your change and release management process. Once everything is organized into packages then future development can continue along with that model. The Production org will be divided into packages that must be individually tested. Salesforce DX is an excellent destination for each package and provides agile tools for testing in Scratch Orgs with different configurations. The flowchart for breaking up your org might look something like this:



When you have built Salesforce DX projects that are working correctly, you can create packages in your Dev Hub and then install the packages into Sandboxes for testing or Production for deployment. The Metadata API can be used to deploy other unpackaged assets. You can also use the Metadata API to move Second-Generation Unlocked Packages into Production. This has the advantage of automatic roll-back if there is an error. Be aware, you cannot yet move unlocked packages with the Metadata API if they have a namespace. Hopefully this limitation will be fixed in the future. The diagram below shows an example release flowchart.



The Production org is the source in the first diagram, and the destination in the second. Packages are divided out and tested individually. Then they are recombined into Sandboxes and deployed back into Production. The functionality of the org will not be changed, but now the ability to maintain the org and implement new projects has been dramatically improved.

Bill Appleton
CTO Metazoa