IdPUpgrades

A Suggested Strategy

A sound strategy for most sites is: get your 1.3 house in order **first**, including making any desired metadata changes. Then do a controlled software upgrade on your own schedule with the ability to roll back during the early phases. If you can pull this off, and I believe most sites can, you will be able to upgrade with zero downtime and not lose the ability to unilaterally move the IdP between 1.3 and 2.x until you deploy a feature not supported by 1.3.

As should be clear from the story here, the situations for IdP sites can vary considerably based on how you have managed your interactions with SPs. Please use the Shibboleth and federation community mailing lists to ask questions and report your own successful migration strategies via the lists or this wiki.

Emulation

The key to this approach is *emulation* of a 2.x IdP by your existing 1.3 IdP. The 2.x IdP cannot easily be made to look as if it were a 1.3 IdP. I would not say it's impossible (frankly I don't know), but I would say that it's *simpler* to go in the reverse direction. In most cases, a 1.3 IdP can be made to operate with the same default/accepted behavior as the new software, at least from the point of view of an SP. In particular, its paths and locations for responding to requests can be altered relatively easily to match the 2.x software.

> ✓ Part of emulation naturally usually also includes using the same entityID and credentials on both the old and new installations. From an external point of view, they are identical except for the endpoints. If desired, it is possible to change the credentials, since the metadata that contains the new endpoints can contain the new credentials as well. But since there are often SP configurations and links/bookmarks that reference the IdP's entityID, it is usually a terrible idea to change your IdP's entityID. I would also note that changing the entityID can affect the values of certain attributes, such as `eduPersonTargetedID`, as well.

Changing Endpoints

There are at least a couple of different approaches to getting a 1.3 IdP to run on endpoints that look like 2.x:

- URL Rewriting
- Running a Second Copy

Starting the Migration

Once the emulation phase is done and tested, you can alter any metadata you publish to reflect these new endpoints and you can run **both** the old and the new 1.3 configurations at the same time (more on this below). Having done this, you're now in a state I would refer to as "in-migration", during which there will be SPs accessing both the old and new copies. The challenge of this strategy is whether you can minimize this period (or at least whether that period of time is one you can live with).

In my own case at OSU, that period is literally years. We have had two instances of 1.3 running since early 2008 and I don't expect to complete the switch until late in 2009. The reason is not because this is a bad approach but because I made a big mistake by letting SPs here locally install a metadata file I created and get away without installing refresh scripts. As a result, I am at their mercy in getting that metadata changed, and with over 100 SPs, the implication is obvious.

As part of our migration, we are upgrading SPs to 2.x (which has better metadata management options) or in some cases just re-pointing the older software to the InCommon federation metadata and getting refresh scripts in place. The federation metadata now reflects the changed endpoint information that will be consistent between 1.3 and 2.x. So as SPs make this correction, they complete the necessary migration process. If I had started them with the InCommon metadata instead of my own, this migration could have been only a few weeks long. Lesson learned.

Running Simultaneously

The key to the "in-migration" period is the ability to run two copies of the 1.3 IdP at the same time, one on the original endpoints and the other on the changed endpoints that are consistent with the new software. The effort and complexity is in making this happen with the expected user experience. At most campuses, this is going to require that the SSO experience be seamless between SPs using the old endpoints and those using the new ones. Of course, if you decide that maintaining SSO through the migration period is not essential, you have it easy, you just ignore the problem. This may be acceptable if migration is expected to be quick.

Assuming you do require SSO, how this is accomplished depends on how you perform authentication for the IdP, and of course with 1.3, that choice is outside the IdP and so is quite difficult to make assumptions about. This may be simple, or I imagine in some cases impossible, and a dealbreaker for some sites.

One simple case is for those who have a separate SSO system like CAS or Pubcookie deployed in front of their IdP. In that case, there's no problem because SSO will be automatic no matter how many copies of the IdP you have or which one is used.

Another relatively simple case would be those relying on a Java container for authentication and who can deploy both copies of the IdP in the same container. Most containers, or at least Tomcat certainly, have an option to allow SSO between servlet contexts. Enabling this should allow authentication to one of the copies to apply to the other.

Still another case (the one I actually rely on) is a combination of Java container authentication and the built-in SSO cookie supported by the most recent versions of the 1.3 IdP. This cookie feature relies on an encrypted cookie that is set once a user logs in and can be read by other IdP instances (including replicated copies in a cluster) to recover the user identity without requiring a login, basic SSO stuff.

In this case, you have to run both the old and new IdP instances on the same server(s) using the same hostname, so that the cookie can be read. Additionally, you may have to customize the "path" attribute of the SSO cookie set by the IdP so that it will be visible between the two copies. Normally the path will be based on the servlet context, so this must be expanded to make the cookie more accessible, by modifying the SSOHandler.java source with a setPath("/") call, a simple one line change.

Completing Migration

Once all supported SPs are safely relying on the new metadata, there should be no traffic into the old IdP instance on the original endpoints, something you can verify easily with logs to find stragglers. Congratulations, you're done with the migration and can turn off the old copy.

Upgrading

Recall that the point of this exercise was to get your production IdP migrated to Shib 2.x. That's now extremely simple (apart from actually installing and configuring it). You can deploy and test a new installation that matches the production IdP in every respect. Even back-channels can be tested sufficiently using local SPs with modified DNS information.

The key is that the thoroughness of the testing process is really up to you because you now control the exact timing and duration of any switchover. In fact, you may well wish to slip the 2.x IdP into operation for short periods to evaluate the results before you make a full-time change. You don't need to tell anybody you're doing this (other than as a point of information), and you can stop the test any time you need to because **the metadata is the same for the two IdPs**.

In my opinion, the simplest way to perform the switch is to install two separate copies of your Java container, and use one for 1.3 and the other for 2.x. To swap them, you just stop one and start the other.

The Point of No Return

The "no turning back" moment for your upgrade is once you have an SP relying on a feature or behavior that only the new software supports. The most visible change of course would be to publish metadata for the IdP with SAML 2.0 support included, since this may cause SPs that support it to switch over. At that point, reverting to 1.3 would break those SPs unless you can undo that metadata change, and that's exactly the situation we're trying to avoid here. So don't make this kind of change until you're ready to make the move for good.

A Possibly Inadvisable Approach

Many sites use (or attempt to use) this strategy for migrating their IdP from one major version to another (e.g., from Shib 1.3 to Shib 2.x):

- Set up a new installation on a new server with a new production hostname, a new entityID, certificate, etc.;
- Test the new IdP using SPs set up for the purpose, or other nearby SPs that can be convinced to try it;
- Once it's tested, update the metadata in federations you participate in to point to the new IdP;
- Fix whatever broke.

I don't wish to disparage that strategy if it works for you, but I don't think it's a particularly good one, mainly because of that last bit, and also because if something goes wrong, it's a difficult situation to correct in an organized way.

In general you have no explicit control over when the updated metadata will get installed by any particular SP. It may take days. You cannot revert your metadata to quickly move back to the old IdP because it will take time for that backed out change to propagate to federation SPs also. It's failure by water torture. There may even be hopeless SPs that aren't updating their metadata regularly at all, though frankly they're not worth caring about. (If they don't care if it works, you shouldn't.)

Obviously, most of these concerns are more relevant when you're involved in a true federation with SPs that don't live in the same organization, though frankly I don't feel able to dictate to the SPs in my organization when it comes to making changes on my schedule. I probably couldn't even reach the right people if I needed to.

Note also that what you might like to register the new IdP in your federation(s) in order to test it with federation SPs. This doesn't work very well in practice, even if the federation supports it. There may be problems for users knowing which IdP to use. High-volume production SPs will be unlikely to want to test with your IdP. For SPs that implement their own discovery mechanism, you may not have any control over which IdP they use. You also won't have access to their logs, which you need to diagnose problems when testing. So testing is much more likely to be successful when handled outside of the federation.

Real life experiences

University of Washington's migration