Migrating from Earlier Versions

FastX 3.2 uses a generic database architecture that is not compatible with 3.1 and below. You should export your 3.1 configuration before upgrading to 3.2. If you are updating from 2.X, please first migrate to 3.1, then follow these instructions. Users can download 3.1 by going to https://www.starnet.com/download/legacy.php.

Migrating the configuration before installing 3.2

Migrate the configuration (bookmarks, profiles, admins, load balancing, etc.) from the FastX 3.1 database with these steps:

a. Shutdown the fastx3.1 service: **systemctl stop fastx3**
b. Download https://www.starnet.com/files/private/FastX3/export-3.1.tgz
c. Extract it: **tar xf export-3.1.tgz**
d. Export the 3.1 configuration: **export-3.1/export.sh**

This will convert the 3.1 data into the 3.2 “standalone” format in the directory /usr/lib/fastx/var/local/store. If you will be running a standalone (not clustered) version of FastX, you are done. If you want to migrate the standalone information to the cluster, see the clustering guide.

Changes in FastX 3.2

New Clustering Architecture

FastX 3.2 introduces a new architecture to resolve several issues with previous versions.

FastX 2.X clustering which was based around mesh networks. While this architecture made it easy to set up and configure, it would create stability issues and was very inefficient in transferring data.

FastX 3.0 introduced a distributed database architecture which resolved the issues of version 2. The distributed database was fault tolerant and could easily distribute updates across a cluster. However, the nature of database replication in a distributed database uses an append-only architecture which would lead to increased storage requirements. Furthermore, the larger storage would also cause slowdowns in the database itself which would prevent scalability.

FastX 3.2 moved the database into a central location. The new database is not revision based (fixing the increased storage and slowdown issues) and instead overwrites the existing data. In a real-time system such as FastX this is preferred. In one stress test, a FastX customer launched and terminated 100,000+ sessions over the course of a week to measure any slowdown from creating new sessions. The first session ran just as fast as the 100,001 session.

CouchDB replaced with more efficient databases

CouchDB was designed to be a fault tolerant, distributed, master-master replication database. CouchDB is used widely and has several features that make it a good choice for a clustering system such as FastX, namely multi-master replication.

However, upon further real-world usage, we found that the drawbacks of CouchDB outweigh the features. FastX is a real-time system with many updates per day. FastX periodically updates the database with new information. Every 30 seconds for example, the database is updated with new information regarding the state of the server. Every connection, disconnection, termination and change to a session updates the database. CouchDB stores every revision made to the database by default. Furthermore CouchDB saves any conflict that occurs when the same database document is updated simultaneously. Given the thousands of operations that can happen in a cluster, this amount of data can grow exponentially. And every time a new cluster member is added, the new member must download this information to sync with the rest of the cluster members.

This method proved inefficient at scale. Instead FastX now uses two database systems by default. NeDB and MongoDB. NeDB is a javascript based database that is an in memory database that writes to files for long term storage. It is small, fast, and efficient for small datasets. We use NeDB for standalone installations, and for local only storage when in a cluster.

MongoDB is a standard NoSQL database used for large datasets. It is well maintained and well documented. We use MongoDB as the central repository for all cluster configuration as well as communicating information such as server updates and storing sessions to all the cluster members. It is much more efficient than CouchDB in our use case.

New Database Formats

FastX 3.0 and 3.1 used a shared database to distribute information across all cluster members. In order to simplify replication, the majority of data was in one large database named shared. Extra effort would have to be made to filter out data that was not related to the data in question (for example get all the servers).

FastX 3.2 moved to a centralized database server. This new method allows FastX to use multiple smaller collections of related data. For example, one collection can be only the servers, another only the bookmarks. These micro collections have the advantage of easily updating and downloading all the information from a specific collection without bothering the other data. This also allows different collections to be located on different systems closer to where it is needed.

NeDB Database Format

Collections that will only be used by the local compute node are stored locally on the system. They are stored as files and loaded into memory. FastX uses a standard format for these database files.

These files are located in */usr/lib/fastx/var/local/store/_collection_name_.db*

The files are human readable with the format
One JSON object per line with a required **_id** parameter.

If there are multiple lines with the same **_id** parameter, the latest line takes precedence.

Separation of Standalone and Cluster Compute Nodes

FastX 2.X was designed as a standalone system with clustering capabilities added on. Configuration was never shared across a cluster meaning configuration could easily get out of sync.

FastX 3.0 was designed as a cluster first system that could be reduced to a cluster of one making it a standalone system. This caused extra overhead on any system that was not in the cluster.

FastX 3.2 makes a distinction between standalone and cluster formats. Standalone and cluster formats use different locations to store information and will not interact with each other. You can even install a standalone server and a cluster compute node on the same system!

To convert a standalone system to a cluster member, simply add the broker.json file of the cluster to */usr/lib/fastx/var/config/broker.json*. The standalone system will then be part of the cluster. The system will take the configuration from the cluster configuration rather than its local configuration (except the parts that are always local).

You can revert to a standalone system by removing the broker.json file, and restarting the service.

New Cluster Manager Service

FastX 3.0 and 3.1 installed CouchDB with each system. This was used to get the benefit of the distributed database without having a centralized database. However, this would put extra load on each of the compute nodes. Each compute node would have to run FastX sessions, as well as do the processing and management of synchronizing the CouchDB database. Changing the architecture removes this extra load from the compute nodes so that they can spend more time handling the running of FastX sessions.

Override.json removed

Override.json was a json file that admins would use to dump information into the CouchDB database. Admins would put this config file in /usr/lib/fastx/var/config/override.json and the local system would read it, hash it and update the database with the configuration if it did not match the current hash.

In 3.0 and 3.1 every compute node installed and shipped with a CouchDB installation. Dumping override.json was a simple way to get this information into the synchronized database. However, since everything was distributed, it could also cause the database to be overwritten since every compute node could blindly update the database like this.

FastX 3.2 changed the architecture by having a centralized cluster manager. The database can now be updated from anywhere through the cluster manager, not just a compute node. This makes it easier to update the database in real time, without having to shut down a compute node.

Admin.json removed

admin.json was an interim way to populate the admin on install. After that it was ignored. The architectural changes makes this obsolete and not needed. In version 3.2 and later, permissions can be set in permissions-store.db

Suggestions.json removed

Suggestions.json was used in previous versions as a fallback for users who do not run the web server. The web server always ignored this file. In 3.2 and later bookmarks can be populated in system-bookmark-store.db

Service Based Architecture

FastX 3.2 introduces a service based architecture to FastX. We have separated services out from the monolithic architecture of previous versions so that things can be more fault tolerant and take up less resources on compute nodes. This also allows for hot swapping and reloading of updates without having to shut down your services allowing for greater uptime.

New Fine Grained Permissions

FastX 3.2 expands the permissions available to administrators that were implemented in previous versions of FastX 3.X. Admins, Mangers, Full Users, and Disabled Users can now be assigned by either user name, or groups.

Admin Group defaults to wheel

In earlier versions, the FastX installer would ask which group wanted to be the admin group. This could cause issues if there were no user groups. Then FastX also reimplemented admin users. In 3.2 the default admin group is "wheel" since these groups typically have sudo privileges.

Superusers can always change this after installation
permissions can be set in permissions-store.db