Starting Sessions

Background

FastX is an application for starting, connecting to, and managing FastX X Server sessions. Without sessions, FastX is of limited usefulness. FastX goes through several steps in order to choose, configure, schedule and ultimately start a session. Some time should be spent discussing the action of starting sessions to allow administrators to make better decisions in how to integrate FastX in their environment.

Audience

This document is intended to give administrators better understanding of the procedures that are taken to start sessions. End users who wish to know how to launch sessions

Starting From the Command Line

No matter how a session is started, the final operation is running a script from the command line. The goal of all the operations in the next section is to decide on which system to start a session with which parameters. Users can run the start script directly to start a session using the following command

/usr/lib/fastx/3/scripts/start --command COMMAND --geometry GEOMETRY --name NAME --param PARAM1=VALUE1 --param PARAM2=VALUE2 ...

Admins who wish to integrate load balancers into their systems should especially get accustomed to this command. Their job scheduling script will eventually run this command.

Command Line Parameters

- *--command* (required): This is run after the DISPLAY has been started. It is typically the command that defines the session. For example, “xterm -ls” or “gnome-session”. If this is a desktop session, the session will be terminated when the command exits. If this is a

rootless session, the session will be terminated when all windows are closed. This allows commands that exit but continue to run in the background, such as gnome-terminal, to keep the session running.

- *--geometry* : For a desktop session, this specifies the **initial** geometry of the session's root window. The geometry may be changed when a FastX client connects. If this is a rootless session, this parameter must be "*rootless*".

- --name : The name of the session, as seen in the session browser. This is a shortcut for "*--param name="* as the name is a session parameter.

- --param : Session parameters are key/value pairs assigned to the session. They can be changed or deleted while a session is running. They are mainly used for information, but 3rd party applications can use them to associate them with batch system, for example.

- *--cli* (boolean) : If this is specified (it doesn't take a value), it puts start in command-line mode. Usually, start will return the session information as a JSON object and exit, but if --cli is specified, the start script will remain running in the foreground until the session exits. When running in command-line mode, output, errors, and exit code are passed from the session command.

- *--id* : While not recommended, it is possible to assign a session-ID using this parameter. Unusually, the session-ID is randomly generated and returned when the session is started.

- *--config_path* : Normally used internally, this can be used in those rare cases where there are multiple FastX instances on the same machine. This parameter is used to specify the config directory, usually in /usr/lib/fastx/var/config.

- *--profile* : FastX has profile settings, used to control the behavior of sessions. This parameter is used to name a particular profile to use. If the profile doesn't exist, the session will fail.

- *--keyboardLayout* : The keyboard layout of a session is established when the FastX client connects. Some desktop managers require the keyboard layout to be defined when they start. FastX sets the default keyboard layout to "us".

- *--userdata* : This parameter sets the "userdata" param. It is here for historical reasons.

- *--userid* : Another parameter left around for historical reasons.

- *--env* : Some parameters, such as a FastX token, cannot be securely specified on the command line. By setting --env to a file descriptor, these sensitive parameters can be given securely via a pipe as a JSON object. If the member of this object named "args" is itself an object, it is used to specify any other parameters listed above.

- *--json* : All of the parameters above, and more, can be specified in one JSON object.

Order of Operations

Web Server Enabled

From the moment the start request comes from the client, the FastX server begins an involved process in determining how to handle the API call. Each function is configurable by the administrator. The order of operations is listed below.

1. Receive API Start call on the Gateway Server
2. Check Permissions
3. Call Form Filter Function
4. Call Server List Filter Function (if form == SERVERLIST)
5. Call Schedule Function
6. Call Start or Schedule Event Function
7. Call Load Balancing Function
8. Send the information to the Session Server
9. Check if starting sessions is enabled
10. Attempt to Start the session via link/fastx-protocol
11. Attempt to start the session via sudo (if link start failed)
12. Attempt to start the session via ssh (if sudo start failed)
13. License Check
14. Session Start

Web Server Disabled

When the web server is disabled FastX goes through a much more limited start process.

1. Receive API Start call
2. Attempt to Start the session via fastx-protocol
3. License Check
4. Session Start

FastX protocol is the SSH equivalent of link. If the web server is disabled you can only start via ssh. As the amount of configuration is limited when running FastX with the web server disabled, the rest of the discussion will assume the web server is enabled.

Permissions

Users with the start permission can start sessions. For a full discussion see FastX Permissions.

Forms

Administrators can create custom forms in FastX that will appear in the start session dialog box. These forms allow admins to create user friendly options that will be added to the start data when sending the start command.

The purpose of these forms is to create a params object that will be passed with the start data. Administrators can use this params object to help customize the decision making of load balancers and job schedulers, or just to have extra information marked by the user at start. This data will be added to the params object of the session. Note the params object is mutable by the user meaning the user can be changed after the session has started. Administrators should not rely on this data after the session has started successfully.

Form Filter Function

The form filter function is a custom function that the administrator creates to determine which forms should be displayed to the user and in which order. Typically the default template is sufficient, however, administrators do have the ability to customize the function to short circuit the form process if needed.

```
async function(input) {
  // input.data -- input sent from the api call
  // input.user -- user login object
  // input.isAdmin -- is the user an admin
  // input.isManager -- is the user a manager
  // input.userGroups -- Linux User Groups of the user
  // input.forms -- array of form object

  //return a falsy value to continue
  //return a form object to pop up a form
  //return the string 'SERVERLIST' to allow the user to choose from the
available servers (dynamically generated form from the serverList function)
  return;
}
```

Return Value

The goal of the filter function is to decide the next step in form processing.

- Fasly value -- end form processing and move to the next step
- Object -- Assume it is a form object and return this object to the client to be displayed to the user
- String === 'SERVERLIST' -- special case that will dynamically generate a form based on the servers you wish to display to the user.

If the return value is NOT falsy, the API start call will return without starting the session. The return object will look like

```
{
  "stage": "form"
  "form": FORM_OBJECT
}
```

where FORM_OBJECT is the object that was returned in the filter function (or the dynamically generated SERVERLIST form). The client will remove this form from its form list and continue. When the user submits the form, the client will add the form data to the params object and resend the start command with new params data. This process will continue until the filter function gets a falsy value.

Creating a Custom Params Form

System administrations may have custom parameters that they want their users to set when launching sessions. For example, if job scheduling, a system admin may want to know if a user needs a server with a video card. The custom params form allows the system admin to use basic html to create a form that will be injected into the start script.

- Log in as an admin
- System > Sessions > Forms > Create a new form
- Save

The form allows for a subset of HTML to be used. Javascript is disabled to prevent XSS attacks. The admin can simply add in the form inputs he wants and it will appear.
FastX uses Bootstrap 4 styling which can be added to keep the look and feel consistent through the form. Valid input will be added to the params object of the start data.

Example

```
<div class="form-group">
   <label>My Custom Input</label>
   <input type="text" class="form-control" name="myCustomInput" />
</div>
<div class="form-group">
   <select name="minRam" class="form-control">
       <option value="1">1 GB</option>
       <option value="2">2 GB</option>
       <option value="10">10 GB</option>
   </select>
</div>
```

Setting myCustomInput to “hello” and selecting minRam to 1 GB will set the params object to

```
{
   "myCustomInput":"hello",
   "minRam": "1"
}
```

Note that both values are strings

data-type option

You can set a custom **data-type** attribute on a form input that will convert the string into a more useful type.

The **data-type** options include

- “number” — parse the value as a Floating Point number
- “boolean” — parse the value as a boolean
- “json” — parse the value as a JSON object

Adding data-type to our original example

```
<div class="form-group">
 <label>My Custom Input</label>
 <input type="text" class="form-control" name="myCustomInput" />
</div>
<div class="form-group">
 <select name="minRam" data-type="number" class="form-control">
    <option value="1">1 GB</option>
    <option value="2">2 GB</option>
    <option value="10">10 GB</option>
 </select>
</div>
```

Will result in

```
{
 "myCustomInput":"hello",
 "minRam": 1
}
```

Reserved names

FastX automatically overwrites the following parameter names. You should avoid using these

- “name” — sets the display name
- “icon” — sets the default icon

- “bookmarkId” — sets the bookmark id when connecting with a bookmark. This id is used in making shortcuts
- “clientVersion” — overwritten with the client version on connect
- “clientId” — The desktop client that connected
- “clientIP” — IP address the client connected from (when using the desktop client)
- “clientTransport” — The method used to connect (“SSH”, “Web” etc)

Setting Forms

Once all the forms have been created and the filter function is set, the client needs to know which forms to send on start.

Administrators can add an ordered list of forms in a bookmark that will be attached to the bookmark.

For custom start commands, the administrator can set the default forms in the System > Sessions > Forms section of the admin section.

Server List Function

The server list function is a special function that is executed if the form filter function returns the string SERVERLIST. This string tells the server to dynamically create a form based on a filtered list of servers returned from this function. This function effectively lets end users choose which server they want to connect to in a clustered setup.

Note that the form returned does not actually choose the server. It only sets a serverId parameter in the params object of the start data. It is up to the load balancer to honor the serverId parameter.

```
async function(input) {
  // input.data -- input sent from the api call
  // input.user -- user login object
  // input.servers -- array of server objects
  // input.isAdmin -- is the user an admin
  // input.isManager -- is the user a manager
  // input.userGroups -- Linux User Groups of the user

  //return a filtered list of servers for the user specified by username,
or none to disable server selection
  return;
}
```

Return Value

If the value is falsy, then continue and don't send the form
Otherwise return an array of servers from input.servers that you want to pass to the user.

Scheduling

The purpose of job scheduling is to take the start data that the user has sent from the API call and merge it with a template string that is returned from the job scheduling function. Instead of starting a session immediately, the FastX server will execute this interpolated string and any output will be returned to the user. FastX leverages this feature to allow admins to configure FastX to execute job schedulers like SLURM, MOAB, LSF etc to start sessions.

Note that in FastX terminology, Job Scheduling and Load Balancing are two different actions.

- Job Scheduling is the act of creating a template string in place of the start command
- Load Balancing is the act of choosing which server to execute the start data.

When integrating LSF (or equivalent), you will execute the template function (job scheduling) on the serverId returned (load balancing). In this instance, where the session finally gets started is up to LSF.

An in-depth discussion of job scheduling is found in the FastX Clustering Guide.

Start/Schedule Event

The form and server list functions helped compile all the data. The job scheduler function decides what will be executed. The next step is to allow the action to happen.

The Start (or Schedule if a schedule template was returned previously) Event is a custom script that the administrator can add that allows the administrator to insert any custom code that he wants to execute before the start call is made. For example, an administrator can log all the information sent to the start command.

The most common action is to either disable the call completely, or to restrict the commands. Administrators can return a modified version of the data sent which will be used as the new start data.

The Start and Schedule Event functions are two separate events and are independent of each other. Only one of the two functions is called in each API call.

Load Balancing

Load Balancing is the act of choosing which system in a cluster to execute the start or schedule data on. Load balancing is covered in depth in the FastX Clustering Guide. The final result of load balancing is the selection of the server. The Gateway server then sends the object to the session server to start the session.

Is Start Enabled

There is a final check done once the data has come to the session server to check if this server allows sessions to start. At this point, in a properly configured cluster, this check should succeed. However it is still made just in case something went wrong (like load balancing based on user selection and the user sending in his own custom serverId which is disabled)

Link Start

Link is a daemon process that is run as the user. Its main purpose is to start sessions to avoid double logins. FastX first tries to start the session using the currently existing link daemon. More information about Link can be found in the FastX Login Guide. If the link start fails, FastX will move on to alternative options. In all the options, the FastX server attempts to launch a new link daemon which will in turn try to start the session

fastx-protocol

If the user has logged in via the SSH connection from the desktop client, and the user is attempting to launch a session on the same system as the ssh connection, the session will be started from the fastx-protocol process which is the SSH equivalent to link.

Sudo Start

If the initial link start has failed, the web server will then try to sudo to the user starting the session and launch Link again. Sudo has to be enabled in order for this to succeed. As calling sudo can be a security issue, this start method can de disabled

SSH Start

The final attempt to start a session is to use an authenticated start via ssh. The FastX web server will attempt to *ssh user@localhost* on the target Session Server. The client and server will pass messages back and forth as if it was logging in (which technically iit is). Upon success, it will once again launch a Link and then from that Link, start the session. If starting sessions fail at this point, an error occurs.

License Check

The FastX web server has now executed the start command, and a session is starting up. The session will now do a license check to determine that there is a license for the product. If the check succeeds, FastX will startup. If the check fails, FastX will error

Session Start

Congratulations! You have successfully started a FastX session!

StarNet Communications Corp.
4677 Old Ironside Drive, Suite 210, Santa Clara, California 95054-1825 • USA
408.739.0881 • 408.739.0936 • sales@starnet.com • www.starnet.com