Syncplify AFT!

This knowledge base covers topics like installation, configuration, and operation of your Syncplify AFT! scriptable/automated file transfer client.

- Operation

    - Understanding what Syncplify AFT! is

    - Forgot your AFT! admin password? Here's what to do

    - How to start AFT! jobs via REST API but using the command-line

    - Alternative (lower-level) way to start a script via PowerShell and REST API

- Scripting

    - How to monitor a local folder and upload files to your SFTP server when they change

Operation

Topics that cover typical operational tasks in Syncplify AFT!

Understanding what Syncplify AFT! is

True **Managed File Transfer (MFT)** requires the interoperation of 2 parts:

- a file transfer server (that's what [Syncplify Server!](#) is)
- an automated file transfer client (that's what [Syncplify AFT!](#) is)

While Syncplify Server! has been well established on the market as one of the overall best SFTP/FTPS servers for several years, we just recently released its MFT counterpart: **Syncplify AFT!**

The video here below shows the main features found in **Syncplify AFT!** and is a good general overview of the product itself.

https://www.youtube.com/embed/0qdROnpPezE

Forgot your AFT! admin password? Here's what to do

AFT! supports multiple administrative profiles, so the best thing to do when an administrator forgot their password is simply to log in with a different admin profile, and reset the password of the admin who has forgotten it.

But what if you only had 1 admin profile, and you forgot its password? How to regain access to AFT! in that case? No worries, here's a little trick that can get you back in your AFT! instance without losing any of your scripts and/or configuration.

First of all, **locate your AFT!'s configuration data directory**.
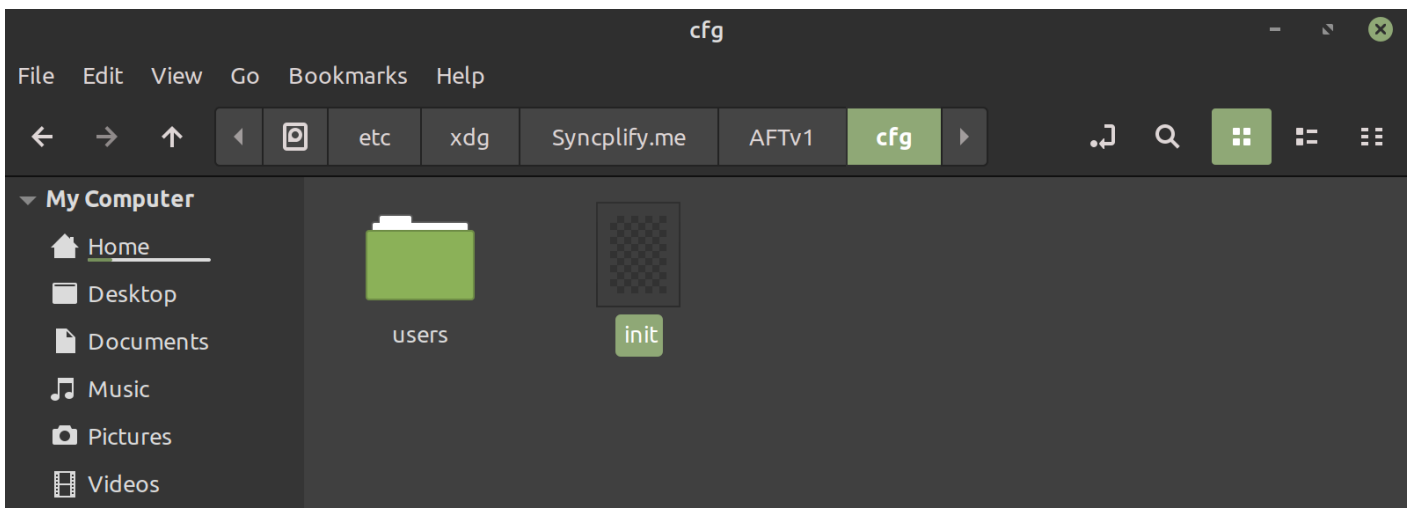
In that directory there is a file named **init** (no extension). Now simply **delete that file named init**.

Once you have deleted the **init** file, launch your browser again, and try to connect to your AFT!'s web UI. You'll notice that you will be asked to create a brand new administrative profile when you do so.
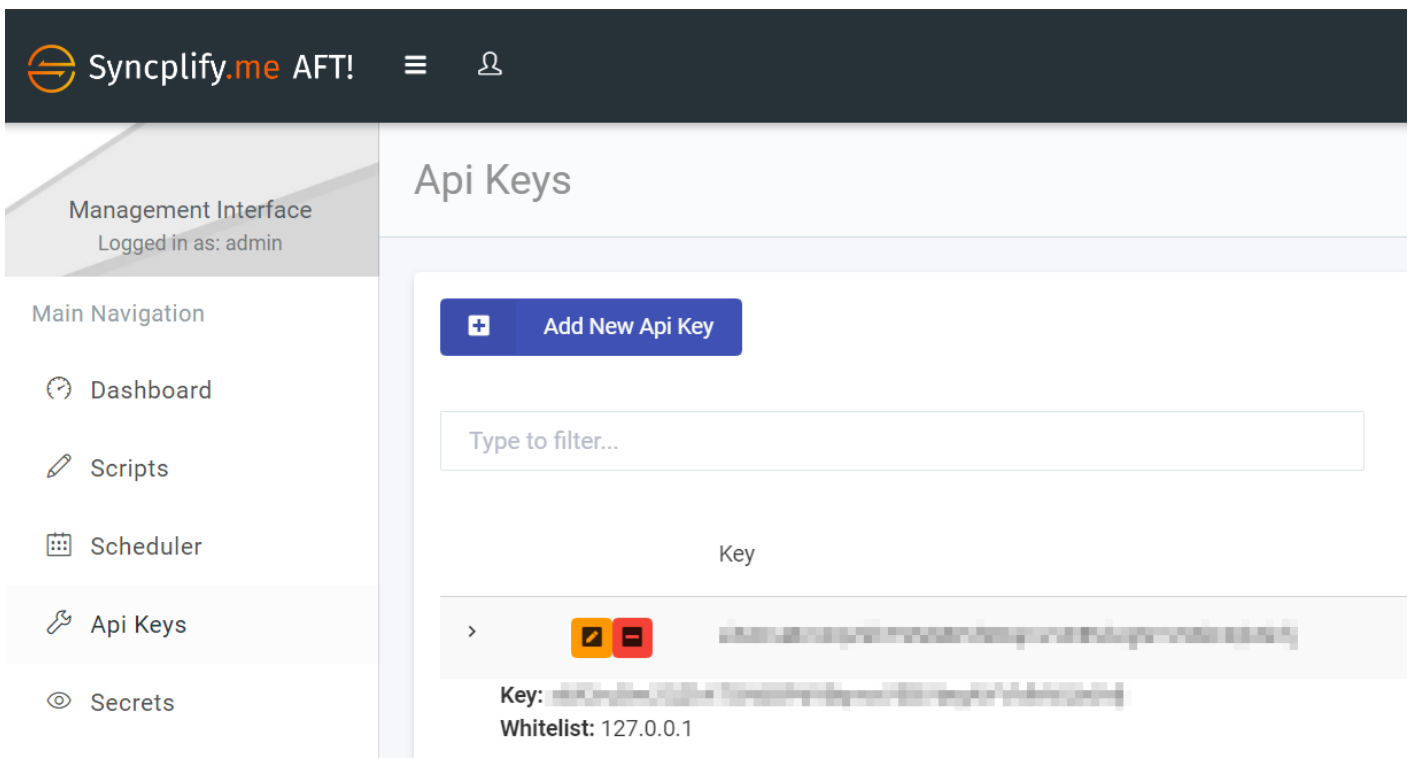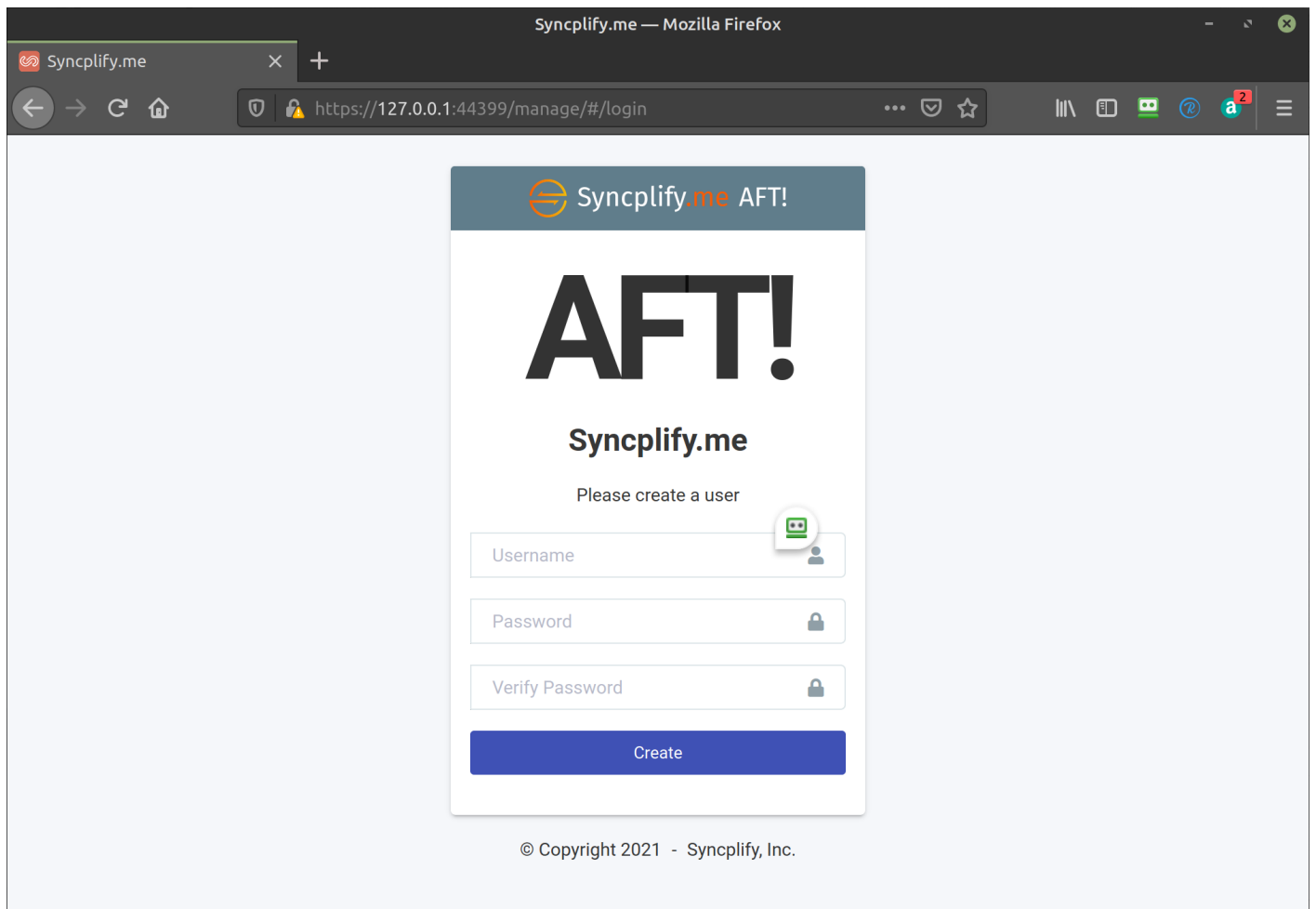
That's it. Create a new admin profile, and you'll be right back into your AFT! web UI.

How to start AFT! jobs via REST API but using the command-line

As of version 3.0, Syncplify.me AFT! has added the capability to start jobs via the **START** command-line switch and parameters. To be clear, unlike the **RUN** command (which executes a script from file in the context of whatever shell you invoked it) the **START** command will actually call a **REST API** for you to start the job within the context of the Syncplify.me AFT! system service.

First of all, you need to make sure you meet the prerequisites, which is basically only one: you need to create an API Key from inside AFT!'s web interface, and make sure such API Key can be used from the IP address you'll be calling it from. Here's an example API Key that will work when invoked from 127.0.0.1 (localhost):

Once you have the API Key, you can use it to invoke the START command via command-line.

Here's the simplest example, using only the API Key and the ID of the script you want to run:

```
.\aft.exe start -s "p3fEoexohdHj6SXrvuT8H8" -a "xiM2ruBm2QZkhTSN6BPd9BqmxVEBVbrgNYVMkNQb6hfj"
```

Once you have the API Key, you can use it to invoke the START command via command-line.

Here's a complete example, which also explicitly adds the optional **HOST** and **PARAMETERS** command-line payload:

```
.\aft.exe start -s "p3fEoexohdHj6SXrvuT8H8" -a "xiM2ruBm2QZkhTSN6BPd9BqmxVEBVbrgNYVMkNQb6hfj" --
params "[{\`"name\`":\`"character\`",\`"value\`":\`"goofy\`"}]" --host "127.0.0.1:44399"
```

Both of these examples assume you're invoking them from PowerShell... please keep in mind that other shells (like cmd.exe or Unix/Linux shells) may have different string "escaping" mechanisms and notations.

Alternative (lower-level) way to start a script via PowerShell and REST API

With Syncplify AFT! you have quite a few different ways to run and/or start the execution of your secure file transfer jobs. One such way is via REST API. But how to invoke such REST API via PowerShell in an easy way? Here's an example script for you:

```
$Header = @{
    "X-API-Key" = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
}

$Body = @{
    jobType = "SCRIPT"
    scriptId = "xxxxxxxxxxxxxxxxxxxxxxx"
}

$Parameters = @{
    Method      = "POST"
    Uri         = "https://127.0.0.1:44399/v1/jobs"
    Headers     = $Header
    ContentType = "application/json"
    Body        = ($Body | ConvertTo-Json)
}

Invoke-RestMethod @Parameters -SkipCertificateCheck
```

All you need to do is substitute the censored (xxxxxx...) API Key and Script ID with your own values, and that's it, it'll work just fine.

Scripting

Scripting examples and techniques in SyncJS, the JavaScript-based scripting language behind Syncplify AFT!

How to monitor a local folder and upload files to your SFTP server when they change

Let's say you have an SFTP server somewhere, and you want to use it as some form of "real-time backup". That implies monitoring a local folder/directory on your computer's hard disk (or SSD), and:

- detect when new files are created and upload them
- detect when files are modified and upload them

Furthermore, there are several other aspects to consider. For example:

- your SFTP server may (should) not allow your client to be always connected
- you may need to "delay" your uploads, because the OS (file-system) needs time to finish writing the local file before you can access it and upload it to the remote SFTP server

AFT! can help you with all of that.

Here below you can see a well-commented AFT! script (in pure SyncJS language) that shows you how to do all of the above the right way, taking all the above-mentioned circumstances into account:

```
{
  // Let's enable console feedback, in case we're running this script via the shell
  ConsoleFeedback = true;

  // First, let's create the file-system watcher
  watcher = new FsWatcher();
  // Then we elect to delay notification by *at least* 300 seconds (5 minutes)
  // (useful to allow the file system to finish whatever operation is ongoing)
  watcher.DelayBySeconds = 300;
  // We may choose *not* to be notified of certain events
  watcher.NotifyRename = false;
```

```
watcher.NotifyRemove = false;
watcher.NotifyChmod = false;
// We can specify inclusion and exclusion filters (optional, not mandatory)
watcher.InclusionFilter = ['*.*'];
watcher.ExclusionFilter = ['notes.txt', 'budget.xlsx'];
// Almost ready, let's tell the object what folder we want to monitor
watcher.WatchDir('C:\\TestFolder', false);
// And then start the watcher
watcher.Start();

// We need to keep checking events indefinitely, an endless cycle is what we need
while (true) {
  // Let's sleep for 500 milliseconds at each cycle, to keep CPU usage low
  Sleep(500);
  // When inside an endless cycle, it's always safe to check if we received a Halt signal at every cycle
  if (HaltSignalReceived()) {
    break;
  }
  // No Halt signal? Good, then let's acquire the list of pending event that we need to process
  evt = watcher.Events()
  // Do we have at least 1 event to process?
  if (evt.length > 0) {
    // We only connect to the server IF there are events to be processes
    var scli = new SftpClient();
    scli.Host = 'your.sftpserver.com:22';
    scli.User = 'your_username';
    scli.Pass = 'your_password';
    scli.Options.UploadPolicy = AlwaysOverwrite;
    if (scli.Connect()) {
      // Cycle over all pending events...
      for (var i = 0; i < evt.length; i++) {
        if (evt[i].Event == 'WRITE') {
          // If it is a WRITE event (new or modified file) let's upload it to the server
          scli.UploadWithPath(evt[i].Object, '/destinationpath', 1);
        }
      }
      // Do not forget to close the connection
      scli.Close();
    }
    // Set the client object to null to save memory
```

```
        scli = null;
      }
    }
  }
```

Happy coding!