Sending a Slack summary after a batch file transfer

Per-file notifications are noisy. A job that uploads fifty files and sends fifty Slack messages trains people to ignore all of them, which defeats the purpose. A better pattern is to collect the results of an entire batch, build a single formatted summary, send it once, and exit. This article shows how to write a script that downloads all new files from a remote directory, counts successes and failures, tracks total bytes transferred, and delivers one Slack message at the end of the run.

How It Works

The script retrieves the list of files waiting in a remote directory, downloads each one to a local inbox folder, tracks counters through the loop, and then sends a formatted Slack notification summarizing the outcome. A guard at the top skips the Slack notification entirely if the remote directory is empty so your Slack channel does not fill up with "nothing to do" messages overnight.

```
// Batch transfer with Slack summary notification
// Downloads all files from a remote directory into a local inbox, tracks
// success/failure and total byte count, and sends a single Slack message
// summarising the batch.

var remoteVfsName = Params("src_vfs")  || "partner-sftp";
var remoteDir     = Params("src_dir")  || "/outbound/";
var localInbox    = Params("dst_dir")  || "/data/inbox/";
var webhookUrl    = GetSecret("slack-webhook-url");

var vfs   = VirtualFSByName(remoteVfsName);
var rdRes = vfs.ReadDir(remoteDir, SortByName, SortAsc);

if (!rdRes.Ok()) {
   Log.Error("cannot list remote directory: " + rdRes.ErrorMsg());
   Exit(1);
}

var files = rdRes.Infos().filter(function(f) { return f.Type === "FILE"; });
```

```
if (files.length === 0) {
    Log.Info("no files to process, exiting quietly");
    Exit(0);
}

var filesOk     = 0;
var filesFailed = 0;
var totalBytes  = 0;

for (var i = 0; i < files.length; i++) {
    var f       = files[i];
    var srcPath = remoteDir + f.Name;
    var dstPath = localInbox + f.Name;

    var res = vfs.ExportFile(srcPath, dstPath);
    if (res.Ok()) {
        filesOk    += 1;
        totalBytes += f.Size;
        Log.Info("downloaded: " + f.Name);
    } else {
        filesFailed += 1;
        Log.Error("failed: " + f.Name + " (" + res.ErrorMsg() + ")");
    }
}

// Format total bytes into a human-readable label.
function bytesLabel(n) {
    if (n >= 1073741824) return (n / 1073741824).toFixed(2) + " GB";
    if (n >= 1048576)    return (n / 1048576).toFixed(2)    + " MB";
    if (n >= 1024)       return (n / 1024).toFixed(2)       + " KB";
    return n + " B";
}

var stamp   = FormatDateTime("YYYY-MM-DD HH:mm");
var icon    = filesFailed > 0 ? ":warning:" : ":white_check_mark:";
var summary = "[" + stamp + "] Batch from " + remoteVfsName + " complete. " +
              filesOk + " succeeded, " + filesFailed + " failed. " +
              "Total: " + bytesLabel(totalBytes) + ".";

SendToSlackWebHook(webhookUrl, summary, "AFT Transfer Bot", icon);
```

```
if (filesFailed > 0) {
    Exit(1);
}
```

`files.filter(...)`: `ReadDir` returns both files and subdirectories. Filtering on `f.Type === "FILE"` ensures the loop only processes actual files, not directory entries, which cannot be downloaded as files.

`Exit(0)` for an empty batch: calling `Exit(0)` sends a clean successful status to the job engine without any Slack notification. This is the right approach for scheduled jobs that run more frequently than files arrive; the AFT! job log records the run, but your team's Slack channel stays quiet.

`SendToSlackWebHook(webhookUrl, summary, "AFT Transfer Bot", icon)`: the third argument is the sender display name that appears in Slack, and the fourth is a Slack emoji shortcode for the icon. Choosing a different emoji when there are failures (`:warning:`) versus all successes (`:white_check_mark:`) lets people scan the channel at a glance without reading every message.

`Exit(1)` after the Slack message when failures occurred: the exit code feeds back to the AFT! job engine and is recorded in the job status. If this job is triggered by a cron schedule, a non-zero exit marks the run as crashed, which lets you configure alerting at the job engine level as a secondary safety net.

Setting up the Slack webhook

In your Slack workspace, go to the App Directory, find the Incoming Webhooks app, and create a new webhook pointed at the channel you want to post to. Copy the webhook URL and store it in AFT! as a secret named `slack-webhook-url`. The URL is sensitive, treat it like a credential; anyone who has it can post to your channel.

Richer formatting with Slack Block Kit

`SendToSlackWebHook` sends a plain-text message. For richer formatting with sections, dividers, and fields laid out in columns, build a Block Kit payload manually and use `HttpCli` to post it directly to the webhook URL:

```
var payload = JSON.stringify({
    blocks: [
        { type: "section", text: { type: "mrkdwn", text: "*Batch complete*: " + filesOk + " files, " +
bytesLabel(totalBytes) } }
    ]
});

var hc = new HttpCli()
var resp = hc.Url(webhookUrl)
```

```
        .Header("Content-Type", "application/json")
        .Post(payload);
```

This gives you the full Slack formatting palette without any additional dependencies.

Revision #1
Created 12 April 2026 15:18:34 by DevTeam
Updated 12 April 2026 15:25:43 by DevTeam