## NAME

**gxa−setup** is a *HOWTO* manual to set up GigA+ on a single (non−balanced) server.

## SYNOPSIS

This document provides an example of the steps needed to install and configure **GigA+** on a single server, without data replication or load balancing involved.

## INSTALLATION and REMOVAL

GigA+ is installed on **Linux** from either a *.deb* package or an *.rpm*, depending on the target distribution. On **FreeBSD**, the installation is done from a compressed *tar* archive.

### Installing (removing) from RPM

The initial file for an RPM installation is a *tarball* archive that one uploads to the destination server. Once copied, the contents of the archive must be extracted: there'll be an RPM and a shell script inside. The script is invoked (with admin privileges) given the name of the *rpm* as a parameter.

Here's an exapmle of how it's done:

```
tar -xvf gigaplus-0.1-2.2.x86_64.tar
gigaplus-0.1-2.2.x86_64.rpm
install-gigaplus-rpm.sh

sudo ./install-gigaplus-rpm.sh gigaplus-0.1-2.2.x86_64.rpm
```

The installation script will install the necessary repositories and packages, invoking **yum**(8) in the process. To uninstall, one must invoke **yum**(8) manually, as in

```
sudo yum remove gigaplus
```

### Installing (removing) from DEB

For a Debian−compliant install the GigA+ *tarball* archive contains a *.deb* package and **install-deb.sh** script. To install, one simply runs the script with the path to the *.deb* as a parameter. The underlying call to **apt−get**(8) takes care of the dependencies. Uinistall would be performed with **apt-get** as well. Here are the relevant examples:

```
sudo ./install-deb.sh gigaplus.deb
~/tmp$ sudo ./install-deb.sh gigaplus.deb
Selecting previously unselected package gigaplus.
(Reading database ... 88467 files and directories currently installed.)
Preparing to unpack gigaplus.deb ...
Unpacking gigaplus (1.5-7.4) ...
Setting up gigaplus (1.5-7.4) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Reading package lists... Done
Building dependency tree
Reading state information... Done
0 upgraded, 0 newly installed, 0 to remove and 9 not upgraded.
```

To uninstall:

```
~/tmp$ sudo apt-get remove gigaplus
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  gigaplus
0 upgraded, 0 newly installed, 1 to remove and 9 not upgraded.
After this operation, 6636 MB disk space will be freed.
Removing gigaplus (1.5-7.4) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
```

**NB**: GigA+ dependencies will **not** be uninstalled automatically. Please use **apt−get autoremove** to purge them.

**Installing (removing) on FreeBSD**

A *tarball* archive is provided for installation on FreeBSD. Extract it into a separate directory and follow the instructions in the *INSTALL.txt* file. Here's an example:

```
$ ls
INSTALL.txt Makefile install-bsd-deps.sh opt tarball-setup.sh usr
$ cat INSTALL.txt
# @(#) installation instructions for Gigaplus (tarball)
#
----------------------
To install Gigaplus, just run at the prompt:

   make install - to install into the default location (root-based);
OR
   PREFIX=/your/cursom/root make install - to install to a custom root location.

To uninstall, do the same but with the 'uninstall' make target:

   make uninstall - to uninstall from the default location;
OR
   PREFIX=/your/cursom/root make uninstall

-----------------------
NB: uninstall will attempt to delete all previosly installed files,
   but not the directories or any files that you've created there.

# __EOF__

$
$ sudo make install
Installing GigA+
All dependencies already installed
Done.
$ type gxws
gxws is /usr/local/bin/gxws
```

Uinistalling is as simple:

```
$ sudo make uninstall
Uninstalling GigA+
```

Done.

**IMPORTANT**: the installation script under FreeBSD uses **pkg**(8) system to install dependencies. If ports are to be used instead, please modify the *install-bsd-deps.sh* (part of the archive) accordingly.

**NB**: GigA+ dependencies will **not** be uninstalled automatically.

## LICENSE

A license is required to run GigA+ modules. If we just run a module now, we would see that a valid license is missing.

```
$ gxws -V
gxws (Web service) 0.1-1.5 (fea395-bsd) regular [FreeBSD 11.0 FreeBSD (amd64) - 11.0-RELEASE-p1/amd64]
==
= ERROR: License not found, please contact support@gigapxy.com ==
==
License validation failed
```

### Generate the server key

For a license to be generated, a *server key* is required. Almost every module of GigA+ can generate one using the **−K** option.

```
$ sudo -u gigaplus gxws -K
System key:          [1dbd23e4d70f2ff6662ff30816958087c9800a2d8354a532] (0x188a)
```

**NB**: Undef Linux you can generate key without **sudo**(8) ; however, on FreeBSD you need to used **sudo -u gigaplus** or add your current user to the **kmem** group.

### Get and validate the license

Include the server key(s) in the license−request email to support@gigapxy.com. Get a license *tarball* containing *gxa.lic* file − this is your license. Save the license in a safe place and make sure you have a backup copy.

Validating a license is simple: copy *gxa.lic* to a temporary folder and run *gxws −V*:

```
$ uname -svr
FreeBSD 11.0-RELEASE-p1 FreeBSD 11.0-RELEASE-p1 #0 r306420: Thu Sep 29 01:43:23 UTC 2016     root(
$ gxws -V
gxws (Web service) 0.1-3.2 (gigaplus) regular [FreeBSD 11.0 FreeBSD (amd64) - 11.0-RELEASE-p1/amd64] (2
```

In the info string the license expiration date in the round brackets, past the build platform: (2017-12-31). We're good to go until the year's end. Now that we know this license works, we can copy (or link) *gxa.lic* to */etc* (Linux) or */usr/local/etc* (FreeBSD). With the license copied, it can be seen from anywhere on the host.

```
cp gxa.lic /opt/gxa/etc
sudo ln -s /opt/gxa/etc/gxa.lic /usr/local/etc/gxa.lic
pushd /tmp
$ gxws -V
gxws (Web service) 0.1-3.4 (fea454-setup) regular [FreeBSD 11.0 FreeBSD (amd64) - 11.0-RELEASE-p1/amd6
```

## PRE−CONFIGURATION TASKS

There are a few things that should be done before we get to setting up configuration and running GigA+ modules.

### Read the documentation

Not **all** of it, of course. But don't go any further untill you've read **gigaplus**(1) man page and understood what kind of a setup you need. Do you need HLS? Single− or multi−server setup? Will you use **gxng**(1) as the file server or would you rely on NginX to do the job? It makes sense to get acquainted with the names of **GigA+** modules, for they will keep getting used throughout this manual without any additional hints on their purpose.

### Check your source streams

The streams that would become your *channels* need to be accessible on this server, period. So, please check that you can read them. For multicast streams, you can use tools like **multicat**(1) from VideoLAN or **ncl**(1) supplied with GigA+, or any other tool. If you are going to deliver a stream via *HLS*, make sure that it is **HLS−compliant** in codecs and can be analyzed with **ffprobe**(1) utility.

**NB**: **ffprobe**(1) and **ffmpeg**(1) are *NOT* installed by default, so you might need to install them (as part of **ffmpeg** suite) manually.

### Decide on your data storage

If you intend to use *DVR* in your HLS delivery (non−HLS DVR is not available), you might need quite a bit of storage space for video segments. That storage could be all in one logical volume or spread across different (sharded) disks. You need to know all the partitions to be involved. If needed, replace one of the */opt/gxa* (GigA+ data root) directories with a soft link, as in:

```
$ ls -ld /opt/gxa/channel
lrwxrwxrwx. 1 gigaplus gigaplus 16 ĐÑĐ» 20 19:27 /opt/gxa/channel -> /mnt/hd1/channel
```

### Adjust ownership on GigA+ data directories

Make sure that ownership/permissions on */opt/gxa* (data root for GigA+) and its subdirectories match your expectations. Initially they are owned by gigaplus:wheel.

```
$ ls -ld /opt/gxa
drwxrwxr-x  8 gigaplus  wheel  512 Aug  2 11:53 /opt/gxa/
```

### Enable core dumps

Core dumps are essential (to analyze crashes of binary components). Please ensure core dumps are enabled, get generated and go to the directory of your choice in the format of your choosing. Having process id in the core−dump name is a lot of help to whoever works on it later. The following script enables core dumps on Linux:

```
cat corep.sh
# @(#) Makes sure cores are dumped in a proper format.

core_dir=/opt/gxa/core
[ -d "${core_dir:?}" ] || mkdir -m 777 ${core_dir}

echo '1' > /proc/sys/fs/suid_dumpable
echo '/opt/gxa/core/%e.%p.core' > /proc/sys/kernel/core_pattern
```

```
# __EOF__
```

**Ensure emails can be sent**

Email notifications are sent by at least one GigA+ script, so SMTP should be working on your server. **vsm** uses **mailx**(1) to send email notifications.

## SETTING UP CONFIGURATION

There are two default config files supplied with GigA+ package: *gigaplus-default.env* and *gigaplus-default.conf*. Both should be made copies of, edited and moved to */etc* (*/usr/local/etc*) as *gigaplus.env* and *gigaplus.conf*. The default config files are initially at */opt/gxa/etc*.

```
cp /opt/gxa/etc/gigaplus-default.env gigaplus.env
cp /opt/gxa/etc/gigaplus-default.conf gigaplus.conf
```

The *.env* config is a small one, but *gigaplus.conf* contains sections for all the GigA+ modules, so we'll be editing it in phases, step by step. In order to understand a section of *gigaplus.conf*, one must read the relevant page(s) of the documentation.

**gigaplus.env**

This configuration is a shell script used by GigA+ to set global environment variables.

```
$ cat gigaplus.env
# @(#) GigA+ settings for shell scripts
#
# This file is included by GigA+ shell scripts
# to initialize the runtime environment.

#   NB: It is YOUR responsibility to set the correct values and
#   uncomment the necessary parameters.

# Root directory for GigA+ data (MANDATORY).
GXA_ROOT=/opt/gxa

# Root directory for channel data (MANDATORY).
GXA_CHANNEL_ROOT=/opt/gxa/channel

# Directory for channel specs.
GXA_SPEC_DIR=/opt/gxa/etc/spec

# Directory for log archives.
# Channel-specific logs would reside within channel-named subdirectories,
# others under "gxa-common" subdirectory.
#
# Re-define when logs are to be stored on a different volume/partition.
#
GXA_LOG_ARCHIVE=/opt/gxa/log/archive

# __EOF__
```

Unless you plan to use a data root different from */opt/gxa*, not much

**gigaplus.conf: GXWS and GXNG**

For this step you must first study **gxws**(1) and **gxng**(1) pages. Without reading them in full, not much could be made sense of here.

In the **ws** the first thing you might want to look at are *network interface* names for the user/admin listeners (set by default to *all* so that you can try the module without any modifications).

**HLS** is enabled by default (**ws.hls.enabled = true**), set it to *false* if you don't need HLS.

Turn off built−in TPUT stats if you won't be using reports: stats consume CPU and (under much stress) have been known to cause some instability. **ws.tput_stats.enabled = false**.

In this scenario, we'll be using HLS **and** linear streaming but (in in this example) will **not** use the TPUT stats (**ws.tput_stats.enabled = false**). The changes are as below:

```
$ diff gigaplus.conf /opt/gxa/etc/gigaplus-default.conf
32,33c32,33
<       admin = { ifc = "lo0"; port = "4047"; default_af = "inet";  };
<       user  = { ifc = "em0"; port = "4046"; default_af = "inet"; };
---
>       admin = { ifc = "all"; port = "4047"; default_af = "inet";  };
>       user  = { ifc = "all"; port = "4046"; default_af = "inet"; };
68c68
<       enabled           = false;
---
>       enabled           = true;

$ sudo gxws -C ./gigaplus.conf
2017-08-02 13:07:07.135342 MSK (INF)  GWS  Config file=[./gigaplus.conf]
$
$ sudo ps aux | grep gxws
gigaplus 3773   0.0  0.8 14140  4152  -  S    13:07     0:00.00 gxws -C ./gigaplus.conf (gxa)

$ tail -f /opt/gxa/log/gxws.log
==========================================
2017-08-02 13:07:07.139158 MSK 3773  (NRM)  GWS  gxws (Web service) 0.1-3.4 (fea454-setup) regular [Fre
2017-08-02 13:07:07.139171 MSK 3773  (INF)  GWS  Config read from [./gigaplus.conf]
2017-08-02 13:07:07.139199 MSK 3773  (INF)  GWS  Runtime pid=3773 uid/gid = 1002/1002 running in [/usr/
2017-08-02 13:07:07.139362 MSK 3773  (INF)  GWS  STARTED listener fd4 for ADMIN requests on lo0:4047
2017-08-02 13:07:07.139447 MSK 3773  (INF)  GWS  STARTED listener fd5 for MODULE requests on /opt/gx
2017-08-02 13:07:07.139469 MSK 3773  (CRI)  GWS  GWS [pid=3773] is running.
2017-08-02 13:07:07.139730 MSK 3773  (INF)  GWS  Entering event loop
```

It's fair to note that **gxws** is not (yet) listening for user requests because it's waiting for the first **gxng** to attach. In the scenario where we don't need a **gxng** to process requests (only **gxws** and **NginX**), we could mark a listener as *no_gng = true*.

```
{ alias="int1"; ifc = "all"; port = "5146"; default_af = "inet"; is_safe = true; no_gng = true; }
```

That would cause **gxws** to start listening on *int1* right away.

The **ng.** section of the config is rather complex by itself (please do not neglect to read **gxng**(1) manpage thoroughly), but for the immediate purpose (initial setup, not tuning), we change only one thing:

**ws.tput_stats.enabled = false**.

**NB**: if we were to go for an **HLS−only** setup, we could completely disable buffers (**ng.bufd.enabled = false**) and save on memory consumption. Since we'd be using linear streams too, we need the buffers, so bufd stays enabled.

We start **gxng** manually to make sure the configuration works:

```
$ sudo gxng -C ./gigaplus.conf
$ ps aux | egrep "gxng|gxws"
gigaplus 6099  0.0  0.8   14140  4152  -  I    16:24     0:00.00 gxws -C ./gigaplus.conf (gxa)
gigaplus 6111  0.0  0.9 1128440  4380  -  I    16:25     0:00.00 gxng -C ./gigaplus.conf (gxa)

$ tail -f /opt/gxa/log/gxng.log
2017-08-02 16:25:25.033550 MSK 6111  (INF)  GNG  new_bufd_rec: 0x8014422c0 path=[]: BR15 - fd-1/m(0x8
2017-08-02 16:25:25.033562 MSK 6111  (INF)  GNG  new_bufd_rec: 0x801442400 path=[]: BR16 - fd-1/m(0x8
2017-08-02 16:25:25.033850 MSK 6111  (INF)  GNG  open_domain_client: fd=2 connected to [/opt/gxa/run/con
2017-08-02 16:25:25.034177 MSK 6111  (CRI)  GNG  CONNECTED to WS [/opt/gxa/run/comm.socket], fd=2
2017-08-02 16:25:25.034426 MSK 6111  (CRI)  GNG  GNG [pid=6111] is running.
2017-08-02 16:25:25.034575 MSK 6111  (INF)  GNG  Entering event loop

$ sudo kill 6099 6111
```

Now that we know that the initial configuration works, we copy/link it as permanent and start **gxws** and **gxng** from the script:

```
$ cp gigaplus.env gigaplus.conf /opt/gxa/etc/
$ sudo ln -s /opt/gxa/etc/gigaplus.conf /usr/local/etc/gigaplus.conf
$ sudo ln -s /opt/gxa/etc/gigaplus.env /usr/local/etc/gigaplus.env

$ ln -s /usr/local/share/gigaplus/scripts/gxa-requests.sh
$ ./gxa-requests.sh
Usage: ./gxa-requests.sh [--nopm|--killpm] start|stop|status [num]
Where:
   gxpm will NOT launch if --nopm specified at start;
   will NOT stop gxpm unless --killpm specified at stop;
   num = number of (gng) engines to launch [1..64]

Set GXA_DEBUG=1 to see what commands are run.

$
$ sudo ./gxa-requests.sh --nopm --nopin start 2
[GXWS] is not running.
Starting GXWS ...
Starting GXNG1 ...
Starting GXNG2 ...
Pausing for 1 second(s).
GXWS [6651] STARTED
GXNG [6664] STARTED
GXNG [6678] STARTED
[GXPM] is not running.
$
$ sudo ps aux | egrep 'gxws|gxng' | grep -v egrep
gigaplus 6651  0.0  0.8   14140  4152  -  S    17:19     0:00.00 /usr/local/bin/gxws -q -l /opt/gxa/log/gxws.log -p /c
gigaplus 6664  0.0  0.9 1128440  4380  -  S    17:19     0:00.00 /usr/local/bin/gxng -q -l /opt/gxa/log/gxng1.log -p
```

```
gigaplus 6678  0.0  0.9 1128440  4380  -  S    17:19     0:00.00 /usr/local/bin/gxng -q -l /opt/gxa/log/gxng2.log -p
$
$ sudo ./gxa-requests.sh stop
GXWS [6651] is running
GXWS [6651] stopped.
$
```

We've used **−−nopm not** to start **gxpm**, because we have not configured it yet, and **−−nopin** not to assign CPU affinity to **gxng**s, becasue the test box has just one core.

This configuration is enough for one to start testing linear streaming. Start **gxws** and **gxng**s as above and use **vlc**, **wget** or any other tool with an available multicast group (or a unicast HTTP URL) to see the data flowing. See **gigaplus**(1) for examples of linear−streaming requests. Example of a linear−stream request:

```
$ wget -O /dev/null 'http://192.168.1.112:4046/udp/239.1.2.30:3333'
```

**gigaplus.conf: GXPM using GXNG**

This part requires one to read **gxpm**(1) page first. Here we edit the **gpm** section of *gigapxy.conf* and test−launch **gxpm**.

**gxng**(1) will be the HTTP server delivering HLS data segments, so we change the **gpm.item_url_prefix**. The prefix will now direct traffic back to **gxws**(1) user listener and the *hls−fra* URL segment specify to gxws that this query is for a data segment via **gxng**.

```
$ diff /opt/gxa/etc/gigaplus.conf /opt/gxa/etc/gigaplus-default.conf
206c211
<     item_url_prefix = "http://192.168.1.112:4046/hls-fra/";
---
>     item_url_prefix = "http://acme.tv:4046/hls-fra/";

$ sudo ./gxa-requests.sh --nopin start 2
[GXWS] is not running.
[GXPM] is not running.
Starting GXPM ...
Starting GXWS ...
Starting GXNG1 ...
Starting GXNG2 ...
Pausing for 1 second(s).
GXWS [6768] STARTED
GXNG [6781] STARTED
GXNG [6794] STARTED
GXPM [6760] STARTED
GXNG [6794] is running
```

## SETTING UP HLS CHANNELS

This chapter requires that you study and understand the **vsm**(1) , **vsm-scripts**(1) and **gxseg**(1) pages, on top of all the prior reading requirements. You may also consider reading **wux**(1) page (not a must).

Launching a simple multicast−based channel (that requires no trans−coding) would be the first step. The chapter concludes with setting up and testing three channels, this would cover both DVR and immediate trans−coding.

**A simple channel**

An example of a **vsm**(1) channel spec is provided in */usr/local/share/doc/gigaplus/examples/* (FreeBSD, skip 'local' under Linux). We copy and edit it to suit out needs. Our channel is going to be multi-cast−based, without much data saved, since it would be serving only *LIVE* HLS requests.

```
$ cp /usr/local/share/doc/gigaplus/examples/vsm-channel.spec spec/cable1.spec
$ vim spec/cable1.spec
$ diff spec/cable1.spec /usr/local/share/doc/gigaplus/examples/vsm-channel.spec
6c6
< cha_ID="cable1"
---
> cha_ID="sample1-hd"
12c12
< cha_URL="udp://239.1.2.30:33333"
---
> cha_URL="http://acme.tv:4046/udp/239.1.2.30:33333"
18c18
< # cha_PFX="http://localhost:8181/seg/"
---
> cha_PFX="http://localhost:8181/seg/"
23c23
< # cha_SHARDS="vol1:vol2:vol3"
---
> cha_SHARDS="vol1:vol2:vol3"
35c35
< # cha_DURATION=6
---
> cha_DURATION=6
41c41
< # cha_ROLLOVER=3600
---
> cha_ROLLOVER=3600
```

No shards are needed (since we don't keep much data). For temporary segments we assume that the default choice (/tmp) is fine. Automatic roll−overs were disabled: we will schedule roll−overs via **cron**(8) using *force−rollover.sh* script. Segment duration is 6 seconds. Channel−specific prefix is not needed since we have the global **gpm.item_url_prefix** set up in *gigaplus.conf*.

If you already know how much of data each channel would be expected to hold, you should set *cha_LMARK_DISKSIZE* and *cha_HMARK_DISKSIZE* parameters. The second of those is the high threshold that your channel should never over−reach in size. The first one is the minimal amount of data that the channel should accumulate before **hos**(1) considers removing stale data segments.

Before we go further, let's check if the channel's codecs are HLS−compliant. Video: H.264 format and audio in AAC, MP3, AC-3 or EC-3. From **ffprobe**(1) we get HLS−compliant A/V specs:

```
Stream #0:0[0x100]: Video: h264 (High) ([27][0][0][0] / 0x001B), yuv420p(progressive), 1280x640 [SAR 1:1
Stream #0:1[0x101](eng): Audio: ac3 ([129][0][0][0] / 0x0081), 48000 Hz, 5.1(side), fltp, 448 kb/s
```

A directory must be created for the channel. There **vsm** will store data segments as well as channel meta−data, such as manifests, pidfiles, channel−specific logs, etc.

```
$ sudo -u gigaplus mkdir -m 775 -p /opt/gxa/channel/cable1
$ ls -ld /opt/gxa/channel/cable1
```

```
drwxrwxr-x  2 gigaplus  wheel  512 Aug  2 22:46 /opt/gxa/channel/cable1/
$
$ vsm
vsm (GigA+ Video stream manager) 0.1.3-8
Usage: /usr/local/bin/vsm [-l logfile] specfile [gxseg-options]
$
```

First we start **vsm**(1) without the log to see if the spec works.

```
$ sudo -u gigaplus vsm spec/cable1.spec
2017-08-04 MSK 11:30:47 vsm 17801 START vsm 0.1.3-15-.: /usr/local/bin/vsm specs/cable1.spec
2017-08-04 MSK 11:30:47 vsm 17801 Re-loading spec=specs/cable1.spec
2017-08-04 MSK 11:30:47 vsm 17801 Set /opt/gxa/channel/cable1/vsm.pid with vsm(17801)
2017-08-04 MSK 11:30:57 vsm 17801 Launched cable1/udp://239.1.2.30:3333
2017-08-04 MSK 11:30:57 vsm 17801 Waiting for gxseg(17868)
1501835465: >> SEGMENT: cable1/20170804-11/64728487.ts 1501835457 8.300000 3957964 0 64728487
1501835470: >> SEGMENT: cable1/20170804-11/65475487.ts 1501835465 5.214000 2478780 0 65475487
1501835477: >> SEGMENT: cable1/20170804-11/65944747.ts 1501835470 6.798000 3314252 0 65944747
1501835483: >> SEGMENT: cable1/20170804-11/66556567.ts 1501835477 5.505000 2890876 0 66556567
```

The bottom lines (after *Waiting for gxseg(17868)*) are the STDOUT from the channel−bound **gxseg**. They lines contain meta−data sent to **gxpm**(1) via **wux**(1) utility. For the immediate purpose they indicate that the stream is being segmented − the spec is operational.

Next, we want to be able to manage (start|stop|status) this channel via the control script: **gxa-channels.sh**. For that, we first need to copy out spec to */opt/gxa/etc/spec*, where it would be seen by the script. Then we can check for the channel's status/uptime, start and stop it.

```
$ cp specs/cable1.spec /opt/gxa/etc/spec/
$ ln -s /usr/local/share/gigaplus/scripts/gxa-channels.sh
$ ./gxa-channels.sh
Usage: ./gxa-channels.sh [--nopm] [--pause N] start|stop|status|shutdown [{channel}|all]

Options:
   --nopm = do NOT attempt to start/stop gxpm.
   --pause = wait N [5] seconds before checking on channels after start.
Note:
   shutdown mode needs no parameters, it will stop all channels and gxpm.

GigA+ channel launch script (feel free to modify).
This script is part of GigA+. Copyright 2017 by Pavel Cherenkov
$
$ ./gxa-channels.sh --nopm status
gxpm [17158] is running

Channels:
cable1: (ON) uptime: 00d 00h:06m:23s
$
$ sudo ./gxa-channels.sh --nopm stop
gxpm [17158] is running

Channels:
```

```
cable1: (ON) uptime: 00d 00h:06m:53s

Stopping all
cable1 stopped (pid=17958)
One second...
gxpm [17158] is running

Channels:
cable1: (OFF) - stale pidfile (17958) removed
$
```

**Channel with incompliant audio**

Moving on to a more complex case, another source. We will use **ffprobe**(1) (again) to check whether the source stream is HLS−compliant.

```
Stream #0:0[0xcd]: Video: h264 (High) ([27][0][0][0] / 0x001B), yuv420p(tv, bt470bg, top first), 720x576 [SA
Stream #0:1[0x131](rus): Audio: mp2 ([4][0][0][0] / 0x0004), 48000 Hz, stereo, s16p, 192 kb/s
```

As we see, audio is **mp2** − incompliant. NOTE: it does not mean that **all** players would not play the stream back, but some definiely will reject it. Before you decide to transcode, check if you may **NOT** require trans−coding for your particular players. If you do opt to transcode, read on.

Unlike video, audio is (relatively) easy to trans−code on the fly, and that's what we'll do by adding a transcoding task to the spec. First we test though:

```
$ ffmpeg -re -i http://acme.tv:4404/champ2 -map 0:0 -map 0:1 -vcodec copy -c:a aac -map_metadata 0:p:0 -f mp
$ ffprobe transcoded.ts
  Stream #0:0[0x100]: Video: h264 (High) ([27][0][0][0] / 0x001B), yuv420p(tv, bt470bg, top first), 720x576 [S
  Stream #0:1[0x101](rus): Audio: aac (LC) ([15][0][0][0] / 0x000F), 48000 Hz, stereo, fltp, 133 kb/s
```

The **ffmpeg** options we supplied keeps video AS−IS and trans−codes audio to *AAC*. Our transcoding task will look as below:

```
# Transcode to AAC audio.
cha_TRANSOPT="−map 0:0 -map 0:1 −vcodec copy −c:a aac"
```

As we test−run the channel, we verify that our audio is now AAC:

```
ffmpeg -loglevel quiet -i http://acme.tv:4404/champ2 -map 0:0 -map 0:1 -vcodec copy -c:a aac -map_metadata 0:

$ ffprobe /opt/gxa/channel/champ2/20170804-14/14910240.ts
  Stream #0:0[0x100]: Video: h264 (High) ([27][0][0][0] / 0x001B), yuv420p(tv, bt470bg, top first), 720x576 [S
  Stream #0:1[0x101](rus): Audio: aac (LC) ([15][0][0][0] / 0x000F), 48000 Hz, stereo, fltp, 126 kb/s
```

**NOTE**: Sometimes a channel would start manually but not the first time from the *gxa-channels.sh* script. The first clue would be the *vsm.log* found at the channel root (in this case − */opt/gxa/channel/champ2/vsm.log*). From there, it may take one to a **gxseg** log, in the same directory. With plenty of warnings there like below:

```
2017-08-04 14:21:41.051865 MSK 18826 app_run: WARNING: stream[1] no PTS/DTS provided, PTS/DTS=-92
```

it would make sense just to re−try the launch:

```
$ sudo ./gxa-channels.sh start champ2
gxpm [17158] is running
appending output to nohup.out
champ2 [18904] is running
Pausing for [5] seconds..
gxpm [17158] is running

Channels:
champ2: (ON) uptime: 00d 00h:00m:06s
$
$ ./gxa-channels.sh status
gxpm [17158] is running

Channels:
cable1: (OFF)
champ2: (ON) uptime: 00d 00h:09m:42s
$
```

Meanwhile, **gxpm** rotates the stale segments (beyond our 600−second window), as we see from the log:

```
2017-08-04 14:36:12.328090 MSK 17158  (INF)  GPM  apply_src_changes: retiring [champ2/20170804-14/732
2017-08-04 14:36:12.328097 MSK 17158  (INF)  GPM  recycle_segfile: removing [/opt/gxa/channel/champ2/20
```

**DVR channel with disk shards**

The next steps would be to set up a long-term DVR channel − **news3**. The channel will hold a lot of of data, this means that we'd need more space for its segments than for the two previous channels. We will spread the segments of the new channels across three large disks mounted to */opt/disk1*, */opt/disk2* and */opt/disk3*. The structure of */opt/gxa/channel* directory will looks as below:

```
$ ls -l
total 12
drwxrwxr-x  3 gigaplus  wheel  512 Aug  4 17:23 cable1
drwxr-xr-x  3 gigaplus  wheel  512 Aug  4 17:04 champ2
lrwxr-xr-x  1 gigaplus  wheel   10 Aug  4 17:45 disk1 -> /opt/disk1
lrwxr-xr-x  1 gigaplus  wheel   10 Aug  4 17:45 disk2 -> /opt/disk2
lrwxr-xr-x  1 gigaplus  wheel   10 Aug  4 17:45 disk3 -> /opt/disk3
drwxr-xr-x  2 gigaplus  wheel  512 Aug  4 17:46 news3
$ ls -l disk1/
total 4
drwxr-xr-x  2 gigaplus  wheel  512 Aug  4 17:46 news3
$
```

The top−level *news3* subdirectory will hold meta−data, the *news3* subdirectories under disk1..disk3 will hold data segments. The channel will hold 24 hours, i.e. 86400 seconds of data. The relevant changes in the spec would be:

```
cha_SHARDS="disk1:disk2:disk3"
cha_CAPACITY=86400
```

As we test−run *news3*, we can see how data segments are spread:

```
1501858803: >> SEGMENT: disk1/news3/20170804-18/7820397290.ts 1501858802 5.000000 1664364 0 78203
2017-08-04 MSK 18:00:03 vsm 2256 base gxseg(2304) is running, playlist=/opt/gxa/channel/news3/gxseg0-new
2017-08-04 MSK 18:00:03 vsm 2256 Launched news3/http://enc14.4net.tv:4046/udp/239.222.55.102:5000
2017-08-04 MSK 18:00:03 vsm 2256 Waiting for gxseg(2304)
```

```
1501858808: >> SEGMENT: disk2/news3/20170804-18/7820847290.ts 1501858803 5.000000 1706852 0 78208
1501858813: >> SEGMENT: disk3/news3/20170804-18/7821297290.ts 1501858808 5.000000 1658160 0 78212
1501858818: >> SEGMENT: disk1/news3/20170804-18/7821747290.ts 1501858813 5.000000 1675644 0 78217
1501858823: >> SEGMENT: disk2/news3/20170804-18/7822197290.ts 1501858818 5.000000 1700648 0 78221
1501858828: >> SEGMENT: disk3/news3/20170804-18/7822647290.ts 1501858823 5.000000 1604768 0 78226
1501858833: >> SEGMENT: disk1/news3/20170804-18/7823097290.ts 1501858828 5.000000 1758176 0 78230
----
$ ls /opt/disk1/news3/20170804-18/
7820397290.ts   7821747290.ts   7823097290.ts   7824447290.ts   7825797290.ts
$
```

We start *news3* in the regular fashion now:

```
$ sudo ./gxa-channels.sh start news3
gxpm [1902] is running
appending output to nohup.out
news3 [2410] is running
Pausing for [5] seconds..
gxpm [1902] is running

Channels:
news3: (ON) uptime: 00d 00h:00m:06s
```

**Setting up for 24/7 service**

Reading material for this section is **vsm-scripts**(5) page from GigA+ and the **crontab**(8) page from the system documentation. For reference on spec parameters, please consider re−reading **vsm**(1) documentation.

The first thing we do is: we add (uncomment) **cha_ADMIN_EMAIL** parameter in all our specs and set it to a valid email of a dedicated tech−support person.

The next thing to consider is **scheduling rollovers**. From prior experience, a rollover should happen at least once per day. Things*can* (potentially) go wrong during a rollo ver, so it should not be done at the busy hours. Neither should it be at an hour when a failure would be unnoticed − it's truly your call when it should happen. For this example I picked the time close to **5 am** for rollovers on the three channels we've set up.

Another consideration is **NOT** to roll over **all channels at once**. Should rollovers fail on more than one channel, nobody wants to handle it all at once. But scheduling is truly your call.

**Log rotation** for **vsm** and **gxseg** is next. **hos** script will do that (see **vsm-scripts**(5) for proper options). For each channel we create (set up) a log−archive directory under */opt/gxa/log*: archive/channel1, archive/champ2, etc. Of course, those directories could be just links to another partition/storage, if needed (would make sense to keep archives on a slower medium than segment files and meta−data).

**hos** also does segment rotation. **gxpm** handles rotation only when a channel is *ONLINE*, so **hos** is needed for the (inevitable) cases when a channel goes *OFFLINE* and **gxsm** no longer remembers the segments left behind. **hos** is a safety belt, in a way. We want to schedule at least one run of it that would rotate *vsm.log*. We also schedule a status report that would alert us if any channels are offline. A channel that**is supposed** to be offline should have a *SUSPEND* file in its root directory, as in */opt/gxa/channel/champ2/SUSPEND*.

The crontab now looks like below:

```
$ sudo -u gigaplus crontab -l
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin:/usr/local/sbin
SHELL=/usr/local/bin/bash
#
# CABLE1
#
# Hourly hos(1).
34 * * * * /usr/local/bin/hos --sysrep cable1
# Daily hos(1) to rotate vsm log.
20 2 * * * /usr/local/bin/hos --sysrep -L cable1
# Roll-over
30 0 * * * /opt/gxa/tmp/force-rollover.sh /opt/gxa/channel/cable1
#
# CHAMP2
#
# Hourly hos(1).
12 * * * * /usr/local/bin/hos --sysrep champ2
# Daily hos(1) to rotate vsm log.
45 2 * * * /usr/local/bin/hos --sysrep -L champ2
# Roll-over
40 0 * * * /opt/gxa/tmp/force-rollover.sh /opt/gxa/channel/champ2
#
# NEWS3
#
# Hourly hos(1).
05 * * * * /usr/local/bin/hos --sysrep news3
# Daily hos(1) to rotate vsm log.
57 2 * * * /usr/local/bin/hos --sysrep -L news3
# Roll-over
20 0 * * * /opt/gxa/tmp/force-rollover.sh /opt/gxa/channel/news3
##
# COMMON
##
# Hourly alert:
36 * * * * /opt/gxa/tmp/gxa-channels.sh --alert status
40 * * * * /usr/local/bin/hos --common --sysrep any
#
# END OF TABLE
```

**NB**: having *PATH* and *SHELL* assignments is **crucial** for cron jobs' work under **FreeBSD**. **hos** relies on finding **bash**(1) via **env**(1) which is impossible from **cron**(8) environment without the pre−set *PATH*. Do note that under Linux **bash** is NOT under **/usr/local**.

Once you've set up the **crontable**(8) , do make sure you've seen your jobs run before you leave the system be. Go to a mail client (like **alpine**(1) ) and check your email from cron jobs. Make sure to invoke the mail client under **gigaplus** user as in:

```
$ sudo -u gigaplus alpine
```

We start all channels with the **gxa-channels.sh** script now.

```
[bsl45@wi-fbsd11 ~/tmp]$ sudo ./gxa-channels.sh start
gxpm [1902] is running
appending output to nohup.out
```

```
cable1 [3554] is running
appending output to nohup.out
champ2 [3645] is running
appending output to nohup.out
news3 [3732] is running
Pausing for [5] seconds..
gxpm [1902] is running

Channels:
cable1: (ON) uptime: 00d 00h:00m:07s
champ2: (ON) uptime: 00d 00h:00m:06s
news3: (ON) uptime: 00d 00h:00m:06s
$
```

**Playback: testing the whole chain**

We start **gxws**, **gxng** using the control script:

```
$ sudo ./gxa-requests.sh --nopin start
Password:
[GXWS] is not running.
GXPM [18474] is running
Starting GXWS ...
Starting GXNG1 ...
Pausing for 1 second(s).
GXWS [19155] STARTED
GXNG [19167] STARTED
GXPM [18474] STARTED
$
$ sudo ./gxa-requests.sh status
GXPM [18474] is running
GXWS [19155] is running
GXNG [19167] is running
$
```

Requesting non−HLS channel from **gxws**:

```
$ wget -O /dev/null 'http://192.168.1.112:4046/udp/239.1.2.30:3333'
--2017-08-09 18:40:01--  http://192.168.1.112:4046/udp/239.1.2.30:3333
Connecting to 192.168.1.112:4046... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/octet-stream]
Saving to: â/dev/nullâ

  [                                <=>       ] 9Â 839Â 276    652KB/s            ^
```

Using the same URL with **vlc** or another player should result in the playback of the unicast−relayed channel.

Playing back an HLS channel:

```
$ cvlc -vvv http://192.168.1.112:4046/hls-m3u/champ2/playlist.m3u8

[00007f25a0c03df8] httplive stream debug: updating hls stream (program-id=0, bandwidth=0) has 6 segments
[00007f25a0c03df8] httplive stream debug: - segment 85 appended
```

```
[00007f25a0c03df8] httplive stream debug:   - segments new max duration 4
[00007f25a0c03df8] httplive stream debug: - segment 86 appended
[00007f25a0c03df8] httplive stream debug:   - segments new max duration 4
[00007f25a0c03df8] core stream debug: creating access 'http' location='192.168.1.112:4046/hls-fra/champ2/201
[00007f259c000958] core access debug: looking for access module matching "http": 25 candidates
[00007f259c000958] http access debug: querying proxy for http://192.168.1.112:4046/hls-fra/champ2/20170810
[00007f259c000958] http access debug: no proxy
[00007f259c000958] http access debug: http: server='192.168.1.112' port=4046 file='/hls-fra/champ2/20170810
[00007f259c000958] core access debug: net: connecting to 192.168.1.112 port 4046
[00007f259c000958] core access debug: connection succeeded (socket = 12)
[00007f259c000958] http access debug: protocol 'HTTP' answer code 200
[00007f259c000958] http access debug: Server: gxws/0.1-3.24 (fea454-setup)
[00007f259c000958] http access debug: Content-Type: video/MP2T
```

This concludes the single−server setup for GigA+. Steps invloving multi−server load−balanced setup procedure are given in the **gxa-lb-setup**(5) page of the documentation.

## AUTHORS

Pavel V. Cherenkov

## SEE ALSO

**gigaplus**(1),**gxng**(1),**gxpm**(1),**vsm**(1),**vsm−scripts**(1),**gxseg**(1),**ncl**(1),**multicat**(1),**ffprobe**(1),**mailx**(1)

## NAME

**gxa−lb−setup** is a *HOWTO* manual to set up GigA+ on one central−point (CP) server with **N >= 1** additional load−balancing nodes (LBNs).

## SYNOPSIS

This document provides an example of the steps needed to set up **GigA+** on a central−point (CP1) server, with data replication covering multiple (N >= 1) load−balancing nodes. The manual will be specific to **HLS−only** setup, without linear−stream delivery, and with **nginx**(1) third−party HTTP server used both for segment delivery (instead of **gxng**(1) ) and as the load−balancing gateway (or *router*).

## INSTALLATION and REMOVAL

These steps should be re−traced from **gxa−setup**(5) page, they won't differ since the package is the same. All functionality already covered by the non−balanced setup manual will be skipped.

Our **CP1** server is going to be running **Ubuntu 14.04 LTS** (only for the sake of example).

## SETUP of request handlers

The first steps would be to set up **gxws**(1) to work without **gxng**(1) modules and then **nginx**(1) as the web-server for the segment files.

### gxws setup

The initial alterations are, as usual, for the concrete network interfaces, so we change **all** as the interface name to *eth0*. Another thing to do is to mark the user interface as one not requiring a **gxng**(1) component. This is done in **gigaplus.conf** as:

```
listener: {
   admin = { ifc = "lo"; port = "4047"; default_af = "inet";  };
   user  = { ifc = "eth0"; port = "4046"; default_af = "inet"; no_gng = true; };
};
```

First we run **gxws** manually in debug mode to see that the config works:

```
~/tmp$ rm -f gxws.log && sudo -u gigaplus gxws -Tvvv -C /opt/gxa/etc/gigaplus.conf -l gxws.log

less gxws.log
2017-08-11 17:04:46.326512 MSK 4962  (INF)  GWS  STARTED listener fd7 for USER requests on eth0:4046;

~ $ wget -O /dev/null 'http://192.168.1.103:4046/ping'
--2017-08-11 16:56:20--  http://192.168.1.103:4046/ping
Connecting to 192.168.1.103:4046... connected.
HTTP request sent, awaiting response... 400 Bad request
2017-08-11 16:56:20 ERROR 400: Bad request.
$
```

The listener is up and responded to a ping with HTTP 400 (Bad request), which indicates that requests are received and processed. We can proceed to the next step.

### nginx setup

We install nginx and set up web−accessible directories to allow access to segment files. Setting up is quite simple:

```
$ sudo apt-get update
$ sudo apt-get install nginx
~/tmp$
```

There is an example of a **NginX** config in */usr/share/doc/gigaplus/examples/nginx/server.conf* (on Linux, please use **/usr/local on BSD). It has a section that opens up access to our** */opt/gxa/channel* **directory via HTTP port** *8181***.**

```
#
# Direct access to data segments.
#
location /channel/ {
    root /opt/gxa/;
    autoindex on;
}
```

We can add this config to NginX and test the access.

```
$ cp /usr/share/doc/gigaplus/examples/nginx/server.conf /opt/gxa/etc/nginx-gxa.conf
$ sudo ln -s /opt/gxa/etc/nginx-gxa.conf /etc/nginx/conf.d/nginx-gxa.conf
$ pgrep nginx
5492
5493
5494
5495
5496
$ sudo nginx -s reload
$ echo 'empty' > /opt/gxa/channel/empty.txt
$ wget -O /dev/null 'http://192.168.1.103:8181/channel/empty.txt'
--2017-08-11 17:40:55--  http://192.168.1.103:8181/channel/empty.txt
Connecting to 192.168.1.103:8181... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6 [text/plain]
Saving to: â/dev/nullâ

100%[=======================================================>] 6           --.-K/s   in

2017-08-11 17:40:55 (635 KB/s) - â/dev/nullâ saved [6/6]

$
```

Also, you can open *http://192.168.1.103:8181/channel/* (mind the IP) in a browser to see *empty.txt* listed. In any case, we can now retrieve files from our CP1 server via NginX.

**gxpm setup**

We can modify **gxpm**(1) config to provide NginX−oriented URLs in the playlists. First we make it point at CP1 cerver, which is NOT the final version of the config since data segments will be distributed across the nodes (LBNs). The nodes we are yet to set up, but we enable notifications via a multicast group. For now, we make the following changes in */opt/gxa/etc/gigaplus.conf*:

```
gpm.item_url_prefix = "http://192.168.1.103:8181/channel/";
gpm.md_report: {
    enabled = true;
```

```
        pub_url = "udp://227.3.2.160:2020";
        lid: { prefix = "wi-ub1404"; min = 100; max = 299; };
        src_base = "http://192.168.1.103:8181/channel";
        mcast_ifc = "eth0";
};
```

Setting *item_url_prefix* will provide URLs for access via NginX. Enabling reports means that for each new segment registered by **gxpm** there'll be a multicast message to the *pub_url* group that **dwg**s should subscribe to. We link our configs to /etc to make them globally visible and test−start **gxws**(1) and **gxpm**(1) via script.

```
$ sudo ln -s /opt/gxa/etc/gigaplus.env /etc/gigaplus.env
$ sudo ln -s /opt/gxa/etc/gigaplus.conf /etc/gigaplus.conf
$ ln -s /usr/share/gigaplus/scripts/gxa-requests.sh
$ ln -s /usr/share/gigaplus/scripts/gxa-channels.sh
$
$ ./gxa-requests.sh status
[GXPM] is not running.
[GXWS] is not running.
$
$ sudo ./gxa-requests.sh --nopin start none
[GXWS] is not running.
[GXPM] is not running.
Starting GXPM ...
Starting GXWS ...
Pausing for 1 second(s).
GXWS [6225] STARTED
GXPM [6220] STARTED
$
$ tail -f /opt/gxa/log/gxws.log
===========================================
2017-08-11 18:14:49.699713 MSK 6225  (INF)  GWS  STARTED listener fd5 for USER requests on eth0:4046;
2017-08-11 18:14:49.699736 MSK 6225  (INF)  GWS  STARTED listener fd6 for ADMIN requests on lo:4047;
2017-08-11 18:14:49.699807 MSK 6225  (INF)  GWS  STARTED listener fd7 for MODULE requests on /opt/gx
2017-08-11 18:14:49.699828 MSK 6225  (CRI)  GWS  GWS [pid=6225] is running.
2017-08-11 18:14:49.699892 MSK 6225  (INF)  GWS  Entering event loop
$
$ tail -f /opt/gxa/log/gpm.log
-----------------------------------
2017-08-11 18:14:44.658837 MSK 6220  (INF)  GPM  STARTED listener fd4 for URQ requests on /opt/gxa/run
2017-08-11 18:14:44.658894 MSK 6220  (INF)  GPM  STARTED listener fd5 for URQ requests on /opt/gxa/run
2017-08-11 18:14:44.658922 MSK 6220  (NRM)  GPM  gxpm (Playlist manager) 0.1-4.4 (fea468-lb) regular [U
2017-08-11 18:14:44.658931 MSK 6220  (INF)  GPM  Config read from [/etc/gigaplus.conf]
2017-08-11 18:14:44.658943 MSK 6220  (INF)  GPM  Entering event loop
$
```

Both components are running, with **gxws** running without **gxng**s. Please kindly note that in order NOT to start **gxng**s we've used the **none** parameter in gxa-requests.sh.

## Setting up channels

We set up the same channels we did in the single−server manual (see **gxa−setup**(5) for details).

**Checking URLs in the playlist**

We will set up one channel and verify that **gxpm** generates playlists with valid URLs, resolvable via **nginx**.

```
$ sudo ./gxa-channels.sh start cable1
[gxpm] is not running.
Starting GXPM ...
gxpm [8210] is running
nohup: redirecting stderr to stdout
cable1 [8218] is running
Pausing for [5] seconds..
gxpm [8210] is running
----- channels -----
(ON) cable1 uptime: 00d 00h:00m:05s
--------------------
$
$ sudo ./gxa-requests.sh --nopin start none
[GXWS] is not running.
GXPM [8210] is running
Starting GXWS ...
Pausing for 1 second(s).
GXWS [8333] STARTED
GXPM [8210] STARTED
$
$ wget 'http://192.168.1.103:4046/hls-m3u/cable1/playlist.m3u8'
--2017-08-11 22:32:04--  http://192.168.1.103:4046/hls-m3u/cable1/playlist.m3u8
Connecting to 192.168.1.103:4046... connected.
HTTP request sent, awaiting response... 200 OK
Length: 601 [application/x-mpegURL]
Saving to: âplaylist.m3u8â

100%[=============================================================>] 601         --.-K/s

2017-08-11 22:32:04 (62,1 MB/s) - âplaylist.m3u8â saved [601/601]

$ cat playlist.m3u8
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-ALLOW-CACHE:YES
#EXT-X-TARGETDURATION:5
#EXT-X-MEDIA-SEQUENCE:21
#EXTINF:4.800000,
http://192.168.1.103:8181/channel/cable1/20170811-22/768766796.ts
#EXTINF:4.800000,
http://192.168.1.103:8181/channel/cable1/20170811-22/769198796.ts
#EXTINF:4.800000,
http://192.168.1.103:8181/channel/cable1/20170811-22/769630796.ts
#EXTINF:4.800000,
http://192.168.1.103:8181/channel/cable1/20170811-22/770062796.ts
#EXTINF:4.800000,
http://192.168.1.103:8181/channel/cable1/20170811-22/770494796.ts
#EXTINF:4.800000,
http://192.168.1.103:8181/channel/cable1/20170811-22/770926796.ts
$
```

Let's see if we can get one of the segments by that link (bypassing **gxws** and going directly to **nginx**):

```
$ wget -O /dev/null http://192.168.1.103:8181/channel/cable1/20170811-22/770926796.ts
--2017-08-11 22:32:40--  http://192.168.1.103:8181/channel/cable1/20170811-22/770926796.ts
Connecting to 192.168.1.103:8181... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6242540 (6,0M) [application/octet-stream]
Saving to: â/dev/nullâ

100%[=======================================================>] 6Â 242Â 540   --.-

2017-08-11 22:32:40 (114 MB/s) - â/dev/nullâ saved [6242540/6242540]
$
```

## SEGMENT REPLICATION

In order to load−balance streams, we are planning to replicate data segments (from selected channels) to multiple servers (*three*, for this exapmle) and have a mediator component bounce requests between them. We start with a simple case of replicating a channel to one additional server (LBN1).

In **GigA+ dwg**(1) is the component (*download agent*) on the receiving end of the replication. Please read **dwg**(1) manpage for details.

### Install and configure dwg on LBN1

In this section we install the necessary component? get a system key + license and set *dwg* up to receive download tasks (as the only agent on that host). Although we only need one module, it's easier to install the whole package. The received license will apply only to **dwg** though. We get the key the usual way:

```
wi-deb8:~/tmp$ dwg -K
System key:          [dcd184816e87f8b0abc90936212c6c91b7dd2c72cbb156fc] (0x204a)
```

We also add the current user to the *adm* group to enable access to newly−created */opt/gxa* and subdirectories. From */etc/gxa-lb.conf* we copy a generic config for load−balancing components and modify it to suit out needs. For now, we disable *dwag.lbdb* by setting **dwag.lbdb.port = 0;**. We also change *local_url* prefix as: **local_url_prefix = "http://192.168.1.105:8181/";** to match the IP address of the host.

We also disable (for now) *dwag.lbdb* on the CP host (wi-ub1404) and adjust its **local_url_prefix**.

Let's run **dwg**(1) with default config and check if it starts:

```
wi-deb8:/opt/gxa/etc$ sudo dwg -0
sudo dwg -0
wi-deb8:~/tmp$ pgrep dwg
16569
wi-deb8:~/tmp$ tail -f /opt/gxa/log/dwg.log
      task time-out       = 3000 ms
      mmap up to          = 3 MB
      local URL prefix    = http://192.168.1.105:8181

-------------------------------------
2017-08-16 16:08:27.080209 MSK 16569  (INF)  DWG  mcast_join: using GENERIC mulitcast API to MCAST
2017-08-16 16:08:27.082231 MSK 16569  (INF)  DWG  udl_ctx_init: fd2 set up to listen on [udp://227.3.2.160:
2017-08-16 16:08:27.082385 MSK 16569  (NRM)  DWG  dwg (Download manager) 0.1-4.13 (fea468-lb2) regu
2017-08-16 16:08:27.082402 MSK 16569  (INF)  DWG  Config read from [./dwg.conf]
2017-08-16 16:08:27.082417 MSK 16569  (INF)  DWG  Entering event loop
```

```
--------------------------------
wi-deb8:/opt/gxa/etc$ sudo ln -s /opt/gxa/etc/dwg.conf /etc/dwg.conf
```

It makes sense to check if multicast data is received well. We do that via **ncl**(1) utility, running it on both ends:

```
wi-ub1404:/opt/gxa/etc$ ncl -wuv -a 227.3.2.160 -p 2020 -d 50
mk_sock: client socket fd=[3]
WRITE [+MEM+] to fd=3, msg_size=1400 delay=50 ms
49000 bytes 2 sec 23.926 Kb/sec, total=47.852 Kb
56000 bytes 2 sec 27.344 Kb/sec, total=102.539 Kb
56000 bytes 2 sec 27.344 Kb/sec, total=157.227 Kb
56000 bytes 2 sec 27.344 Kb/sec, total=211.914 Kb
56000 bytes 2 sec 27.344 Kb/sec, total=266.602 Kb
^CSIG(2) DONE
total sent to fd=3: 287000 bytes

I/O: 280.273438 Kb within 10.236886 sec; 27.378780 Kb/sec
−−−−
```

```
wi-deb8:/opt/gxa/etc$ ncl -ru -a 227.3.2.160 -p 2020
reading from fd=3, msg_size=1400 delay=0 ms
15.723 Kb/sec, total=31.445 Kb
27.344 Kb/sec, total=86.133 Kb
27.344 Kb/sec, total=140.820 Kb
^CSIG(2) DONE
total received from fd=3: 162400 bytes
```

We also need to create directories for the channels we'd be replicating: **cable1**, **champ2** and **news3**:

```
wi-deb8:/opt/gxa/etc$ sudo -u gigaplus mkdir -p -m 775 /opt/gxa/channel/cable1 /opt/gxa/channel/champ2
wi-deb8:/opt/gxa/etc$ sudo -u gigaplus mkdir -p -m 775 /opt/gxa/channel/news3 /opt/gxa/channel/disk1/news3 /
```

We start *cable1* on CP host (wi-ub1404) and check for multicast−group messages on our LBN1 (*wi-deb8*):

```
wi-ub1404:/opt/gxa/tmp$ sudo ./gxa-channels.sh start cable1
```

```
wi-deb8:/opt/gxa/etc$ ncl -ruv -a 227.3.2.160 -p 2020
reading from fd=3, msg_size=1400 delay=0 ms
READ [108]:0.053 Kb/sec, total=0.105 Kb
$
```

Now we start **dwg** on LBN1 and see if it would process notifications correctly and download segments where they belong.

```
wi-deb8:/opt/gxa/etc$ sudo dwg -0
wi-deb8:/opt/gxa/etc$ pgrep dwg
17409
wi-deb8:/opt/gxa/etc$ tail -f dwg.log
```

```
2017-08-16 19:39:49.856307 MSK 17475  (INF)  T0002  read_src_response: ready to download 5857704 bytes t
2017-08-16 19:39:49.872506 MSK 17475  (NRM)  T0002  downloaded: 5857704 bytes, [http://192.168.1.103:81
2017-08-16 19:39:54.606954 MSK 17475  (INF)  T0003  read_src_response: ready to download 6144968 bytes t
2017-08-16 19:39:54.620001 MSK 17475  (NRM)  T0003  downloaded: 6144968 bytes, [http://192.168.1.103:81
```

We'll need **hos** set up on LBNx (node hosts) to clean up stale data segments. For that, we first copy *cable1.spec* from CB. Then we need to mark **cable1** as a *REPLICA* channel (this way **gxa-channels.sh**

won't alert on **vsm** being down).

```
wi-deb8:/opt/gxa/tmp$ ln -s /usr/share/gigaplus/scripts/gxa-channels.sh
wi-deb8:/opt/gxa/tmp$ touch /opt/gxa/channel/cable1/REPLICA
wi-deb8:/opt/gxa/tmp$ ./gxa-channels.sh status
[gxpm] is not running.
----- channels -----
(REPL) cable1 [214 MB]
--------------------
$
wi-deb8:~/tmp$ sudo -u gigaplus crontab -l
##
# CABLE1
##
# Hourly hos(1)
55 * * * * /usr/bin/hos --sysrep cable1
# Common logs
30 02 * * * /usr/bin/hos --common any
# END OF TABLE
```

**NB**: If setting up under FreeBSD, please make sure to include *PATH* and *SHELL* settings as the top lines of your **crontab**(8) and also verify that your jobs work by checking (sudo −u gigaplus) local email from **cron**.

## LOAD BALANCING ON MULTIPLE NODES

In this section we need to set up a small NoSQL (*Redis*) database for load balancing − hereinforth *LBDB*. Then we hook up the existing (*LBN1*) node to it. Then we set up and plug in two additional nodes: *LBN2* and *LBN3*. In the end we'd have CP1 replicating to LBN1−LBN3, with load balancing across CP1 and the nodes.

### Set up LBDB on CP1 and LBN1

Our database could be set up on CP1 or on any LBN host, you should use your own judgement to decide where it should be. (Think of how far and busy your server may get, consider RAM resources for the database − those would depend on how many segments you keep for the replicated channels.)

For reference on how to set up and administer *Redis DB*, please refer to the documentation on their website.

For the database host you do not need a GigA+ license. All you need is **Redis DB** installed and connectivity established from other hosts to LBDB. In this manual we'll set it up on a separate host that is neither a CP or an LBN.

```
[wi-c70 tmp]$ type redis-cli
redis-cli is /usr/bin/redis-cli
[wi-c70 tmp]$
[wi-c70 tmp]$ /sbin/ifconfig
ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.101  netmask 255.255.255.0  broadcast 192.168.1.255
```

Before we go further, we edit redis.conf (here in */etc*) and bind the listener to both the external and the loop interface.

```
[@wi-c70 tmp]$ sudo systemctl status redis
 redis.service - Redis persistent key-value database
   Loaded: loaded (/usr/lib/systemd/system/redis.service; disabled; vendor preset: disabled)
  Drop-In: /etc/systemd/system/redis.service.d
           ââlimit.conf
```

```
   Active: inactive (dead)
[@wi-c70 tmp]$
[@wi-c70 tmp]$ sudo systemctl enable redis
Created symlink from /etc/systemd/system/multi-user.target.wants/redis.service to /usr/lib/systemd/system/redis.s
[@wi-c70 tmp]$ sudo systemctl start redis
[@wi-c70 tmp]$ sudo systemctl status redis
redis.service - Redis persistent key-value database
   Loaded: loaded (/usr/lib/systemd/system/redis.service; enabled; vendor preset: disabled)
  Drop-In: /etc/systemd/system/redis.service.d
           ââlimit.conf
   Active: active (running) since ÐÐ½ 2017-08-21 14:00:24 MSK; 6s ago
 Main PID: 2422 (redis-server)
   CGroup: /system.slice/redis.service
           ââ2422 /usr/bin/redis-server 192.168.1.101:6379

Ð°Ð²Ð³ 21 14:00:24 wi-c70.localdomain systemd[1]: Started Redis persistent key-value database.
Ð°Ð²Ð³ 21 14:00:24 wi-c70.localdomain systemd[1]: Starting Redis persistent key-value database...

[wi-c70 tmp]$ redis-cli ping
PONG
```

On CP1 and LBN1 we install *redis-tools* and check the connectivity.

```
wi-ub1404:/opt/gxa/tmp$ sudo apt-get install redis-tools
Processing triggers for libc-bin (2.19-0ubuntu6.13) ...
wi-ub1404:/opt/gxa/tmp $
wi-ub1404:/opt/gxa/tmp$ redis-cli -h 192.168.1.101 -p 6379 ping
PONG
$
wi-deb8:/opt/gxa/tmp$ redis-cli -h 192.168.1.101 -p 6379 ping
PONG
```

Now we can initialize the LBDB, and we'll do it from CP1. The distribution provides with the means to set up the Redis *Lua scripts* − LBDB's stored procedures. There is an LBDB−specific directory under /opt/gxa, we do it from there:

```
wi-ub1404:/opt/gxa/tmp$ cd /opt/gxa/lbdb/
wi-ub1404:/opt/gxa/lbdb$ sudu -u gigaplus ln -s /usr/share/gigaplus/scripts/lbdb lua
wi-ub1404:/opt/gxa/lbdb$
wi-ub1404:/opt/gxa/lbdb$ ls -l lua/
total 20
-rw-r--r-- 1 root root 1020 Aug 18 20:41 add_url.lua
-rw-r--r-- 1 root root  527 Aug 18 20:41 del_url.lua
-rwxr-xr-x 1 root root 1721 Aug 18 20:41 load-lbdb-lua.sh
-rw-r--r-- 1 root root 3687 Aug 18 20:41 mbget_url.lua
-rw-r--r-- 1 root root 1967 Aug 18 20:41 next_url.lua
$
wi-ub1404:/opt/gxa/lbdb$ lua/load-lbdb-lua.sh
Usage: lua/load-lbdb-lua.sh [-n] lua
You may wish to define:

LBDB_HOST       = DB hostname
LBDB_PORT       = DB port
LBDB_LUA_DIR    = source directory
LBDB_SHA1_DIR   = destination directory

in lbdb.config
```

As the load−script (*load-lbdb-lua.sh*), we should set up *lbdb.config*, which is just a shell script to set up the environment for *load-lbdb-lua.sh*. We also create a separate directories for **SHA1 signatures** of the LBDB stored procedures.

```
wi-ub1404:/opt/gxa/lbdb$ cat lbdb.config
#!/bin/sh
# @(#) LBDB configuration script.

# Host and port for RedisDB
#
LBDB_HOST=192.168.1.101
LBDB_PORT=6379

# Directory with LBDB lua scripts:
LBDB_LUA_DIR=/opt/gxa/lbdb/lua

# Directory for SHA1 signatures.
LBDB_SHA1_DIR=/opt/gxa/lbdb/sha1

# __EOF__
$
wi-ub1404:/opt/gxa/lbdb$ chown gigaplus:adm lbdb.config
wi-ub1404:/opt/gxa/lbdb$ sudo -u gigaplus mkdir sha1
wi-ub1404:/opt/gxa/lbdb$ ls -l
total 8
-rw-rw-r-- 1 gigaplus adm       160 Aug 21 14:50 lbdb.config
lrwxrwxrwx 1 gigaplus gigaplus   32 Aug 21 15:40 lua -> /usr/share/gigaplus/scripts/lbdb
drwxr-xr-x 2 gigaplus gigaplus 4096 Aug 21 15:40 sha1
wi-ub1404:/opt/gxa/lbdb$ ./lua/load-lbdb-lua.sh
Usage: ./lua/load-lbdb-lua.sh [-n] lua
wi-ub1404:/opt/gxa/lbdb$
```

We first run *load-lbdb-lua.sh* in simulation mode, to see if every check it makes passes, and also to see what exactly it does:

```
wi-ub1404:/opt/gxa/lbdb$ sudo -u gigaplus lua/load-lbdb-lua.sh -n
2017-08-21 MSK 16:20:07 WARNING: dry-run mode
redis-cli -h 192.168.1.101 -p 6379 SCRIPT LOAD cat /opt/gxa/lbdb/lua/add_url.lua > /opt/gxa/lbdb/sha1/add_ur
redis-cli -h 192.168.1.101 -p 6379 SCRIPT LOAD cat /opt/gxa/lbdb/lua/del_url.lua > /opt/gxa/lbdb/sha1/del_url
redis-cli -h 192.168.1.101 -p 6379 SCRIPT LOAD cat /opt/gxa/lbdb/lua/next_url.lua > /opt/gxa/lbdb/sha1/next_u
redis-cli -h 192.168.1.101 -p 6379 SCRIPT LOAD cat /opt/gxa/lbdb/lua/mbget_url.lua > /opt/gxa/lbdb/sha1/mbg
2017-08-21 MSK 16:20:07 Done
```

The script quite simply loads stored procedures (a.k.a. Redis Lua scripts) into LDBD and stores the returned SHA1 signatures as *.sha1* files.

```
wi-ub1404:/opt/gxa/lbdb$ sudo -u gigaplus lua/load-lbdb-lua.sh lua
2017-08-21 MSK 16:25:48 /opt/gxa/lbdb/lua/add_url.lua to 192.168.1.101:6379
add_url SHA1=[94357c7cad39fe76cfd90567c88d6f4d483d5dc1]
2017-08-21 MSK 16:25:48 /opt/gxa/lbdb/lua/del_url.lua to 192.168.1.101:6379
del_url SHA1=[b8da283c527d6d8d5a1ffed82f6f135dd186fc8d]
2017-08-21 MSK 16:25:48 /opt/gxa/lbdb/lua/next_url.lua to 192.168.1.101:6379
next_url SHA1=[abac8ac437da8c3b1ab7647954bb51fec5a2a950]
2017-08-21 MSK 16:25:48 /opt/gxa/lbdb/lua/mbget_url.lua to 192.168.1.101:6379
mbget_url SHA1=[05e67bab2fa4c3bc48844d5a511adc6f803c625b]
2017-08-21 MSK 16:25:48 Done
wi-ub1404:/opt/gxa/lbdb$
```

```
wi-ub1404:/opt/gxa/lbdb$ cat sha1/add_url.sha1
94357c7cad39fe76cfd90567c88d6f4d483d5dc1
```

The load−balancing hosts will reference LDBD stored procedures via their SHA1 signatures, so for LBN1 all we need are those signatures that we've just generated on CP1. We just make a directory on LBN1 (wi−deb8) and copy signatures there:

```
wi-deb8:/opt/gxa/tmp$ cd ../lbdb
wi-deb8:/opt/gxa/lbdb$ sudo -u gigaplus mkdir -m 775 sha1
wi-deb8:/opt/gxa/lbdb$ sudo chown gigaplus:adm sha1
wi-deb8:/opt/gxa/lbdb$ ls -l
total 4
drwxrwxr-x 2 gigaplus adm 4096 Aug 21 16:34 sha1
$
wi-ub1404:/opt/gxa/lbdb$ scp sha1/*.sha1 wi-deb8:/opt/gxa/lbdb/sha1/
add_url.sha1            100%   41     0.0KB/s   00:00
del_url.sha1            100%   41     0.0KB/s   00:00
mbget_url.sha1          100%   41     0.0KB/s   00:00
next_url.sha1           100%   41     0.0KB/s   00:00
$
wi-deb8:/opt/gxa/lbdb$ sudo chown gigaplus:adm sha1/*.sha1
$
```

We'll re-configure **dwg**(1) now on both hosts to reference LDBD. Then we'll test−launch **dwg** to see that the settings work.

```
wi-deb8:/opt/gxa/etc$ diff dwg.conf dwg.conf.old
40c40
<     max_mmap_mb = 8;
---
>     max_mmap_mb = 3;
43,44c43,45
<         host = "192.168.1.101";
<         port = 6379;
---
>         host = "127.0.0.1";
>         port = 0;
>         # port = 6379;
47c48
<         sha1_dir = "/opt/gxa/lbdb/sha1";
---
>         sha1_dir = "/opt/gxa/lbdb";
wi-deb8:/opt/gxa/etc$
```

First we increased the maximum size of the file that **dwg** would **mmap**(2) into memory (to save on repeated **write**(2) syscalls) to 8 Mb − judging by the maximum size of segments processed so far. This has nothing to do with load balancing, it's just an optimization that we've applied before **dwg** starts working under load.

Then we changed localhost binding to the address:port of the database we've set up. By setting *port* to non−zero, we made **dwg** LBDB−aware. We also corrected *sha1_dir* to point at the directory for the SHA1 signatures. Now we can test−launch **dwg**:

```
wi-deb8:/opt/gxa/log$ tail -f dwg.log
Load-balancer DB: 192.168.1.101:6379
      Connect timeout       = 0.500000 sec
      SHA1 directory        = /opt/gxa/lbdb/sha1
-----------------------------------
2017-08-21 16:55:33.137712 MSK 1303  (INF)  DWG  mcast_join: using GENERIC mulitcast API to MCAST_
```

```
2017-08-21 16:55:33.139722 MSK 1303  (INF)  DWG  udl_ctx_init: fd2 set up to listen on [udp://227.3.2.160:20
2017-08-21 16:55:33.140318 MSK 1303  (INF)  DWG  lbdb_connect: connected to 192.168.1.101:6379
2017-08-21 16:55:33.140914 MSK 1303  (NRM)  DWG  dwg (Download manager) 0.1-4.22 (fea468-lb3) regula
2017-08-21 16:55:33.140935 MSK 1303  (INF)  DWG  Config read from [/etc/dwg.conf]
2017-08-21 16:55:33.140945 MSK 1303  (INF)  DWG  Entering event loop
^C
```

We can see now that **dwg** has successfully connected to LBDB from LBN1. The procedure (update config and test−launch) is then repeated on CP1 (and later, on all other participating LBNs).

Please note, that **dwg** needs to be running **ONLY** on the hosts participating in load balancing and **serving data segments**. If you opted for a configuration where CP1 is just generating data (to replicate to LBNs), then you don't need **dwg** running on CP1.

### Set up flb − load balancing FastCGI module on CP1

Now we are to set up the component that would do the actual balancing, i.e. decide which of the nodes (including the participating CP1) to direct data−segment requests to. The name of the component is **flb**(1) and it is run not as a stand−alone executable, but as a FastCGI module of a web server, **NginX** in our case. Support for other web servers, such as Apache, has not been tested yet. **flb**(1) will use LBDB to mak e routing decisions, so we must edit its configuration (in *gigaplus.conf*).

```
wi-ub1404:/opt/gxa/etc$ diff gigaplus.conf g1.conf
294,295d293
<     listener = ":9191";
<
302c302,303
<         host = "192.168.1.101";
---
>         # Database connection:
>         host = "127.0.0.1";
306c307
<         sha1_dir = "/opt/gxa/lbdb/sha1";
---
>         sha1_dir = "/opt/gxa/lbdb";
310c311
<     error_url = "http://192.168.1.50";
---
>     error_url = "http://some-resource.url";
320c313
<     lbr_prefix = "gxa-lb";
---
>     # lbr_prefix = "giga-lbr";
```

Out listener path stays default: local TCP port 9191 (go for a UNIX socket if you will). We've changed database−connection settings, signature directory path and the URL to re-direct to if the requested segment is found unavailable. In that case, the full URL would also include parameters identifying the resource. For instance, for

```
GET /channel/news2/20170823-16/3685452938.ts
```

The result would be:

```
HTTP 302 Moved temporarily
Location: http://192.168.1.50?origin=channel%2Fnews2%2F20170823-16%2F3685452938.ts
```

We've added **lbr_prefix = "gxa-lb"** to the load−balancer configuration. This prefix should be in URLs for load−balanced requests, the URLs come from the playlist served by **gxpm**. We must now adjust the *gpm.item_url_prefix* in the config:

```
<  gpm.item_url_prefix = "http://192.168.1.103:8181/channel/";
```

```
--
>  gpm.item_url_prefix = "http://192.168.1.103:8181/gxa-lb/channel/";
```

Now **gxpm** should generate playlist items with the prefix, as below:

```
http://192.168.1.103:8181/gxa-lb/channel/news2/20170823-17/3998832938.ts
http://192.168.1.103:8181/gxa-lb/channel/news2/20170823-17/3999282938.ts
http://192.168.1.103:8181/gxa-lb/channel/news2/20170823-17/3999732938.ts
```

We should also update NginX config to allow re-direction to our FastCGI module. We add the following section to *nginx-gxa.conf* (in */opt/gxa/etc*):

```
location ~ ^/gxa-lb/channel/.*/.*.ts$ {
    include /etc/nginx/fastcgi_params;
    fastcgi_pass  127.0.0.1:9191;
}
```

Now we can test−launch **flb**(1) on port 9191, to make sure the config is valid.

```
wi-ub1404:/opt/gxa/tmp$ sudo -u gigaplus flb -0
---------------------------------
2017-08-23 13:58:19.808986 MSK 10448  (NRM)  FLB  flb (FastCGI load balancer) 0.1-4.27 (fea468-lb3) regul
2017-08-23 13:58:19.808997 MSK 10448  (INF)  FLB  Config read from [/etc/gigaplus.conf]
2017-08-23 13:58:19.809433 MSK 10448  (INF)  FLB  lbdb_connect: connected to 192.168.1.101:6379
2017-08-23 13:58:19.810209 MSK 10448  (NRM)  FLB  Listening on :9191
^C2017-08-23 13:58:33.073055 MSK 10448  (NRM)  FLB  Received QUIT-SIG(2), flb will exit.
```

**Load−balance a channel between CP1 and LB1**

We will now complete the setup for one channel − *news2* (same as *news3* but w/o shards). Segments will be replicated from CP1 to LB1. Segment requests for this channel will be balanced by **flb** running on CP1. When replicating, leaf nodes most be started first, so we set up on LBN1 (wi-deb8). Let's not forget that *NginX* will be serving data segments, so we'd need it installed (and configured) on each LBN. First we install and configure NignX:

```
wi-deb8:/opt/gxa/etc$ cat nginx-gxa.conf
# @(#) nginx server config
server {
    listen 8181;
    location /channel/ {
        root /opt/gxa/;
        autoindex on;
    }
}
$
wi-deb8:/opt/gxa/channel/news2$ sudo apt-get install nginx
Setting up nginx-common (1.6.2-5+deb8u5) ...
Setting up nginx-full (1.6.2-5+deb8u5) ...
Setting up nginx (1.6.2-5+deb8u5) ...
Processing triggers for libc-bin (2.19-18+deb8u10) ...
Processing triggers for systemd (215-17+deb8u7) ...
$
wi-deb8:/opt/gxa/etc$ sudo ln -s /opt/gxa/etc/nginx-gxa.conf /etc/nginx/conf.d/nginx-gxa.conf
wi-deb8:/opt/gxa/etc$ sudo nginx -s reload
bsl45@wi-deb8:/opt/gxa/etc$ pgrep nginx | wc -l
5
$
wi-ub1404:/opt/gxa$ wget -O /dev/null 'http://192.168.1.105:8181/channel/news2/empty1'
--2017-08-23 16:06:51--  http://192.168.1.105:8181/channel/news2/empty1
```

```
Connecting to 192.168.1.105:8181... connected.
HTTP request sent, awaiting response... 200 OK
Length: 0 [application/octet-stream]
Saving to: '/dev/null'

    [ <=>      ] 0         --.-K/s   in 0s

2017-08-23 16:06:51 (0.00 B/s) - '/dev/null' saved [0/0]
$
wi-deb8:/opt/gxa/etc$ sudo -u gigaplus rm /opt/gxa/channel/news2/empty1
$
```

As you see, we've installed/configured NginX and verified that it works with our config. Now we can launch **dwg** on LBN1 and let it wait for the download tasks. Please note that **dwg** is **the only** module we need on leaf hosts.

```
wi-deb8:/opt/gxa/channel$ sudo -u gigaplus mkdir -m 775 news2
wi-deb8:/opt/gxa/channel$ touch news2/REPLICA
$
bsl45@wi-deb8:/opt/gxa/tmp$ sudo ./gxa-channels.sh --dwg 1 --nopm start -
Starting dwg#1
dwg [13969] is running
$
```

Next, on CP1 (wi-ub1404) we start *news2* channel and the request modules (**gxws** and **gxng**):

```
wi-ub1404:/opt/gxa/tmp$ sudo ./gxa-channels.sh --lb --dwg 1 start news2
[sudo] password for bsl45:
Starting dwg#1
dwg [23115] is running
Starting flb ...
flb [23123] is running
[gxpm] is not running.
Starting GXPM ...
gxpm [23131] is running
nohup: redirecting stderr to stdout
news2 [23138] is running
1 channel(s) started
Pausing for [5] seconds..
gxpm [23131] is running
----- channels -----
(ON) news2 [37 MB] uptime: 00d 00h:00m:05s
--------------------
wi-ub1404:/opt/gxa/tmp$
wi-ub1404:/opt/gxa/tmp$ sudo ./gxa-requests.sh --nopin start 2
[GXWS] is not running.
GXPM [23131] is running
Starting GXWS ...
Starting GXNG1 ...
Starting GXNG2 ...
Pausing for 1 second(s).
GXWS [23317] STARTED
GXNG [23328] STARTED
GXNG [23340] STARTED
GXPM [23131] STARTED
$
```

First we look at LBN1 to see if **dwg** is picking up download tasks:

```
2017-08-24 16:48:09.784244 MSK 13969  (NRM)  uDl  TASK-START [T0046] (0x7f96875b4010/idx=0) [http:/
2017-08-24 16:48:09.785609 MSK 13969  (INF)  T0046  read_src_response: ready to download 1652144 bytes t
2017-08-24 16:48:09.788903 MSK 13969  (INF)  T0046  add_lbdb_url: id=[channel/news2/20170824-16/28345
2017-08-24 16:48:09.789237 MSK 13969  (NRM)  T0046  downloaded: 1652144 bytes, [http://192.168.1.103:81
2017-08-24 16:48:09.789324 MSK 13969  (NRM)  DWG  TASK-END [T0046] (0x7f96875b4010/idx=0) [http:/
2017-08-24 16:48:14.923215 MSK 13969  (NRM)  uDl  TASK-START [T0047] (0x7f96875b4010/idx=0) [http:/
2017-08-24 16:48:14.926020 MSK 13969  (INF)  T0047  read_src_response: ready to download 1862704 bytes t
2017-08-24 16:48:14.932606 MSK 13969  (INF)  T0047  add_lbdb_url: id=[channel/news2/20170824-16/28349
2017-08-24 16:48:14.933132 MSK 13969  (NRM)  T0047  downloaded: 1862704 bytes, [http://192.168.1.103:81
2017-08-24 16:48:14.933300 MSK 13969  (NRM)  DWG  TASK-END [T0047] (0x7f96875b4010/idx=0) [http:/
```

We can also observe that CP1 **dwg** is also picking up tasks, but now downloading since files are local, just updating LBDB:

```
2017-08-24 16:44:58.220605 MSK 23115  (INF)  uDl  add_lbdb_url: id=[channel/news2/20170824-16/2816978
2017-08-24 16:45:03.390771 MSK 23115  (INF)  uDl  add_lbdb_url: id=[channel/news2/20170824-16/2817428
2017-08-24 16:45:08.095992 MSK 23115  (INF)  uDl  add_lbdb_url: id=[channel/news2/20170824-16/2817878
2017-08-24 16:45:13.219761 MSK 23115  (INF)  uDl  add_lbdb_url: id=[channel/news2/20170824-16/2818328
2017-08-24 16:45:18.242066 MSK 23115  (INF)  uDl  add_lbdb_url: id=[channel/news2/20170824-16/2818778
```

We start a video player on *news2* and watch how requests get routed by **flb** in the log:

```
2017-08-24 16:44:20.247549 MSK 23123  (NRM)  FLB  Listening on :9191
2017-08-24 16:45:44.436927 MSK 23123  (INF)  FLB  run_app: [channel/news2/20170824-16/2820128346.ts] r
2017-08-24 16:45:44.654632 MSK 23123  (INF)  FLB  run_app: [channel/news2/20170824-16/2820578346.ts] r
2017-08-24 16:45:44.864475 MSK 23123  (INF)  FLB  run_app: [channel/news2/20170824-16/2821028346.ts] r
2017-08-24 16:45:49.886095 MSK 23123  (INF)  FLB  run_app: [channel/news2/20170824-16/2821478346.ts] r
2017-08-24 16:45:54.882847 MSK 23123  (INF)  FLB  run_app: [channel/news2/20170824-16/2821928346.ts] r
2017-08-24 16:45:59.881699 MSK 23123  (INF)  FLB  run_app: [channel/news2/20170824-16/2822378346.ts] r
2017-08-24 16:46:04.872757 MSK 23123  (INF)  FLB  run_app: [channel/news2/20170824-16/2822828346.ts] r
2017-08-24 16:46:09.874338 MSK 23123  (INF)  FLB  run_app: [channel/news2/20170824-16/2823278346.ts] r
```

As you can see, **flb** alternates between the two participating nodes: CP1 and LBN1.

This is the simplest kind of load balancing where leafs hosts are iterated in a round−robin fashion. More complex ways, invloving numeric metrics will be covered later.

We've used one **dwg** per host for simplicity. A more complex set−up, with N>1 **dwg** instances balancing download tasks on a multi−core server is possible, we will cover it later on.

**Set up LBN2**

We set up replication on one more host, which will become our LBN2. As before, licenses for **dwg** will be needed, otherwise, the procedure is the same as with LB1. We will start LBN2 after CP1 and LB1 and see how **flb** picks up its presence from LBDB. Then we'll shut down news2 on LBN2 and see that **flb** would no longer route requests to it.

Set up on LBN2 (wi-ub1604) is similar to what we did for LBN1, so there's no need to cover it here. We launch LBN1 first (so that it can catch ALL download tasks), then start CP1. If we started CP1, LBN1 would probably miss a few segments. Then we launch a video player for*ne ws2* and watch **flb** routing entries in the log:

```
2017-08-24 19:08:28.814307 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3590618346.ts] r
2017-08-24 19:08:29.018511 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3591068346.ts] r
2017-08-24 19:08:29.243510 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3591518346.ts] r
2017-08-24 19:08:34.239602 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3591968346.ts] r
2017-08-24 19:08:34.444548 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3592418346.ts] r
2017-08-24 19:08:44.248617 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3592868346.ts] r
2017-08-24 19:08:44.350908 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3593318346.ts] r
```

```
2
```

As expected, we alternate between CP1 and LBN1. In a bit we start LBN2 (wi-ub1604 = 192.168.1.173) and see **flb** entries reflect the appearance of the new host:

```
2017-08-24 19:08:54.231491 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3593768346.ts] r
2017-08-24 19:08:59.239232 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3594218346.ts] r
2017-08-24 19:09:04.236200 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3594668346.ts] r
2017-08-24 19:09:04.412276 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3595118346.ts] r
2017-08-24 19:09:14.250755 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3595568346.ts] r
2017-08-24 19:09:19.234769 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3596018346.ts] r
2017-08-24 19:09:24.238210 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3596468346.ts] r
2017-08-24 19:09:29.233826 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3596918346.ts] r
2017-08-24 19:09:34.237161 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3597368346.ts] r
2017-08-24 19:09:34.426468 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3597818346.ts] r
2017-08-24 19:09:44.234789 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3598268346.ts] r
```

Then we shut down relication on LBN2:

```
wi-ub1604:/opt/gxa/tmp$ sudo ./gxa-channels.sh --dwg 1 --nopm shutdown -
Stopping [all]
<skip> news2
0 channels stopped.
Stopped dwg[5205]
[flb] is not running.
$
```

We see that **flb** dropped it too, no 192.168.1.173 from this point on:

```
2017-08-24 19:09:49.235374 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3598718346.ts] r
2017-08-24 19:09:54.240530 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3599168346.ts] r
2017-08-24 19:09:59.259531 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3599618346.ts] r
2017-08-24 19:10:04.250137 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3600068346.ts] r
2017-08-24 19:10:04.454696 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3600518346.ts] r
2017-08-24 19:10:14.240247 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3600968346.ts] r
2017-08-24 19:10:14.442211 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3601418346.ts] r
2017-08-24 19:10:24.250055 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3601868346.ts] r
2017-08-24 19:10:29.249214 MSK 23716  (INF)  FLB  run_app: [channel/news2/20170824-19/3602318346.ts] r
```

This concludes our goal of setting the simplest kind of load balancing on 2+ hosts. Read further for advanced topics.

## ADVANCED LOAD BALANCING

This chapter will cover using numeric *sensors* for load balancing (as vs. doing it round−robin style). We will also set up multiple **dwg** instances on the same box and have them handle the same channels, balancing download tasks between them.

### Using numeric sensors

TODO TODO

### Balancing download tasks between multiple dwg instances.

TODO TODO

## AUTHORS

Pavel V. Cherenkov

## SEE ALSO

**gxa−setup**(5),**gigaplus**(1),**gxpm**(1),**vsm**(1),**vsm−scripts**(1),**gxseg**(1),**ncl**(1)