DynMap

DynMap is my submission to the Winter 2013 "Nexus/mobile app" contest (presentation here). It's basically a live, global "heat map" of sorts that can display any location-based event data in real-time. It is architected to scale horizontally, though at this point I've not put it through enough paces to be sure it will!

Major components

1. **Flume Node.** Log or other ingress data come from one of several "sources" and is loaded into Flume (NG) as "events". (Alternatively, log data can be replayed manually using a Python script I wrote; this is the method used to demo the app for the contest.)
2. Flume "event" data can be pre-processed by Flume sink nodes if necessary (not implemented in the initial, proof-of-concept version), for instance to convert source log data. For scalability, this step should probably be the one to convert IP data to lat/lng coordinates.
3. **Map Event Listener.** Event data is then sent to one or more Node.js listeners using a custom Flume TCP "sink". This is where data can be apportioned for scale, either by category or sharded within a category (maybe acceptable for an app like this).
4. **Map Event Server.** A pub/sub server using Node.js and the socket.io library, serving lat/lng data out to subscribed clients for map rendering. The TCP sink listener and map server listeners are currently part of the same file but can certainly be broken out and load-balanced to allow for scalability (i.e. the apportioning or sharding mentioned above).
5. **Web Client.** Serves the map client itself, which utilizes Google Maps JavaScript API v3.

See the "Technical Stuff" section below for more info on how these are set up.

More Immediate To-Do Items

- Come up with better visualizations for open and click icons (fading? more of a heat map look?)
- Improve map client to handle lost connections (gracefully re-establish them)
- Update map rendering so it sizes and zooms itself more appropriately to the viewing window
- Resolve Flume exec source batchSize issue (seems to be a bug in Flume)
- Break out the listener and server into separate Node.js daemons so they can be run on separate servers
- Move to socket.io "room" functionality (i.e. one "room" for each event type), to ensure the server publishes only the events each client wants
- Add disconnect/reconnect functionality; could not get this working in shorter order for some reason
- In custom Flume TCP sink, improve TCP connection handling: open one connection at start and re-use, and gracefully handle and re-establish lost connections
- Look into costs for Google Maps licensing (each map key comes with 25,000 free map points a day)

Ideas for the Future

There are many uses for this sort of visualization. By its nature, the effect says "wow" and both technical and especially non-technical people should find it compelling. Thanks to so many that have shared their ideas! Add yours below if you like.

- Platform-wide activity
  - DynECT Email: showing all or a repesentative subset of opens, clicks, and even sends
  - DynECT DNS: queries, zone prop?, other?
  - Dyn: dyn.com hits, dyndns queries?, #dyn and other related tweets
- Customer-specific
  - live or replay of an email campaign, for instance (i.e. by xheader) -- show sends, opens, clicks
  - DNS queries for a given timeframe of the customer's choosing, such as following an email campaign, following a news release, a new website launch, you name it
- replay of any historical series of events

The Technical Stuff

DynMap was engineered to be scalable to a great degree. It's currently running on one machine, but its architecture provides for easy scaling.

Source Code

Source is available on the local GitHub instance.

Flume Node (ingress and egress)

For now, the Flume node takes in data and sends it to the Map Event Listener. The initial version uses an "exec" source to tail an apache/nginx log, but data can be accepted through any source supported by Flume (ng).

1. **Install Java runtime.**

```
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java7-installer
```

Verify Java is installed and running normally:

```
$ java -version
java version "1.7.0_17"
Java(TM) SE Runtime Environment (build 1.7.0_17-b02)
Java HotSpot(TM) 64-Bit Server VM (build 23.7-b01, mixed mode)
```

2. **Download Flume** (http://flume.apache.org/download.html) and extract into a program folder.

```
wget
http://www.gtlib.gatech.edu/pub/apache/flume/1.3.1/apache-flume-1.3.1-bin.
tar.gz
tar xzvf apache-flume-1.3.1-bin.tar.gz
```

3. **Create flume-env.sh file to make Flume the most comfortable it can be about the location of the Java runtime.**

```
nano conf/flume-env.sh
```

Here's a sample flume-env.sh file:

```
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# If this file is placed at FLUME_CONF_DIR/flume-env.sh, it will be sourced
# during Flume startup.

# Enviroment variables can be set here.

JAVA_HOME=/usr/lib/jvm/java-7-oracle

# Give Flume more memory and pre-allocate, enable remote monitoring via JMX
#JAVA_OPTS="-Xms100m -Xmx200m -Dcom.sun.management.jmxremote"

# Note that the Flume conf directory is always included in the classpath.
#FLUME_CLASSPATH=""
```

4. **Create Flume configuration file** in [flumedir]/conf folder.

```
nano conf/flume.conf
```

Example:

```
# Flume configuration file format:

# list the sources, sinks and channels for the agent
#<Agent>.sources = <Source>
#<Agent>.sinks = <Sink>
#<Agent>.channels = <Channel1> <Channel2>

# set channel for source
#<Agent>.sources.<Source>.channels = <Channel1> <Channel2> ...

# set channel for sink
#<Agent>.sinks.<Sink>.channel = <Channel1>

##############################################################

# TAIL source to Flume (I know, not reliable, but it's good for now...)

a1.sources = r1
a1.sinks = k1  #k2
a1.channels = c1

# Source is a command (in this case, TAIL of a file with extraction of
first value, the IP address)
a1.sources.r1.type = exec
a1.sources.r1.command = tail -F /var/log/nginx/de-clicktrack.log | awk '{
print $1 }'

# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

# Sink will output to TCP port (the DynMap listener) on port 4242
a1.sinks.k1.type = org.apache.flume.sink.TCPSink
a1.sinks.k1.channel = c1
a1.sinks.k1.hostname = 127.0.0.1
a1.sinks.k1.port = 4242

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1

# Sink also to console (debug only)
a1.sinks.k2.type = logger
a1.sinks.k2.channel = c1
```

5. **Copy TCPSink.jar into the Flume lib folder** (available in GIT).

```
cp TCPSink.jar [flumedir]/lib/
```

If you need to compile it, this is the command line:

```
javac -Xlint -cp
"/home/erik/code/DynMap/apache-flume-1.3.1-bin/flume-ng-core/src/main/java
:/home/erik/code/DynMap/apache-flume
-1.3.1-bin/flume-ng-sdk/src/main/java:/home/erik/code/DynMap/apache-flume-
1.3.1-bin/lib:/home/erik/code/DynMap/apache-flume-1.3.1-bin/
flume-ng-configuration/src/main/java:/home/erik/code/DynMap/slf4j-1.7.2/sl
f4j-api/src/main/java:/home/erik/code/DynMap/protobuf-2.5.0r
c1/java/src/main/java:/home/erik/code/DynMap/guava-14.0-rc3:/home/erik/cod
e/DynMap/commons-io-2.4-src/src/main/java" TCPSink.java
```

Note that the command above references these dependencies:

- the Flume runtime (which you should already have)
- Simple Logging Facade for Java (slf4j) source
- Google Protocol Buffers binaries (which you'll have to download and compile)
- Guava (the Google Core Libraries for Java)
- Apache Commons IO libraries

6. Start up Flume.

```
[flumedir]/bin/flume-ng agent --conf conf --conf-file
[flumedir]/conf/flume.conf --name a1 -Dflume.root.logger=INFO,console
```

That "-D" option can be left off, obviously, so you don't see all that Java gobbledygook. Here's a sample:

```
$ bin/flume-ng agent --conf conf --conf-file conf/flume.conf --name a1
Info: Sourcing environment configuration script
/home/ebertrand/apache-flume-1.3.1-bin/conf/flume-env.sh
+ exec /usr/lib/jvm/java-7-oracle/bin/java -Xmx20m -cp
'/home/ebertrand/apache-flume-1.3.1-bin/conf:/home/ebertrand/apache-flume-
1.3.1-bin/lib/*' -Djava.library.path= org.apache.flume.node.Application
--conf-file conf/flume.conf --name a1
```

Map Event Listener & Map Event Server

1. Install Node.js (https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager):

```
sudo apt-get install python-software-properties python g++ make
sudo add-apt-repository ppa:chris-lea/node.js
sudo apt-get update
sudo apt-get install nodejs npm
```

2. Install socket.io and geoip-lite packages.

```
npm install socket.io geoip-lite
```

3. Verify the free city database exists within the ?[geoipdir]/data folder (geoip-city.dat).

```
$ find node_modules/geoip-lite/data
node_modules/geoip-lite/data
node_modules/geoip-lite/data/geoip-country6.dat
node_modules/geoip-lite/data/geoip-country.dat
```

If the geoip-city files are not there, retrieve the latest:

```
wget -O geoip-lite.zip
https://github.com/bluesmoon/node-geoip/archive/master.zip
# Extract the two "geoip-city*" files from the enclosed data folder, into
node_modules/geoip-lite/data
```

4. **Download source (dynmap-server.js from GIT) and copy into an application folder.**

5. **Open up ports if needed (one for the Event Listener, another for the Web Server).**

6. **Start up the listener/web server:**

```
node dynmap-server.js
```

Web Client

1. **Install nginx.**

```
sudo apt-get install nginx
```

2. **Configure nginx** with a vhost, etc., and ideal performance, caching, and other measures if you want. Sample nginx.conf (or sites-enabled/etc..) excerpt:

```
server {
    root /home/ebertrand/dynmap-server/www;
    index index.html index.htm;

    # Make site accessible from http://localhost/
    server_name dynmap.dyndns.com;

    # Performance optimizations (optional)
    add_header  Cache-Control public;
    expires     max;
}
```

3. **Bring up the map! http://dynmap.mht.dyndns.com**

(***NOTE:*** this won't work unless you have "dynmap.dyndns.com" aliased to the server on which the map server is running. It's in final verification, so see me for info on that.)