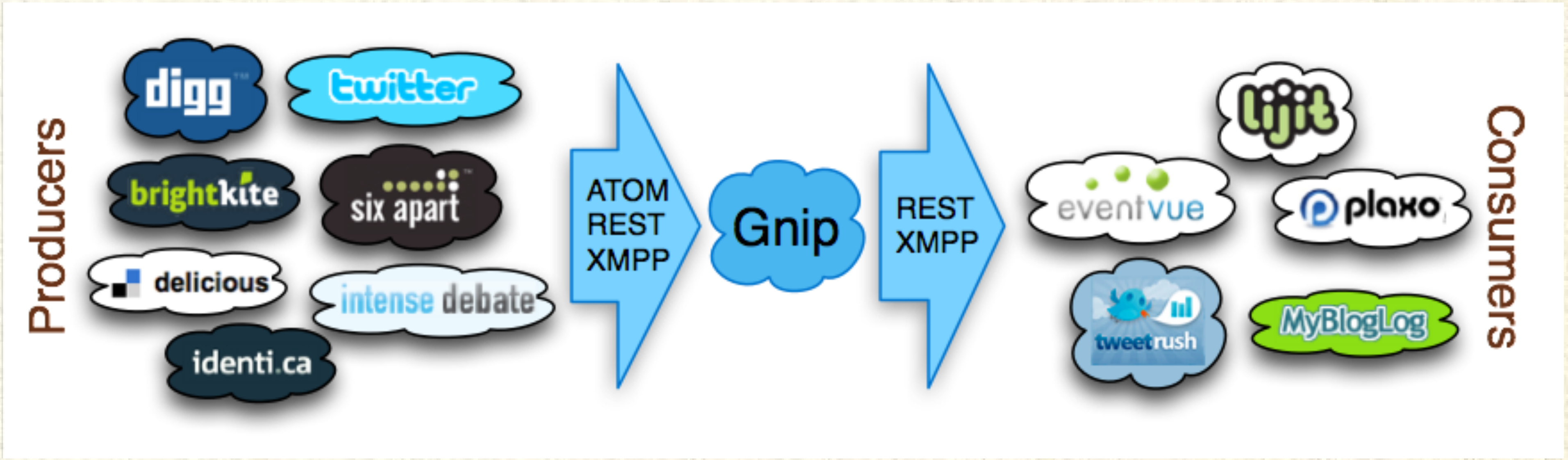

Gnip

(guh-nip)

Grand Central Station for the Social Web

presented by John Kelley

Making data portability suck less.

Polling for updates:

- Problem
  - Web 2.0 environment is mostly Pull data delivery
  - Ping blizzards tax server resources
  - waste of bandwidth

- Solution
  - Publish/Subscribe data delivery
  - Producers push data to Gnip who then pushes to Consumers
  - In realtime

Making data portability suck less. -cont

Protocol mismatch:

- Problem
  - Web app only understands protocol X, you want to talk in protocol Y
- Solution
  - Gnip sits in the middle and translates back and forth

Standardized Metadata:

- Problem
  - Different services label similar data differently
  - Hard to figure out what is being said
- Solution
  - Create meta data standard and make available with original

Making data portability suck less. -cont

Identity discovery:

- Problem
  - Hard for services and users to tie multiple accounts together from around the web.
- Solution
  - Gnip can use usernames and emails to check where else those identifiers are being used.

Current Gnip API v2.0 (rev2)

Provides two major components:

- change notifications for activities (events)
- full content associated with those events
- i.e. : new blog post, a user digg, twitter notice

Primary roles of API users:

- Publisher   - push data into activity streams
- Subscriber - consumes data from a Publishers activity stream
- (everyone needs an account to do anything, of course)

Current Gnip API v2.0 (rev2) - cont

Activity Streams (two types):

- Public Timelines
- Filters

Public Timelines

- stream of all activities from a given Publisher
- change notifications only, no full data
- not supported by all Publishers
- can only be polled (HTTP GET)

Current Gnip API v2.0 (rev2) - cont

Filter

- custom stream containing all activities that meet Subscriber defined criteria (ie. User X's friends)
- option to contain full data
- only Subscriber who created Filter can use it
- can be polled (HTTP GET)
- can be pushed (HTTP POST) to Subscriber who gives URL endpoint during Filter creation

Some Examples: Get Activity

Retrieve recent Activities for a given Publisher :

```
   ===>
     GET /publishers/digg/notification/current.
xml
     Accept: application/xml
```

Some Example: Get Activity

```
<---
      200 OK
      Content-Type: application/xml

      <activities publisher="digg">
       <activity source="web" regarding="http:
//services.digg.com/story/8571625" to="" url="
http://services.digg.
com/story/8538612/comment/18959806" action="
comment" actor="cryosteel" at="2008-09-19T16:
20:22.000-04:00"></activity>
      <activities>
```

Some Example: Create Filter

Create a Filter and have its Activity POSTed to specified URL:

```
===>
POST /publishers/digg/filters.xml
Accept: application/xml
Content-Type: application/xml

<filter name="example" fullData="true">
<postURL>http://mysite.example/inbound-activity-handler.cgi</postURL>
<rule type="actor" value="joe"/>
<rule type="actor" value="jane"/>
</filter><---
200 OK
Content-Type: application/xml

<result>Success</result>
```

Some Example: Filter Push Delivery

Example HTTP POST Exchange:===>

```
POST http://mysite.example/inbound-activity-handler.cgi
Content-Type: application/xml

<activities publisher="digg">
<activity source="web" regarding="http://services.digg.com/story/8571625"..
<payload>
<body>So in summary, we have two choices. Allow routine catastrophic...
<raw>gzip'd, base64'd original activity meta-data</raw>
</payload>
</activity>
</activities>
<---
200 OK
```

Reference

- Main site: http://www.gnipcentral.com
- API Documentation: http://docs.google.com/View?docid=dgkhvp8s_5svzn35fw