



loggly

Lucene Revolution
San Fran 2011

Using SolrCloud for real

Whoggly?



- I'm Jon, a happy lucene hacker since 2004
- We do Logging as a Service (SAAS with a single focus)
 - Consolidation, Archiving, Search, Alerting (soon!)
 - You stream your logs to us, we index them, you search
- Full public launch on Feb 2, 2011
- Every customer has their own index
 - ~1500 customers, ~8k shards, ~7B docs, ~3TB index
- Search finds the splinter in the log-jam
 - What happened? When? How often?

Loggly | Shell

https://geekceo.loggly.com/shell/#/_graph 404

loggly

From: NOW-1HOUR Until: NOW Inputs: All

```

3.0.30729; .NET4.0C)" RG4AxX8AAAEAAAxAUaUEAAAAT 404
2011-04-26 14:48:51.076 loggly_web 174.129.233.147
207.46.204.180 - - [26/Apr/2011:14:48:50 -0700] "GET /robots.txt HT
" "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingb
Results from range 2011-04-26T21:20:04.463Z - 2011-04-26T21:48:51.076Z for term
kordless@geekceo> graph 404

```

Events from NOW-1HOUR until NOW

kordless@geekceo>

Loggly | Main Dashboard

https://geekceo.loggly.com

profile | help | logout

kordless@geekceo> search for broken things

dashboard inputs users account monitor

Log History

73,608 events

Inputs

Name	Events In Last Day
default	5,837
gecko_http	0
geekceo_http	0
kedge_http	6
logg_ly	51,038
loggly_web	15,212
port514	0
titus	1,571

Devices

Name/IP	Events In Last Day
72.14.194.33	34,175
72.14.194.17	16,863
Loggly's web server	15,212
Loggly Office	5,837
98.210.51.152	1,577

Daily Usage

Today's usage: 11 MB

Legend: Dropped (red), Indexed (blue)

Time is on our side...

- We're not solving a typical search problem
 - ▶ Endless stream of data (all your logs)
 - ▶ Time is our “score”
 - ▶ Write once, update never
 - ▶ Large number of very small documents (events)
 - ▶ Search is mostly about what just happened
- So, simple index life-cycle with “natural” sharding
 - ▶ For us, a shard is slice of a single customers index, based on start and end time

Why SolrCloud?



- The wheel existed, and keeps getting better
 - Thanks to the community. Big Thanks to Mark & Yonik
- Solr: Multi-core, Plugins, Facets, migration, caching, ...
 - Our Solr instances have hundreds of cores (one per shard)
 - We've made very few changes to core Solr code
- Zookeeper: state of all nodes & shards, so...
 - Automatic cluster config
 - Cluster & Shard changes visible everywhere, almost instantly
 - Any node can service any request (except for indexing)

Cluster Management



- Solr instances register & deregister themselves in ZooKeeper
 - always know what Solr instances are available
- All Solr configs in ZK except for solr.xml
 - all instances use same schema, etc, so simple management
 - solr.xml is “special”, instance specific
- We’ve added our own persistent data
 - Loggly-specific config, and some performance data for Solr
 - Other app configs, “live” status

Index Management

- One Collection (“index”) per customer
 - Sharded by time, multiple shards per customer
- Shards migrated from node to node using replication
 - Minor changes to existing Solr replication code
- Shards merged by us, never automatically
 - We merge to create longer time-slices
 - Doing it manually makes merge load more predictable
- Completely distributed management, no “Master”
 - Simple, Robust, “need to know”

SolrCloud, meet reality

- Day 1 (patch to trunk, 18 months ago), almost everything JFW'ed
- The exceptions...
 - We had to fix a couple of TODO's
 - We select shards for search a little differently
 - We hit a performance wall
 - We added some utilities to the ZK controller/client
- None of these were difficult
 - Today, everything **does** JFW (for us)

TODO's

- Very little missing, even 18 months ago...
- *FacetComponent.countFacets()*
 - ▶ `// TODO: facet dates`
 - ▶ We did it. Since been added to trunk (not our code)
- *ZkController.unregister()*
 - ▶ `// TODO : perhaps mark the core down in zk?`
 - ▶ We did it: remove the shard from ZK, walk up the tree removing its parents if they're empty
- One more, but no spoilers...

Shard selection

- *QueryComponent.checkDistributed()* selects shards for each “slice”, based on the Collection of the core being queried.
- We changed some things for version 1...
 - use a “loggly” core, and pass in the collection
 - select the biggest shard when overlaps occur
 - select the “best” copy of a shard on multiple machines
- Now we use plugin variant of admin handler
 - avoids special case core
 - lets us do shard-level short-circuiting for search (not facets)

Performance Fun

- ZkStateReader...
 - `// TODO: - possibly: incremental update rather than reread everything?`
 - Yep, EVERYTHING in ZooKeeper, every time we update anything
 - to be fair, we're kind of a pathological case
 - Watchers for every collection, every shard
- The Perfect storm
 - Full read of ZK with 1000's of shards ain't fast
 - We weren't using scheduled updates correctly
 - We're updating frequently, triggering lots of Watcher updates
- Up to 20 second wait for CloudState

“Just avoid holding it in that way”

- Watch fewer things
 - Every node doesn't have to know about every shard change
- Incremental updates
 - When a watch triggers, we rebuild only that data
- On-demand Collection updates
 - We read individual Collection info whenever its needed
- **Wait for CloudState is now 10's of milliseconds**
 - 200–400 CloudState updates / minute

ZK Utilities

- Cleanup
 - **nuke**: `rm -rf` for the ZooKeeper tree
 - **purgeShards**: delete all shards for collection X
 - **purgeNode**: delete all references to node Y
- **upload**: upload a file or directory
- **loggly_node**: our config secret sauce
 - based on `ZkNodeProps`

Loggly magic

- We've spent most of our time on Shard Management
 - Plugins FTW
- Minimal changes to Solr itself
 - 0MQ streaming data input
 - More logging, to verify our Shard Management is working
- Lots of admin API extensions
 - Lets our other apps tell Solr what to expect
 - Lets Solr tell other apps whats going on
 - Lets us fix things by hand when things go wrong

Loggly ZK magic

- We use ZK to store indexing performance data
 - lets us load balance our indexers
- We use ZK in our other apps
 - we have “live_nodes”++ for ALL apps
 - one-off’s easy when entire state of the system is available
 - json output + ruby + REST = easy-peasy
- ZK is very robust
 - transitioned 5-node ZK cluster to all new hosts with 0 impact

Wish List

- Standalone jar for the Zk* classes
 - Simplify access to the Solr specific data in other apps
 - Re-use SolrCloud wrappers around ZK (which are nice)
- Better control of watchers
 - watching too many things considered harmful
- Plugin shard selection for search
 - Hacking existing QueryComponent or replacing it entirely are both kind of brute-force.
 - Maybe this just us though

Other Stuff We Use

- Amazon's AWS for infrastructure (EC2, S3)
- Syslog-NG for syslog/TLS input services
- 0MQ for event queuing & work distribution
- MongoDB for statistics and API /stat methods
- Node.js for HTTP/HTTPs input services
- Django/Python for middleware/app

loggly

ACK/FIN

One Last Thing... <http://loggly.ly/jobs>