glu

deployment automation platform

July 2011

Yan Pujante

http://www.linkedin.com/in/yan

blog: http://pongasoft.com/blog/yan

@yanpujante

* To see a video of this presentation given at Chicago devops, check this link:
• to see a video of this presentation given at Chicago devops, check this link:

A little bit about me...

• Software engineer (16 years experience)

• Software is my passion (28 years! TI-99/4A)

• Currently not working... for a boss... :)
  
  • glu, kiwidoc (www.kiwidoc.com)

• Worked @ LinkedIn for 8 years (founding team!)
  
  • Worked on a lot of infrastructure projects and early features (security, payment, graph, etc...)

• Last (big) project was glu (main author/contributor/maintainer)
Why glu?
Before glu... :('}
Before glu...

• Operations performs manual deployment:
  • ssh, rcp, etc...
  • non shared manually edited scripts
  ➡ extremely time-consuming
  ➡ error prone
• Address operations pain points

• Deploy (and monitor) applications to an arbitrary large set of nodes:
  • efficiently
  • with minimum/no human interaction
  • securely
  • in a reproducible manner

• ensure consistency over time (prevent drifting)

• detect and troubleshoot quickly when problems arise
After... :) Nothing to do here... Sit back and enjoy!
After... :D

Deploy - Fabric [glu-dev-1] - PARALLEL

Show Errors Only: ☐  Auto Refresh: ☑

12/12 - 100%

Deploy - Fabric [glu-dev-1] - PARALLEL

<table>
<thead>
<tr>
<th>agent-1 - /sample/i001 - 16s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install script for [/sample/i001] on [agent-1] - 1s</td>
</tr>
<tr>
<td>Run [install] phase for [/sample/i001] on [agent-1] - 3s</td>
</tr>
<tr>
<td>Run [configure] phase for [/sample/i001] on [agent-1] - 2s</td>
</tr>
<tr>
<td>Run [start] phase for [/sample/i001] on [agent-1] - 8s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>agent-1 - /sample/i002 - 12s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install script for [/sample/i002] on [agent-1] - 2s</td>
</tr>
<tr>
<td>Run [install] phase for [/sample/i002] on [agent-1] - 2s</td>
</tr>
<tr>
<td>Run [configure] phase for [/sample/i002] on [agent-1] - 1s</td>
</tr>
<tr>
<td>Run [start] phase for [/sample/i002] on [agent-1] - 5s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>agent-1 - /sample/i003 - 11s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install script for [/sample/i003] on [agent-1] - 2s</td>
</tr>
<tr>
<td>Run [install] phase for [/sample/i003] on [agent-1] - 2s</td>
</tr>
<tr>
<td>Run [configure] phase for [/sample/i003] on [agent-1] - 1s</td>
</tr>
<tr>
<td>Run [start] phase for [/sample/i003] on [agent-1] - 4s</td>
</tr>
</tbody>
</table>
History of glu

- July 2009: glu project started
- March 2010: limited rollout to production
- July 2010: 100% rollout
- November 2010: glu open source
- June 2011: latest release 3.0.0
- July 2011: Orbitz tech Talk
- September 2011: glu ? :)

Monday, July 11, 2011
Rollout to production

- glu project started in July 2009
- Initial rollout to LinkedIn production in March 2010
- Gradual until full rollout in July 2010
- As of June 2011 LinkedIn glu numbers:
  - 5 different ‘fabrics’ (2 prod + 2 stg + 1 int. lab)
  - ~2650 nodes, ~9000 instances, ~300 services
- LinkedIn working on ‘glu on the desktop’ (dev)
• Before I left LinkedIn, open sourced glu (~3 months effort)
  • 1.0.0 released in November 2010
  • 2.0.0 released in February 2011 (tagging)
  • 3.0.0 released in June 2011 (parent/child)
  • (~ 20 releases total... smaller releases)
• since 11/2010, glu has generated a lot of interest
  • oubrain.com is using glu (integrated in CI!)
  • companies interested in glu: Orbitz, Netflix, GigaSpaces, Rearden Commerce, etc...
  • some academic use (Budapest university)
  • a lot of ‘followers’ on github
  • lots of downloads
Architecture
Components/Concepts

- 3 physical components
  - Agent
  - Zoo Keeper
  - glu orchestration engine

- 3 concepts
  - Static Model
  - Live Model
  - Script
• 1 ZooKeeper cluster (3 or 5 instances enough)
  • ZooKeeper is an Apache project
  • similar to a (networked) filesystem (think nfs)
    • ‘directories’ can also contain data
    • ephemeral nodes
    • powerful watcher concept => notifications
  • ZooKeeper is used to maintain the state of the system
glu Agent

• 1 agent per node => as many agents as there are nodes

• agent is active process (groovy)

• (secure) REST API

• Reports its state to ZooKeeper
• 1 orchestration engine
• runs inside a webapp
  • offers both browser and REST interface
• Listens to ZooKeeper events (to compute ‘live state’)
• Talks to the agents
• model is a json document which describes
  • where to deploy
  • what and how to deploy
• “Static” is what you want
• “Live” is what is actually deployed/running
Static Model: Where?

```
{
    "fabric": "prod-chicago",
    "entries": [{
        "agent": "node01.prod",
        "mountPoint": "/search/i001",
        "script": "http://repository.prod/scripts/webapp-deploy-1.0.0.groovy",
        "initParameters": {
            "container": {
                "skeleton": "http://repository.prod/tgzs/jetty-7.2.2.v20101205.tgz",
                "port": 8080,
            },
            "webapp": {
                "war": "http://repository.prod/wars/search-2.1.0.war",
                "contextPath": "/
            }
        }
    }]}
```

- “agent” => node which runs this agent
- “mountPoint” => unique key
- can deploy more than 1 ‘thing’ per agent
• "script" => instructions about what 'deploy' means

• "initParameters" => parameters provided to the script
• groovy class which defines

• a set of ‘phases’ (install, start, etc...) backed by a state machine

• properties (exported to ZooKeeper)

• glu does not dictate what goes in each ‘phase’
• glu Script code runs inside the (java) VM of the agent

• in general, a glu Script will spawn external processes (ex: webapp container, memcached, etc...) but it is not a requirement!
How does it all work?
• each agent reports its state to ZooKeeper

• the orchestration engine listens to ZooKeeper and builds the ‘live’ model
the ‘static’ model is loaded in the orchestration engine
• orchestration engine computes a delta by comparing the static model and the live model

• “desired” state vs “current” state
• delta is used to compute a deployment plan

• orchestration engine sends commands (REST) to the appropriate agents
as the agents run the commands they update their state in ZooKeeper
The live model and the static model match

=> no more delta
System Stable (no delta)

- remains stable until:
  - static model changes (ex: new version of software)
  - live model changes (ex: hardware crash)
• Static model changes

• ex: new version of software, new node, etc...

• => delta => deploy/upgrade software, provision new nodes
• Live Model changes
  
  • ex: hardware crash, bad behavior, high load, etc...
  
  • => delta => monitoring!
Monitoring: built-in

- agent registers a ZooKeeper ephemeral node
- => when agent disappears, state changes!
Monitoring: add-on

• script runs in "active" agent

• agent has "timer" capability

• =>script can also monitor what it starts and change state when failure detected
Monitoring: advanced

• You can even build a full monitoring solution on top of glu

• Not enough time/space here :)

• Check out my blog (source examples included!) @ http://www.pongasoft.com/blog/yan/categories/glu/
What about security?
• User must authenticate (LDAP and/or glu)

• Agent REST API is ‘protected’ behind HTTPS with client auth

• Every ‘change’ is audited in the audit log
Live Demo...

* You can see the live demo in the presentation given at Chicago devops (starts around 27:00):
glu as a platform
glu is more than a tool

• glu is a tool with a lot of customization points

• it is also a **platform** on top of which you can build your **own** deployment (and optionally monitoring) solution
• Agent CLI and Console CLI are mostly wrappers/examples around the REST API

• => you can use the REST API directly or use the CLI
A glu script is any code you want (groovy/java) made easier by agent capabilities (but you don’t have to use them!)

- `shell.exec` capability allow you to write your script in any language you want (will be ‘promoted’ native soon...)

```ruby
class RubyGluScript {
  def install = {
    shell.exec("./ruby/install.rb")
  }
  def start = {
    shell.exec("./ruby/start.rb")
  }
}
```
• One way to look at the agent: script engine remotely accessible through a (secure) REST API

• => can also be used on its own (no ZooKeeper or orchestration engine)
ZooKeeper is independently accessible

- => can build your own listeners/watchers directly

- => use AgentsTracker library which comes with glu (check the blog for more details)

- Ex: build a monitoring solution
• For example, you can integrate your CI directly with glu by using the orchestration engine REST api (ex: outbrain.com)

• Although very customizable, you can also build your own UI if you do not like the one that comes with glu
• Powerful tagging/filtering feature allow to create concepts that glu does not know about (ex: webapp, frontend, cluster, etc...)

• Query language allows you to slice & dice the models

• => build higher level constructs (like dynamic node assignment)
 Disclaimer: I have spent 2 years with glu (I wrote it :-) ) and 1 day with puppet...
• Great news: intrinsically similar concepts
  • ‘desired’ vs ‘current’!
  • declarative approach
• Minor difference:
  • puppet is ruby vs glu is groovy/java
glu vs puppet: orchestration

- Delta computation / orchestration takes place at a different level
  - => glu can orchestrate across nodes
  - => glu delta is system wide (and real-time)
glu vs puppet: conclusion

- puppet is very good at **configuring** the infrastructure of a machine (users, groups, packages, etc...)
  - => static/stable does not change often

- glu is very good at **provisioning** dynamic applications on an ensemble of machines (the system)
  - => changes often, real-time failure detection (monitoring), “bounce”, etc...
class PuppetGluScript {
    def puppetManifest

    def install = {
        // download manifest
        puppetManifest = shell.fetch(params.puppetManifestURI)
    }

    def start = {
        // execute manifest
        shell.exec("puppet apply ${puppetManifest}"
    }
}
References

• glu source: github.com/linkedin/glu (links to all you need)

• blog: www.pongasoft.com/blog/yan

• twitter: @glutweets