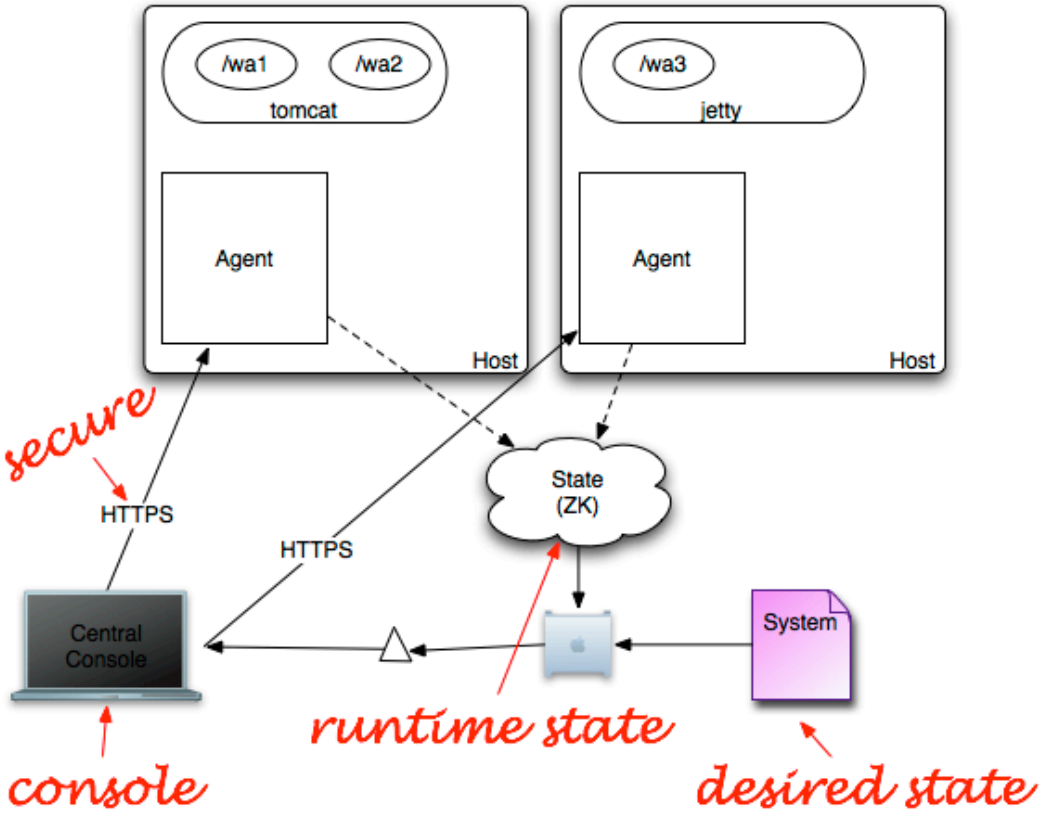
GLU Overview

Introduction
GLU is a deployment automation platform. It has been built and deployed at LinkedIn in early 2010 and then released as open source in November 2010. The goal is to be able to automate the deployment of any kind of applications accross many nodes. Although written in groovy/java, the type of applications that can be deployed through GLU is not limited to java applications. GLU is a platform and the way it was architected and designed allows you to pick and choose which part you want to use. For example, if you want to use the glu agent only without ZooKeeper and build your own tooling on top of it, you can do so.

Architecture overview



This diagram shows an example of setup, the one used at LinkedIn:

- Each host runs the GLU agent which connects to ZooKeeper to report its state (the 'runtime state')
- The system (also known as 'desired state') describes which applications need to run on which hosts as well as how to install and run them. The system is a simple json document.
- A console (web application) which provides a central access point to control the system: it connects to ZooKeeper to track the 'runtime state' and knows about the 'desired state'. It then computes a delta between the 2 states to determine what actions need to be taken in order to 'fix' the delta. Actions are commands sent to individual agents over https.

GLU Agent
The GLU Agent is an active process which connects (optionally) to ZooKeeper to report its state and provides:
- a REST api (secure over https to allow only authorized clients)
- a command line to directly access the agent remotely (uses the REST api)

ZooKeeper
ZooKeeper (http://hadoop.apache.org/zookeeper/) is used to maintain the state in a central location and is used for its powerful notification capabilities (ephemeral nodes and watchers). ZooKeeper is required if you are also using the console otherwise it is optional if you use only the GLU agent.

Console

The console is the orchestrator: it connects to ZooKeeper to keep track of the entire runtime system state in real time (in a very lightweight fashion) and uses the information coming from the static system state (desired state) to compute a delta in order to:

* display discrepencies (dashboard view)

container:40	I:40	E:16	agent:14	release:1	cluster:0	status:4
abook	1	0	ei2-app1-zone1.qa	R949		running
activemq	1	0	ei2-app1-zone1.qa	R949		running
anet-cloud	1	0	ei2-app2-zone1.qa	R949		running
anet-databusrelay	1	0	ei2-app1-zone4.qa	R949		running
appreg	1	0	ei2-app3-zone3.qa	R949		running
auth-server	1	1	ei2-app3-zone2.qa	R949		NOT deployed
comm-dispatcher	1	0	ei2-app2-zone1.qa	R949		running

* trigger the set of actions to run on each agent to 'fix' the delta

ei2-app3-zone2.qa

/auth-server/i001

1. install GLU script - R949 | JettyServerWithWarsScript-1.4.0.groovy
2. install container - R949 | container-jetty-0.0.517-RC1.2368.tar.gz
3. install wars - R949 | security-auth-war - 0.0.504-RC4.2617|auth
4. start container - R949

/sas-jobs/i001

1. start container - R949

ei2-app3-zone5.qa

/ptagent-server/i001

1. start container - R949

The console has 3 main entry points:

- a webapp to use with a regular browser
- a REST api
- a command line (uses the REST api) (coming soon)

The console offers the following capabilities:

- user authentication and management (ldap or console password)
- auditing (to keep track of who does what and when)
- access to all agents functionalities (like viewing log files and displaying folders, killing processes…)
- configurable to suit your needs in terms of what gets displayed and in which order
- parallel deployment accross any kinds of node
- powerful filtering capabilities (allow to create notions like cluster for example)