OPEN SOURCE
CLOUD COMPUTING FORUM

February 10, 2010

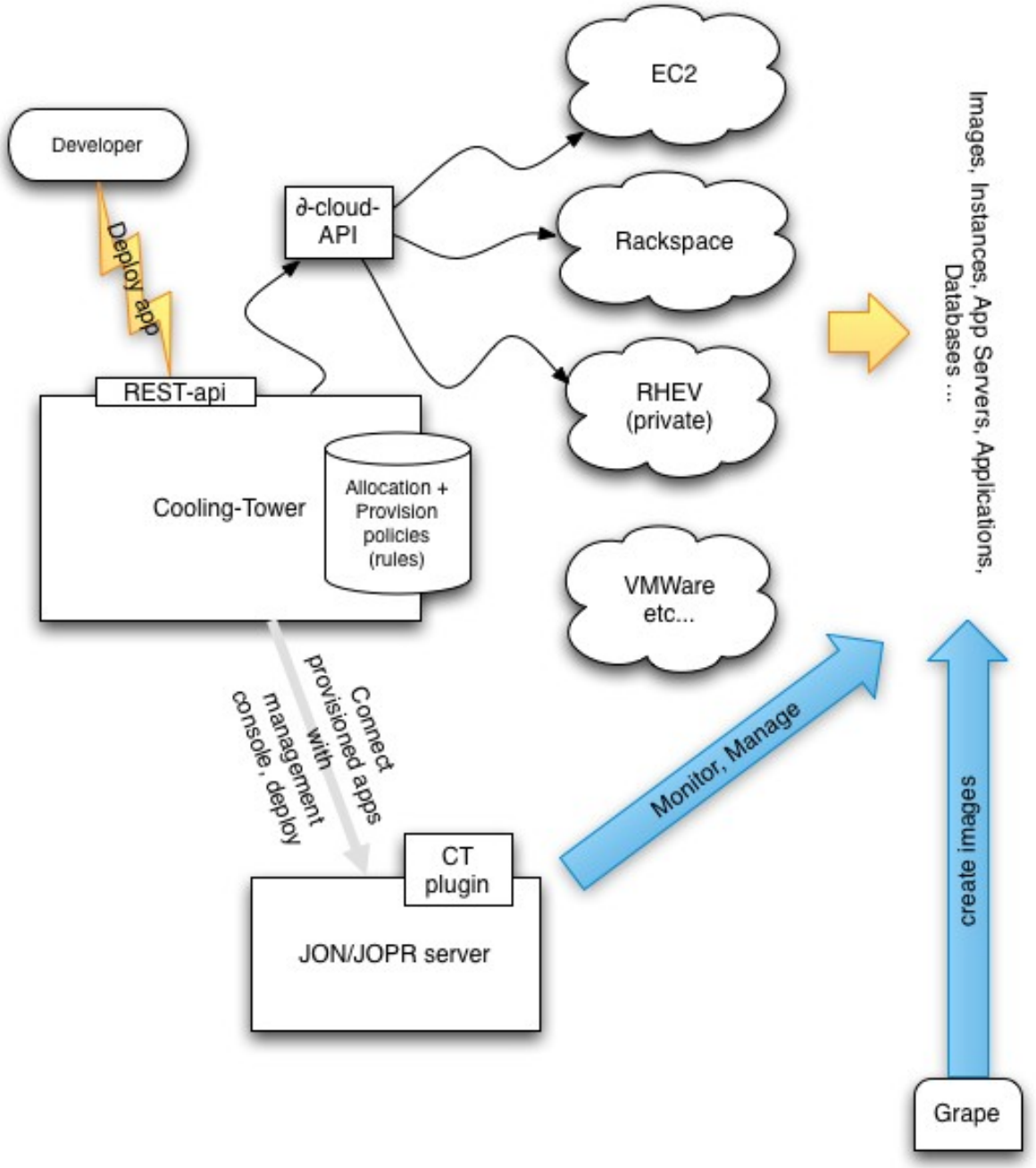
# CoolingTower

Michael Neale
Senior Engineer, Red Hat (JBoss)

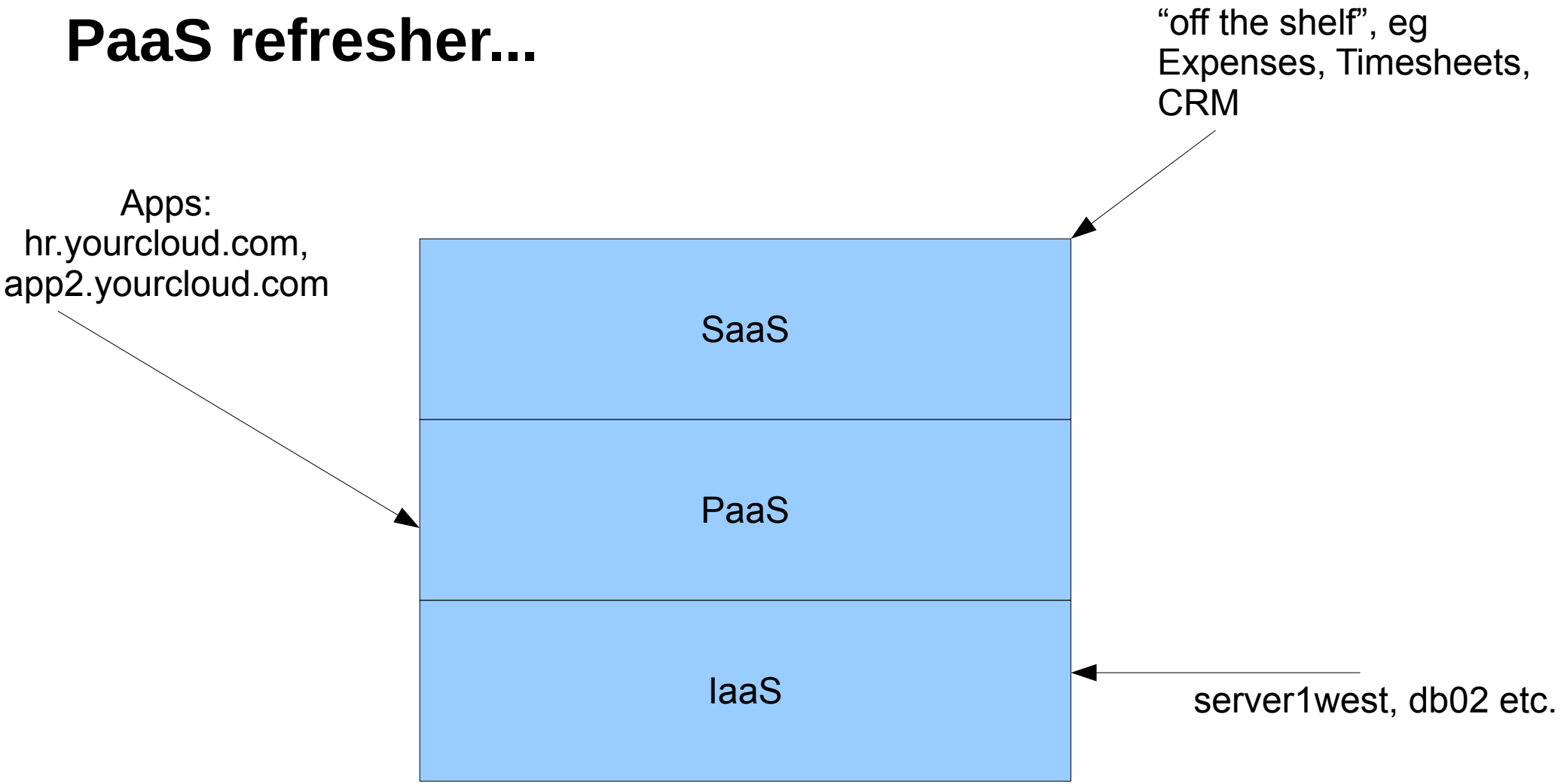
# CoolingTower: Turnkey PaaS

- What is CoolingTower
  - A platform-as-a-service for Java Server side apps (Servlet/EE container) – completely self service
  - Application centric (it is a PaaS) not server/infrastructure centric
  - Is a project of utilities and aggressive automation of existing (or soon to be existing) components
    - Runs as a server component with RESTful API
  - Policy driven
  - Driven by RESTful apis, multiple front ends
  - Using open API's such as deltacloud for multiple IaaS platforms

# CoolingTower:



# PaaS refresher...

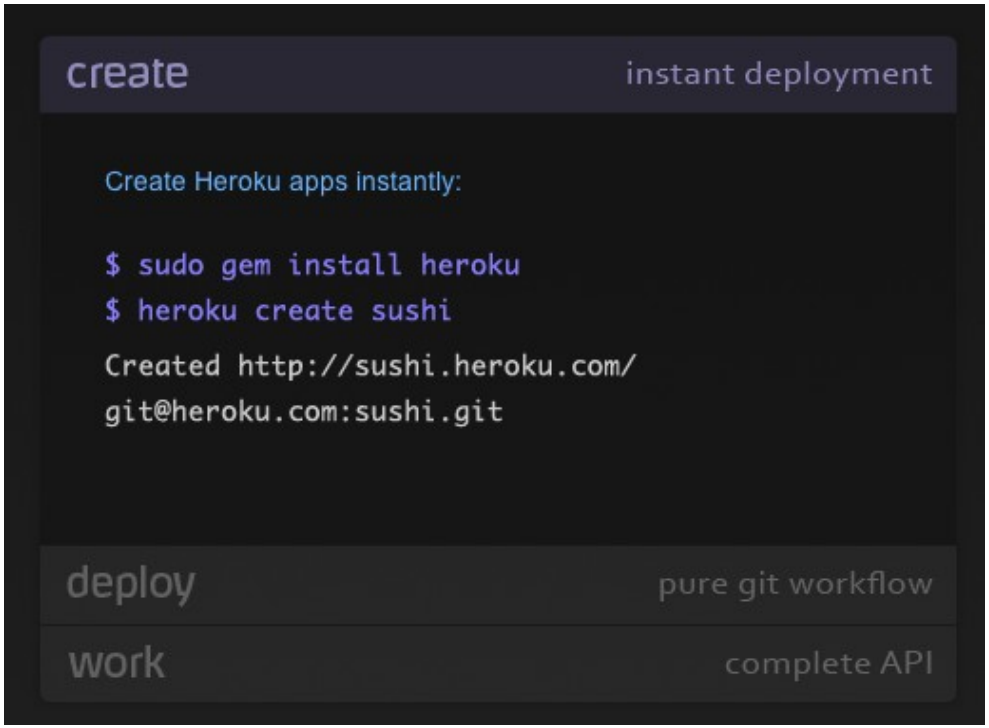


# Motivation

- Many clouds are infrastructure and operations centric
- A space exists for application and developer centric clouds
  - Yet adhering to organisational policies, workflows for application lifecycles etc.
  - Automation !
- Build on standards !
  - emerging for clouds, and for apps

# Public PaaS already in use

- Heroku.com
  - Ruby based



# Google App engine

- Java or Python editions
  - Free or Pay (based on quotas)
  - Can run other languages via JVM

**Run your web apps on Google's infrastructure.**
Easy to build, easy to maintain, easy to scale.

# What do they have in common?

- Stunning developer experience (ease, low/no barrier)
- Instant results (via a nice URL)
- Some ability to scale elastically *
- NO management of server resources *
- Infrastructure abstracted (automated) away
  - You don't manage/see servers
- Very good uptake for small/medium apps
- Very ATTRACTIVE to **developers**
  - **(who don't want to administer !)**

# Limitations (not all bad)

- Often a subset of a “full stack”
  - Must stick within API
  - Locked to specific platform versions
  - Build apps to scale
- Run on public infrastructure (shared)
- Follow the conventions, live with with specific versions
- Don't have usual “enterprise” needs for promotion, QA environments etc

# CoolingTower - motivations

- Use standards as much as possible
  - JavaEE, Servlet API (JVM allows many languages now)
  - Deltacloud for IaaS layer
- A PaaS cloud is a PaaS cloud: avoid lock in to IaaS layer
- Allow PaaS cloud to be built on private infrastructure, or public
- Address “enterprise concerns”
  - Use configurable deployment and management policies
  - Allow promotion through cloud “environments” (dev/test → QA → production etc).

# Further motivations

- In enterprise, apps need to scale in many directions:
  - Up: lots of users
  - Deep: complex app with lots of calculations (but maybe not many users)
  - Many: many small departmental apps – come and go.
- Reality: many apps don't need to scale like GAE style apps do (but high availability is very nice if possible !)
- With JEE/JBossAS we get relatively easy app tier clustering

# CoolingTower - components

- ***Deployment API***
  - RESTful api, Policy driven deployment
- ***Elastic scaler***
  - Policy driven horizontal scaling of clustered apps
- ***Server minimiser***
  - “densely” pack applications for development clusters
- ***Naming Service***
  - RESTful api for self service of application names
  - Authoritative only DNS
- ***Clients: Web, Eclipse, RESTful***

# CoolingTower - deployment

- Applications packaged up in standard war/ear (optional extra meta data)
- Deploy via RESTful API

**Usage of REST api (draft)**

```
GET /cooling-tower -- get a list of links to something like the following:
POST /cooling-tower/applications --- post the war/app contents, will return a link to status
GET /cooling-tower/applications/your-app -- return the status of it (where it is running, if it is ready etc)
POST /cooling-tower/applications/your-app -- update the version
DELETE /cooling-tower/applications/your-app -- decommission the app (may result in some server savings)
GET /cooling-tower/applications -- list the apps available and status
```

# CoolingTower - deployment

- On request:
  - service applies policy rules, finds a running instance, or starts instances as needed (using CirrAS to start a cluster up of app servers)
  - deploys application, runs promotion “tasks”
- Track status of applications via RESTful call
- Update an application easily, decommission
- Policy could enforce what certain developers can deploy to – notify people of new version
  - Workflow: new version, approval, promotion to QA, PROD etc.

# Deployment - policies

- Currently uses deltacloud “model”
- Policies are declarative rules !
- Selection of appropriate image, or running instance

# Deployment - policies

```
rule "Can't colocate - so need a new server"
    when
        application: Application(canCoExist == false)
        not Instance(numberOfApps == 0,
                     spareMemory >= application.memory,
                     spareStorage >= application.disk)
    then
        insert(new NewInstanceNeeded(application))
end

#
#TIP: you can lock it down to specific image types here if you like...
#
rule "Can colocate - see if we can fit it in somewhere"
    when
        application: Application(canCoExist == true)
        instance: Instance(numberOfApps < 4,
                           spareMemory >= (application.memory * 1.2),
                           spareStorage >= (application.disk * 1.5) )
        #we want to make sure there is no other instance with less apps !
        not Instance(numberOfApps < instance.numberOfApps,
                     spareMemory >= (application.memory * 1.2),
                     spareStorage >= (application.disk * 1.5) )
        forall(Application(canCoExist == true) from instance.applications)
    then
        insert(new Assignment(application, instance))
end
```

# Declarative rules

- Like scripts, but far more powerful
- Combine data of cloud (images, instances) with application needs (meta data, other running applications) to reason over in rules
- Efficient use of cloud IaaS layer
- Rules, so customisable for a given cloud installation
- Cover:
  - Allocation for deployment, image selection
  - instance life cycle
  - application lifecycle (promotion)

# Roadmap: deployment front ends

- Eclipse
  - Developers deploy from eclipse to development cloud (whence workflow for promotion can be initialised)
- Command line
  - Scripts, ant (all good RESTful interfaces should be able to be driven from the command line)
- Web Client
  - Web based PaaS console for creating/updating apps

# Roadmap: data services

- Most practical applications require persistent data
  - NoSQL (Infinispan REST api – as well as cache)
    - But not for everyone
  - RDBMS – will need to offer a fixed set (Postgres likely, possibly MySQL in limited configuration)
  - Most JVM apps use ORM which provides RDBMS flexibility
  - TBD...

# CoolingTower – Elastic Scaler

- Once again, use declarative policy rules
- Commonly done with scripts: monitor load, grow and shrink as needed (application tier mainly)
- Rules will apply that, as well as conform to policy
  - limiting resource load
  - prioritising for important apps
  - attempt to anticipate load and scale up ahead of time (user driven, or based on historical data patterns).

# A note on app multi-tenancy

- Instances can run multiple application servers
- Application servers (1 process) can run multiple applications
- Need to have policies to decide what level of isolation is needed for each app.
- For even “lite” applications, can have 2 instances supporting it for HA reasons
  - So multi-tenanting often makes sense
- For development purposes, may want to multi tenant for efficiency.

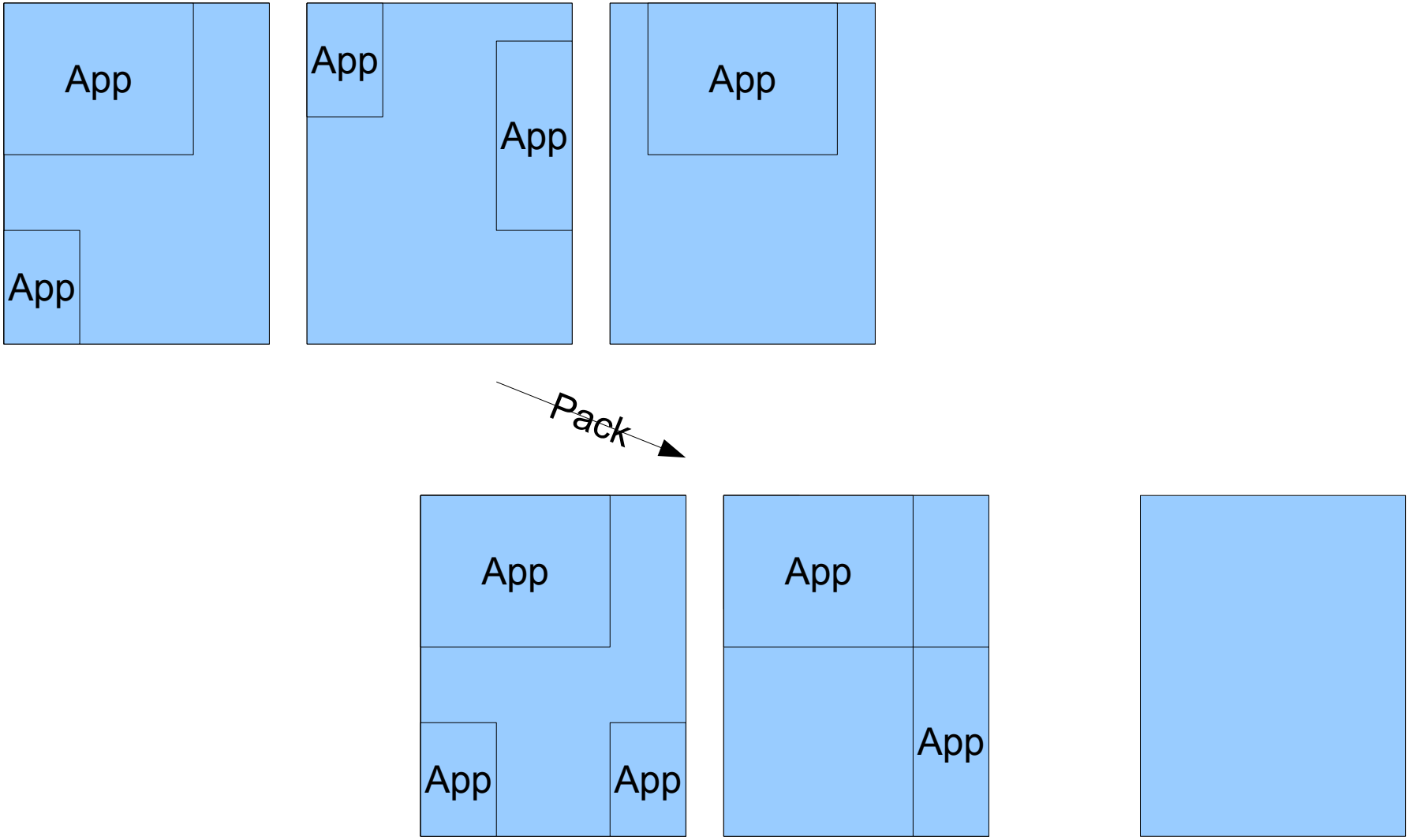
# CoolingTower – Server Minimiser

- Keep costs low (both in $ and resources)
- Many IaaS “bill” per-instance
- Keep instance numbers as low as possible for applications
- Mainly important for development clusters
- Each process/app has a footprint of memory, cpu, disk needs, an instance has finite amount of each
- Resembles the “bin packing problem”

# Minimiser

- We have a “constraint solver” (called Drools Planner)
- Declare rules to describe the hard and soft constraints
- Try multiple configurations of applications per instance to result in a solution with lower number of instances needed
- Could be run periodically (in the case of many apps coming and going)

# Minimiser



# Minimiser: roadmap

- Very experimental
  - Most likely of use for public IaaS where cost is clear. For private clouds, not as urgent
- May be important longer term
  - Minimise hardware foot print
  - Will need to take into account data we don't yet easily have (eg energy usage) or understand

# CoolingTower: Naming Service

- The “forgotten” cloud service
  - http://www.rackspacecloud.com/blog/2009/06/04/dns-the-overlooked-cloud-service/
- In a PaaS cloud, typically have many apps:
  - hr.yourcloud.com, incentive.yourcloud.com are nice
  - Web “entry-point” may move around
  - Apps can come and go
  - Eg EC2 “ugly” names, or rackspace IPs
- You may want to refer to “cloud services” via URLs, eg caching, vertical services etc

# Naming Service

- Typical process
  - When setting up a cloud, buy domain name
  - Register CoolingTower/PaaS clouds DNS servers with registrar you use (own DNS gives control)
  - CoolingTower does the rest
    - DNS can be error prone, so automate !

# Naming Service – a RESTful api for DNS

- Register a domain
  - POST /cooling-tower/naming name=yourcloud.com
- Set/update address of subdomain (or default)
  - POST ../yourdomain.com subdomain=applicationName&address=1.2.3.4
  - PUT ../yourdomain.com/www address=newAddress...
- (IP or domain name, either works)
- Many more REST verbs apply for modifying, removing etc.
- CoolingTower will call this when creating an instance etc.

# Naming – RESTful DNS

- CoolingTower stores DNS records in standard zone files
- Can rsync to BIND servers (so the “master” is hidden), or use built in authoritative server (which is updated instantly with changes)
- Use very short TTLs to allow services to move around in the cloud
- For both internal and external services

# Work to be done..

- Defining “cloud descriptor” for meta data needed to deploy
  - Include automating of data/migration scripts when “promoting” an application
- Defining common RDBMS services that cover enough use-cases
- Integration with CirrAS for deployment of applications into clustered App Servers.
- Capturing more policies/samples in rules
- Much more...

# Community: Getting involved

- Subproject of StormGrind @ JBoss
  - jboss.org/stormgrind/projects/coolingtower
    - Code currently on github/michaelneale/cooling-tower
    - wiki.github.com/michaelneale/cooling-tower
    - (will migrate to jboss.org)
  - IRC: #stormgrind on irc.freenode.net