adopting 'infrastructure as code'

to run HEP applications

Misha Zynovyev (m.zynovyev@gsi.de)
Dennis Klein (d.klein@gsi.de)
Victor Penso (v.penso@gsi.de)

HEPiX Spring 2011

GSI, Darmstadt, 5th of May 2011

acknowledgements

- Dr. Peter Malzacher
- Bastian Neuburger
- Christopher Huhn
- Dr. Stefan Haller
- Helge Brust
- Oliver Wirth

outline of the talk

- our mission statement
- current approaches to infrastructure management
- our approach to running HEP applications
- describing infrastructure with code
- our playgrounds
- results
- outlook

clarification

- target audience: developers, system administrators and infrastructure providers
- you accept the benefits of virtualization
- presented concepts are transparent to users[1]
- this is not about how to build the Infrastructure-as-a-Service (IaaS) clouds, but about efficient ways to use them

[1]nevertheless users can have as much control of details as they can cope with

our mission statement

we want to simplify deployment and operation of scientific computing applications and their infrastructure by describing everything in code

furthermore, we need a mechanism which will allow us to use external resources provided by IaaS clouds in a transparent way

deploying infrastructure

- manual (with OS tools, text-editor): ultra flexible, implicit knowledge, lack of history, scattered documentation, difficult to scale
- ad-hoc (scripting with remote login): more repeatable, built your way with your model, dispersed knowledge, little collaboration, no API, rarely idempotent[1]
- code (Chef, Puppet, Cfengine, Bcfg2, etc.): repeatable, idempotent, common configuration description language, explicit sharable knowledge, self-documenting, encourages collaboration, large initial investment in learning

[1]in practice, it means that an operation can be repeated or retried as often as necessary without causing unintended effects

operating infrastructure

- manual (spot problem, login, intervene): freedom to hack, easy to find somebody to blame, human factor, error prone, slow incident response, not repeatable, human factor
- ad-hoc (disposable helper-scripts): more repeatable, less error prone, tooling sprawl, hard to share solutions, time-consuming
- code (Capistrano, DSH, ControlTier, Func, etc.): repeatable, scales well, predictable results, not everything maps cleanly

running applications

- manual (start and take care about it): ultra flexible, time-consuming, difficult to scale
- ad-hoc (write a script to keep it running): use tools of your choice, not portable, complicated
- code (integrate with configuration and infrastructure management): fully automatic, scales well, self-documenting, sharable with others, large initial investment in learning

infrastructure as code

definition

"Enable the reconstruction of the business from nothing but a source code repository, an application data backup, and bare metal resources"

reference

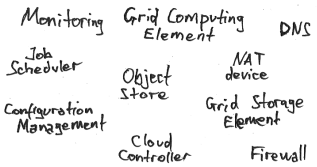
"Web Operations, Keeping the Data on Time" by John Allspaw, Jesse Robbins

Chapter 5: 'Infrastructure As Code', by Adam Jacob[1]

[1] the creator of Chef

how to make it a reality

what components do we need



- **one language** to describe infrastructure and applications, as well as all matters of operation and maintenance in code
- **an API** for every infrastructure element
- access to all kind of **monitoring data** via an API including the current status of all system and application components
- **a control system** to deploy our code which describes the infrastructure and to enforce desired state

a key prerequisite

IaaS enabled by virtualization technology

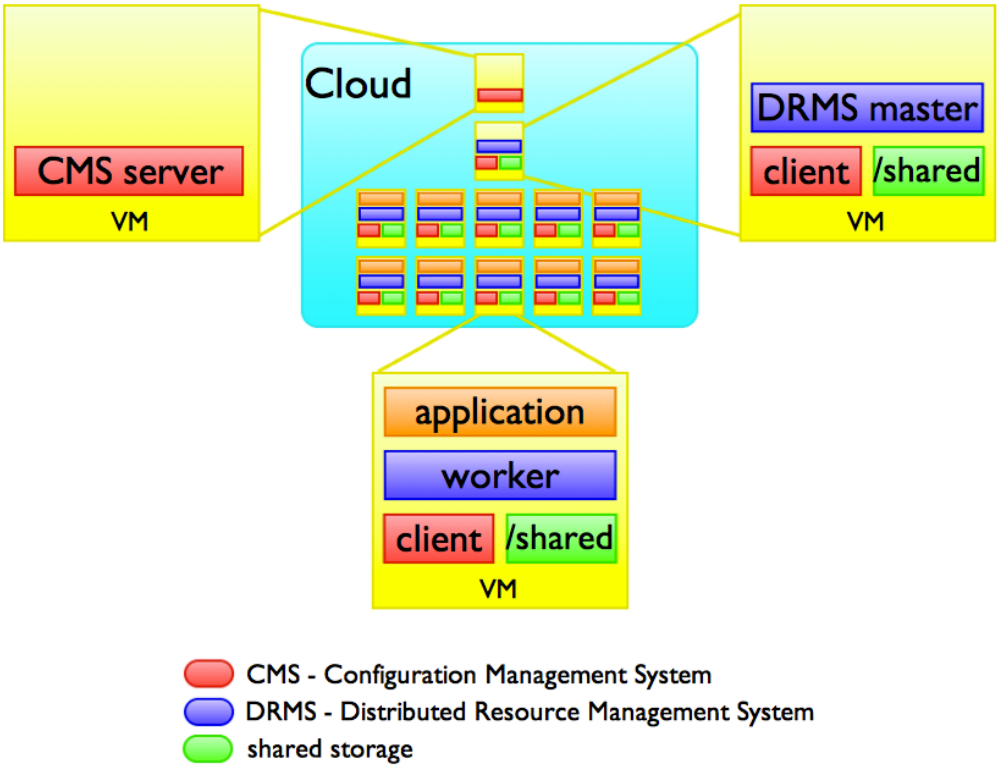
- freedom to choose a software stack, from system to application level
- very dynamic deployment with an API
- easy recycling/recovery (rebuild from scratch)
- better scalability

our approach

to deployment of a virtual computing cluster executing a HEP application

- start a VM with a control/configuration management system
- describe the infrastructure to deploy
- deploy a number of VMs needed for the application
- execute the application

virtual cluster architecture



tools we use

for virtual clusters:

- Debian, Scientific Linux 5 as operating systems for virtual machines
- Chef as configuration management and control system[1]
- Torque as a batch system
- Ganglia as a monitoring system
- our Ruby code to integrate all the components

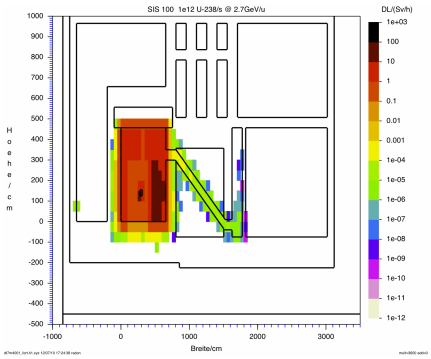
for development:

- a private IaaS-cloud at our lab
- Oracle's VirtualBox and KVM as desktop hypervisors
- Vagrant for building local virtualized development environments
- git as a version control system

[1] http://www.opscode.com/chef
[2] http://vagrantup.com

use case: FLUKA

- particle physics MonteCarlo simulation package
- CPU-bound
- trivially parallel
- low I/O, no need for shared storage
- Scientific Linux 5.5
- 1 CPU, 2GB RAM



simulation for the Safety and Radiation department at FAIR

this picture took 1067 jobs x 122 minutes ~= 2170 CPU hours or ~90 CPU days

describing an application

with the help of attributes[1]

```
"fluka": {
 "version": "2008.3c-0",
 "conf": {}
}
```

```
an example with dependencies:
"aliroot": {
  "version": "v4-20-01-AN",
  "depends": { "root": { "version": "5-26-00" } },
}
```

- defined by developer/user
- searchable in the central inventory database

```
> knife search node 'aliroot_version:v4-20-01-AN'
```

[1]examples are based on Chef

describing deployment

with the help of recipes[1]

```
name = "fluka-#{node[:fluka][:version]}.rpm"
path_to_fluka = "/tmp/#{name}"
remote_file path_to_fluka do
  source "http://.../#{name}"
  mode "0644"
  checksum "383660f3ce90412f"
end
package "Installing Fluka..." do
  action :nothing
  source path_to_fluka
  subscribes :install,
   resources(:remote_file => path_to_fluka),
   :immediately
end
```

[1]examples are based on Chef

describing a node

with the help of roles[1]

```
{
 "name": "fluka_worker_node",
 "description": "the base role for a worker node with FLUKA",
 "fluka_version": "2008.3c-0",
 "run_list": [ "role[torque_worker]","recipe[fluka]","role[ganglia_agent]"
}
```

- defined by developer/user
- searchable in the central inventory database

```
> knife node run_list add worker2.domain "role[fluka_worker_node]"
```

[1]examples are based on Chef

dynamic configuration

with the help of search functionality[1]

an example of a **Torque server** configuration:

```
workers = Array.new
search(:node, 'role:torque_worker') do |worker|
  workers << worker.name
end
log "Query index for TORQUE workers: #{workers.join(',')}"
template '/var/spool/torque/server_priv/nodes' do
  source "conf/server_priv/nodes.erb"
  mode '0644'
  variables :workers => workers
  notifies :restart, resources(:service => 'pbs_server')
end
```

[1]examples are based on Chef

dynamic configuration

with the help of search functionality[1]

an example of a **Torque worker** configuration:

```
server_name = search(:node, 'role:torque_server')[0].name
log "Query index for TORQUE master: #{server_name}"
# deploy the configuration files defining the TORQUE server
template '/var/spool/torque/server_name' do
  source 'conf/server_name.erb'
  mode '0644'
  variables :server => server_name
  notifies :restart, resources(:service => "pbs_mom")
end
```

[1]examples are based on Chef

accessing monitoring data

from configuration code[1]

```
{
  "domain": "gsi.de",
  "os": "linux",
  "idletime": "55 days 11 hours 55 minutes 22 seconds",
  "hostname": "lxbox1",
  "platform": "debian"
  "kernel" : {
     "release" : "2.6.26-2-amd64"
  }
}
```

central information hub reflects the current state of the infrastructure

```
search(:node, "hostname:lx* AND platform:debian").each do
   # some code to execute
end
```

[1]examples are based on Chef

flexible deployment

through interaction with a network-accessible API[1]

```
# apply a role to a torque server node
system('knife node run_list add torque_server.domain "role[fluka_torque_se

# apply a role to torque worker nodes
data = `knife search node "name:worker*" -i`
data.scan(/^\S+$/).each do |k|
    `knife node run_list add #{k} "role[fluka_worker_node]"`
end

# run chef clients on all fluka nodes to apply the configuration
system('knife ssh "role:fluka*" "sudo chef-client"')
```

[1]examples are based on Chef and Ruby code

flexible operation

through interaction with a network-accessible API[1]

```
#List all nodes
 knife node list
#Search for nodes that have applied recipe X, put out the fqdn
 knife search node 'run_list:"recipe[fluka]"' -a fqdn
#Show node status and applied roles & recipes for Torque workers
 knife status "role:torque_worker" --run-list
#Run 'uptime' on all Torque workers
 knife ssh "role:torque_worker" "uptime"

#Delete nodes who had not put in an appearance since 12:45 on 24/03/2011
 t = Time.local(2011,03,24,12,45,00).to_i
 data = `knife search node 'ohai_time:[0 TO #{t}]' -i`
 data.scan(/^\S+$/).each do |k|
   `knife node delete #{k} -y`
 end
```

[1]examples are based on Chef and Ruby code

sharing code globally

like at http://community.opscode.com/cookbooks

All Categories

See all Cookbooks →

mysql ★★★★★ 2168 downloads
nginx ★★★★★ 1196 downloads
apache2 ★★★★★ 933 downloads

Databases	mysql ★★★★★ 2168 downloads postgresql ★★★★★ 362 downloads couchdb ★★★★★ 179 downloads See all →	**Web Servers**	nginx ★★★★★ 1196 downloads apache2 ★★★★★ 933 downloads passenger_apache2 ★★★★☆ 164 downloads See all →
Process Management	runit ★★★★★ 131 downloads god ★★★★★ 101 downloads bluepill ★★★★★ 21 downloads See all →	**Monitoring & Trending**	nagios ★★★★★ 452 downloads munin ★★★★★ 130 downloads monit ★★★★★ 89 downloads See all →
Programming Languages	java ★★★★★ 464 downloads php ★★★★★ 218 downloads ruby ★★★★★ 174 downloads See all →	**Package Management**	apt ★★★★★ 293 downloads bootstrap ★★★☆☆ 93 downloads packages ★★★★★ 85 downloads See all →
Applications	tomcat6 ★★☆☆☆ 416 downloads postfix ★★★★★ 123 downloads wordpress ★★★★★ 112 downloads See all →	**Networking**	iptables ★★☆☆☆ 193 downloads haproxy ★★★★★ 126 downloads ntp ★★★★★ 121 downloads See all →
Operating Systems & Virtualization	ubuntu ★★★★★ 221 downloads ec2 ★★★★☆ 154 downloads aws ★★★★☆ 111 downloads See all →	**Utilities**	chef ★★★★☆ 281 downloads git ★★★★★ 268 downloads openssl ★★★★★ 152 downloads See all →

+ Add a new Cookbook

Need Help?

If you have questions, or you're stuck, we're here to help.

Get Help ▸

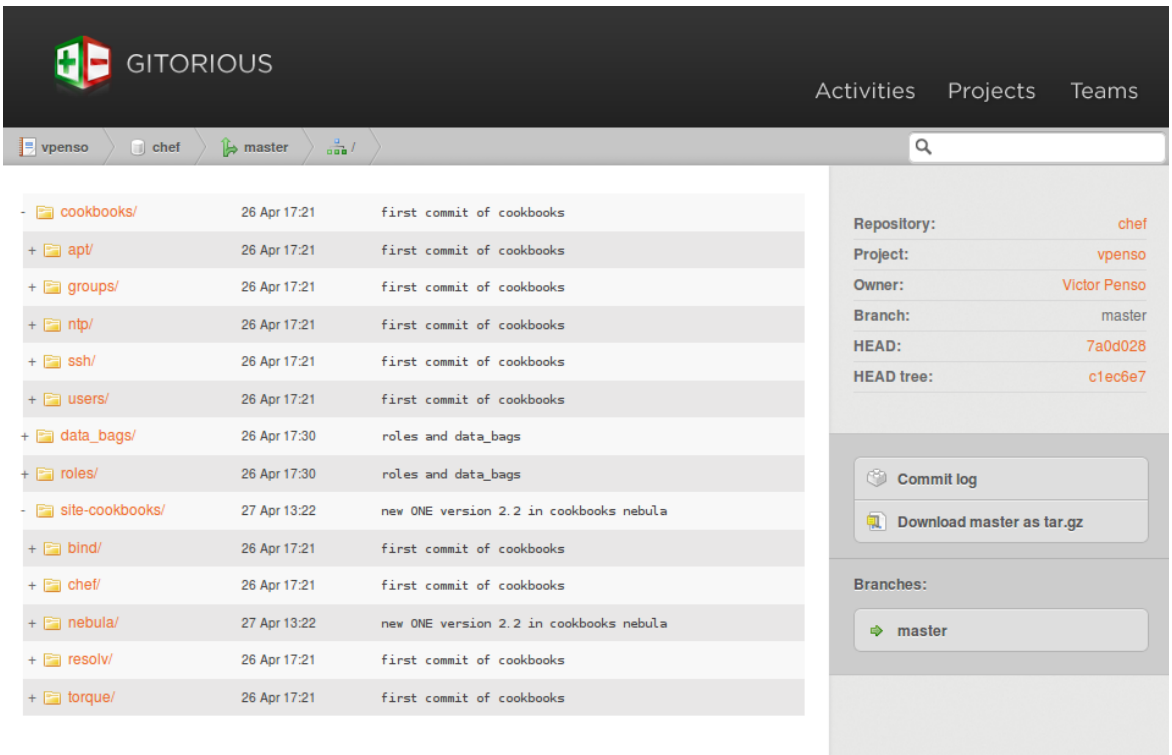
REST API

Access all cookbooks on this site programatically.

Learn More ▸

sharing code locally at site

like with Gitorious[1]



[1] http://gitorious.org

our playgrounds

private and public clouds

- Amazon EC2
- a private prototype installation of OpenNebula at GSI called SCLab
- a community cloud in Frankfurt[1]
- test installations of Nimbus & Eucalyptus have been used for evaluation[2]

[1] http://www.frankfurt-cloud.com
[2] we are interested in OpenStack too

SCLab

a private prototype IaaS-facility at our lab[1]

- dedicated VLAN and subnet
- Debian Lenny/Squeeze as host OS with...
- ...KVM as a virtual machine hypervisor;
- OpenNebula toolkit for building an IaaS-cloud using...
- ...libvirt (virtualization API) as an abstraction above a hypervisor
- 16 physical boxes, >100 VMs in parallel
- ~15 users touched it, ~5 working with it intensively

[1]we have integrated this prototype into the existing IT infrastructure as far as it was possible, but development of an IaaS facility for GSI is not our primary task

results & outreach

practical results[1]:

- an AliEn grid site for the ALICE experiment at LHC
- FLUKA simulation for the Safety and Radiation department at FAIR
- state-of-the-art nuclear structure calculations with energy density functional methods
- a testbed for transition to virtual environment for the HPC department at GSI

outreach:

- GSI internal tutorial series by Victor Penso & Bastian Neuburger
- a poster at the USENIX LISA 2010
- a presentation at ISGC 2011 by Dr. Peter Malzacher

[1]all documented in the GSI Scientific Report 2010 http://www.gsi.de/informationen/wti/library/index_e.html

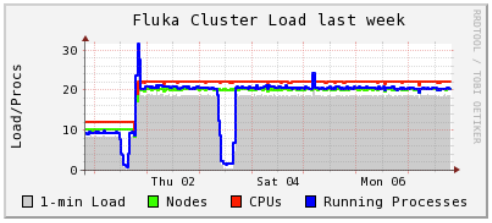
FLUKA @ Frankfurt Cloud

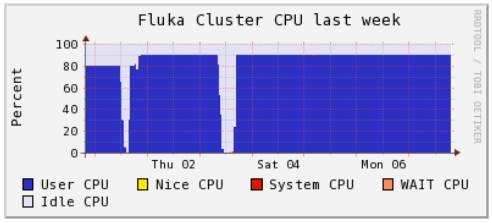
Overview of Fluka

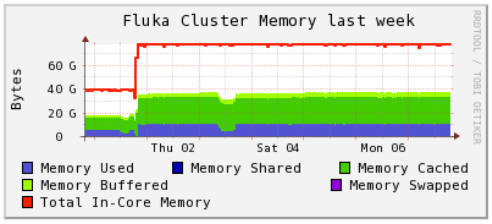
CPUs Total: **22**
Hosts up: **20**
Hosts down: **0**

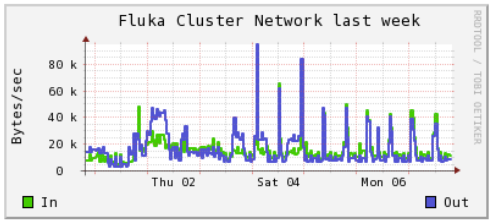
Avg Load (15, 5, 1m): **83%, 87%, 89%**
Localtime: **2010-12-07 19:14**

Pie Chart

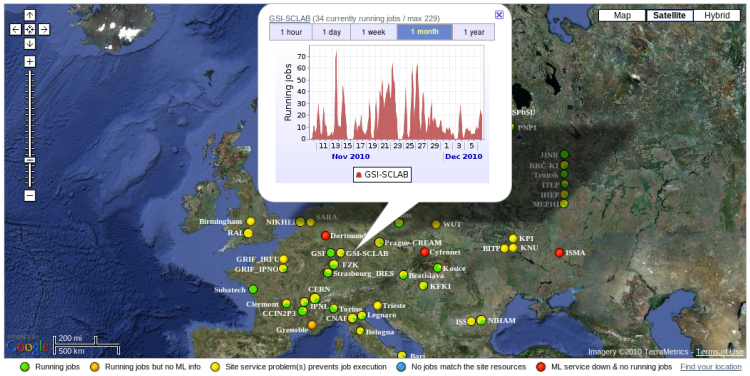








a virtual grid site for ALICE @ SCLab



outlook

hopefully in the near future

- open cloud APIs (OCCI, EC2, DeltaCloud) on our facility
- virtual nodes integrated to GSI batch system
- operation of several AliEn Grid sites on different clouds
- commissioning an IaaS-cloud facility at the TOP500 supercomputer in Frankfurt[1]
- more users, more applications

[1] http://compeng.uni-frankfurt.de/index.php?id=86