Towards *real* Cloud automation

Roberto Di Cosmo
(for the Aeolus project)

July 11th 2013
The Aeolus project

ANR funded: 4 years, ends on 12/14.
The Aeolus project

ANR funded: 4 years, ends on 12/14.

Aeolus team

- University Paris Diderot (coordinator, language design and dependencies),
- University Sophia Antipolis (solvers),
- Inria Focus project (distributed systems),
- Mandriva
The Aeolus project

ANR funded: 4 years, ends on 12/14.

Aeolus team

- University Paris Diderot (coordinator, language design and dependencies),
- University Sophia Antipolis (solvers),
- Inria Focus project (distributed systems),
- Mandriva

Aeolus’ goal

to make a quantum leap toward automatic reconfiguration in the cloud
An Ideal Cloud Application

The promise:
- automatic deployment
- automatic (re)configuration
- automatic scaling

The reality:
Long, detailed, manual descriptions of the full system are mandatory.

Aeolus plans to help fill the gap.

R. Di Cosmo (CNRS/INRIA/MDV/UPD)
An Ideal Cloud Application

The promise:
- automatic deployment
- automatic (re)configuration
- automatic scaling
An Ideal Cloud Application

The promise:
- automatic deployment
- automatic (re)configuration
- automatic scaling

The reality:
Long, detailed, manual descriptions of the full system are mandatory
An Ideal Cloud Application

The promise:
- automatic deployment
- automatic (re)configuration
- automatic scaling

The reality:
Long, detailed, manual descriptions of the full system are mandatory

Aeolus plans to help fill the gap
Aeolus Focus

Key ingredients:
a model of a cloud system,
a high level description language for expressing reconfiguration requests,
a low-level vendor-independent deployment description language,
advanced, specialised algorithms for optimised platform deployment and (re)configuration.

R. Di Cosmo (CNRS/INRIA/MDV/UPD)
Key ingredients:

- a model of a cloud system,
- a high level description language for expressing reconfiguration requests,
- a low-level vendor-independent deployment description language,
- advanced, specialised algorithms for optimised platform deployment and (re)configuration.
A fragment of a realistic configuration for a WordPress web service.
A fragment of a realistic configuration for a WordPress web service.

Simple, but expressive model: functional requirements and provides, nonfunctional requirements via capacity constraints.
Overview of the approach

Long term goal

Provide a full \textit{reconfiguration} plan from an initial configuration to a final configuration of a distributed application in the cloud.

Limiting results

A simple model with conflicts and no capacity constraints has a decidable reachability problem which is EXPSPACE-complete.

The full model with conflicts and capacity constraints has an undecidable reachability problem.

That's why existing tools for automatic reconfiguration are so limited!

Current results

We are able to compute automatically a final configuration from a concise user specification, and various nonfunctional constraints (see Jakub's talk).
Overview of the approach

Long term goal
Provide a full *reconfiguration* plan from an initial configuration to a final configuration of a distributed application in the cloud

Limiting results

*a simple model* with conflicts and no capacity constraints has a *decidable* reachability problem which is EXPSPACE-complete

the *full model* with conflicts and capacity constraints has an *undecidable* reachability problem

That’s why existing tools for automatic reconfiguration are so limited!
Overview of the approach

Long term goal
Provide a full reconfiguration plan from an initial configuration to a final configuration of a distributed application in the cloud

Limiting results

a simple model with conflicts and no capacity constraints has a decidable reachability problem which is EXPSPACE-complete

the full model with conflicts and capacity constraints has an undecidable reachability problem

That’s why existing tools for automatic reconfiguration are so limited!

Current results

We are able to compute automatically a final configuration from a concise user specification, and various nonfunctional constraints (see Jakub’s talk)
Computing a final configuration: a simple use case

Your goal

Deploy a load balanced \texttt{wordpress} farm according to house policy and using the minimum number of virtual machines sporting 2Gb of ram
Computing a final configuration: a simple use case

Your goal

Deploy a load balanced \texttt{wordpress} farm according to house policy and using the minimum number of virtual machines sporting 2Gb of ram

Available components

- **load balancer**: can be DNS based (but then only one DNS must be present), or Varnish based
- **wordpress**: needs a MySQL back-end

House policy

- DNS load balancer uses only 256Mb, but requires at least 7 WP backends
- Varnish requires only 3 WP backends, but uses up 2Gb
- \texttt{wordpress} needs 2 MySQL back-ends, eats 512Mb, max 1 per VM
- MySQL needs 512Mb, max 1 per VM, max 3 WP connections
Computing a final configuration: a simple use case

**Your goal**
Deploy a load balanced `wordpress` farm according to house policy and using the minimum number of virtual machines sporting 2Gb of ram

**Available components**
- **load balancer** can be DNS based (but then only one DNS must be present), or Varnish based
- **wordpress** needs a MySQL back-end

**House policy**
- **DNS load balancer** uses only 256Mb, but requires at least 7 WP backends
- **varnish** requires only 3 WP backends, but uses up 2Gb
- **wordpress** needs 2 MySQL back-ends, eats 512Mb, max 1 per VM
- **MySQL** needs 512Mb, max 1 per VM, max 3 WP connections
That’s not so easy

Sure, Cfengine, Puppet, Chef, etc. will be needed...
That’s not so easy

Sure, Cfengine, Puppet, Chef, etc. will be needed...at the end, but first you need to come up with the architectural description of your deployment!
That’s not so easy

Sure, Cfengine, Puppet, Chef, etc. will be needed... at the end, but first you need to come up with the architectural description of your deployment!

Remark

We want to be sure that when you decide to place $n$ services on the same VM, they will actually be deployable together. Puppet and friends cannot ensure this, as they know nothing of the package dependencies and conflicts underneath!
That’s not so easy

Sure, Cfengine, Puppet, Chef, etc. will be needed...at the end, but first you need to come up with the architectural description of your deployment!

Remark

We want to be sure that when you decide to place \( n \) services on the same VM, they will actually be *deployable* together. Puppet and friends cannot ensure this, as they know nothing of the package dependencies and conflicts underneath!
That’s not so easy

Sure, Cfengine, Puppet, Chef, etc. will be needed...at the end, but first you need to come up with the architectural description of your deployment!

Remark

We want to be sure that when you decide to place $n$ services on the same VM, they will actually be deployable together. Puppet and friends cannot ensure this, as they know nothing of the package dependencies and conflicts underneath! We do.

And what would you do if

you are asked to minimize the disk image footprint of the VMs you will deploy?
Zephyrus to the rescue

Zephyrus

is a tool from the Aeolus suite that computes an optimal final configuration out of all the pieces of information seen above.
Zephyrus to the rescue

Zephyrus

is a tool from the Aeolus suite that computes an optimal final configuration out of all the pieces of information seen above.

Zephyrus explained

See Jakub’s talk in a few minutes
Learn more, get involved

