Beyond deployment - management and reconfiguration

Advanced distributed systems

Hélène Coullon

Associate professor at IMT Atlantique, France

Inria researcher, France

Adjunct professor at UiT, Tromsø, Norway

March 2nd, 2021

Outline

Let's start with a question on Wooclap

WEB

1. Connect to www.wooclap.com/DQATAR
2. You can participate

SMS

1. Not yet connected? Send @DQATAR to 06 44 60 96 62
2. You can participate

Table of Contents

Deployment, what's next?

- Ever-running and long-running distributed systems
- Management/reconfiguration through time
  - Need to add/remove modules and/or connections
  - Need to change internal configurations

Examples of management reasons

- Faults or errors on services or hardware,
- dynamic energy or security constraints,
- dynamic improvement of performance,
- dynamic upgrade of some modules.

Reconfiguration

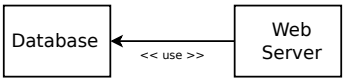
What is a reconfiguration?

- A set of instructions to move from one state of the system to another,
- transient state of the system,
- the application of each instruction leads to a new transient state.

What is needed to reconfigure?

- Need to know the state of the system,
- need for a **reconfiguration language** with reconfiguration operations,
  - change the set of components,
  - change the set of connections,
  - change the configuration of components,
- need for a **well-defined execution semantics** of the language
  - safety
  - automation.

Example - Update a database in-place



Database (DB) [WHAT]

[HOW] [LIFECYCLE]

1. Backup data
2. Stop the service
3. Download update
4. Configure parameters
5. Start the service
6. Restore data

Web-server (WS) [WHAT]

[HOW] [LIFECYCLE]

1. Pause the service
2. Configure parameters
3. Start the service

[DEPENDENCIES]: DB(2) $\rightarrow$ WS(1), WS(2) $\rightarrow$ DB(5), WS(3) $\rightarrow$ DB(6) (lifecycle granularity)

Which alternative? (immutability)

WEB

1. Connect to www.wooclap.com/DQATAR
2. You can participate

SMS

1. Not yet connected? Send @DQATAR to 06 44 60 96 62
2. You can participate

Table of Contents

Overview of reconfiguration tools

	Config Management	Docker ecosystem	Provisioning	Orchestration
Ansible (2012)	✓		(✓)	
Puppet (2005)	✓		(✓)	
Docker Compose (2014)		✓		
Docker Swarm (2014)		✓		✓
Kubernetes (2014)		✓		✓
Nomad (v1.0)				✓
Terraform (2014)	(✓)		✓	
Juju (2011)	✓		✓	
Cloud Formation (2006)			✓	
HOT and Heat (2010)			✓	✓
Tosca ecosystem (2014)	✓	(✓)	✓	✓

Configuration management

Designed to install, configure, manage software on existing servers.

Reconfiguration with CM tools

- Procedural approach (e.g., Ansible): manually written
- Declarative approach (e.g., Puppet): automatic with `diff`, complex cases handled manually.

Ansible example

- Deploy 3 instances of a role,
- deploy 4 more to get 7.

Puppet example

- Specify 3 instances of a resource,
- modify this specification to 7, adding 4 is done automatically.

Provisioning

Initially designed to install, configure, manage software on existing servers.

Reconfiguration with provisioning tools

- Declarative approach (e.g., TERRAFORM): automatic with `diff`, limited to provisioning.

TERRAFORM example

- Specify 3 instances of a instances to provision,
- modify this specification to 7, adding 4 is done automatically.

Docker ecosystem

Solve portability problem and reduce configuration issues through containers.

Reconfiguration with Docker ecosystem

- Immutable approach handled manually,
  - creating and starting a new container from another Docker image,
  - stop the old container.
- Automated management of scalability or faults

Kubernetes and Docker Swarm

- automatic management of containers replicas,
  - faults
  - load balancing
- automated service discovery.

Orchestration tools

Examples of orchestrators

KUBERNETES, DOCKER SWARM, NOMAD, HEAT project of OPENSTACK etc.

Orchestration tools

- **[HOW]** is abstracted and left to containers, VMs or CM tools,
- **[WHAT]** is abstracted as a container or set of containers (e.g., pods in KUBERNETES).

Manage a set of containers of a set of applications that share a set of resources (virtual or physical), and often come with a more complex software stack, with a scheduler for instance.

Reconfiguration with orchestrators

Immutability (easy rollback), scalability and fault tolerance.

Engage

Reconfiguration is seen more or less as a new deployment

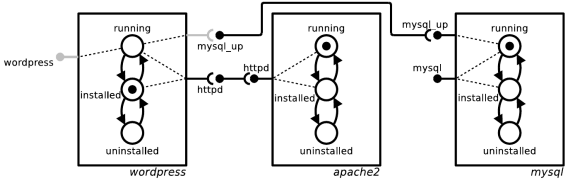
1. backup of the current system
2. components to remove are uninstalled
3. the new system is deployed

In-place vs immutable

"*all upgrades using this approach experience the worst case upgrade time, even if there are only minor differences between the old and new configurations*"

Aeolus

- programmable lifecycle
- deployment/reconfiguration language




```
1  new(z1 : wordpress)
2  new(z2 : apache2)
3  stateChange(z2 , uninstalled , installed)
4  bind(httpd, z 1 , z 2 )
5  stateChange(z1 , uninstalled , installed)
6  new(z3 : mysql)
7  stateChange(z3 , uninstalled , installed)
8  stateChange(z3 , installed , running)
9  bind(mysql_up, z 1 , z3 )
10 stateChange(z2 , installed , running)
```

Is deployment a specific reconfiguration?

WEB

1. Connect to www.wooclap.com/DQATAR
2. You can participate

SMS

1. Not yet connected? Send @DQATAR to 06 44 60 96 62
2. You can participate

Table of Contents

Small quiz on parallelism

WEB

1. Connect to www.wooclap.com/DQATAR
2. You can participate

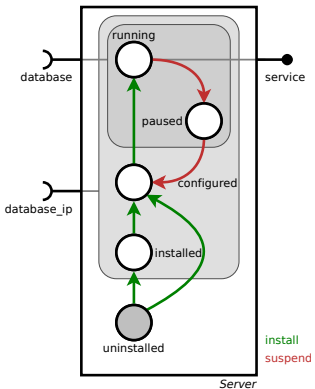
SMS

1. Not yet connected? Send @DQATAR to 06 44 60 96 62
2. You can participate

Table of Contents

Control components

Written by the component developers

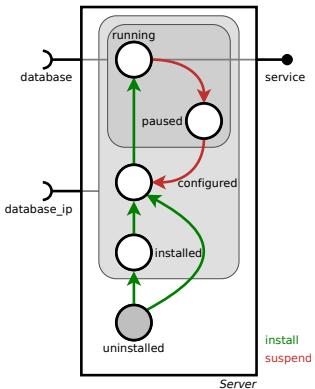


Internal net **[LIFECYCLE]**

- places = milestones
- transitions = actions to perform
  - concretely: scripts are attached to transitions
  - in the model: exact nature/effects of actions not represented, only coordination

Control components

Written by the component developers



Interfaces [DEPENDENCIES]

- data or service ports
  - use ports = requirements
  - provide ports = provisions
  - during execution: active/inactive
- behaviors
  - subset of transitions
  - during execution: active/inactive

Control components in practice

Written by the component developers

```
class Server(Component):
    def create(self):
        self.places = ['uninstalled','installed','configured','running','paused']

        self.initial_place = 'uninstalled'

        self.behaviors = ['b_install','b_suspend']

        self.transitions = {
            'install1': ('uninstalled','installed','b_install',self.install1),
            'install2': ('uninstalled','configured','b_install',self.install2),
            'configure': ('installed','configured','b_install',self.configure),
            'start': ('configured','running','b_install',self.start),
            'suspend1': ('running','paused','b_suspend',self.suspend1),
            'suspend2': ('paused','configured','b_suspend', self.suspend2)
        }
```

Control components in practice

Written by the component developers

```
class Server(Component):
    def create(self):
        ...

        self.dependencies = {
            'database_ip': (DepType.USE, ['installed','configured','running','paused']),
            'database': (DepType.USE, ['running','paused']),
            'service': (DepType.PROVIDE, ['running'])
        }

    # Definition of the actions
    def install1(self):
        remote = SSHClient()
        remote.connect(host, user, pwd)
        remote.exec_command(cmd)
        ...
```

Reconfiguration language

Add/remove

Add/remove a component instance to the current assembly

Connect/disconnect

Connect/disconnect two component instances with compatible ports

Push behavior

Push a behavior to the behavior queue on a component instance

Wait

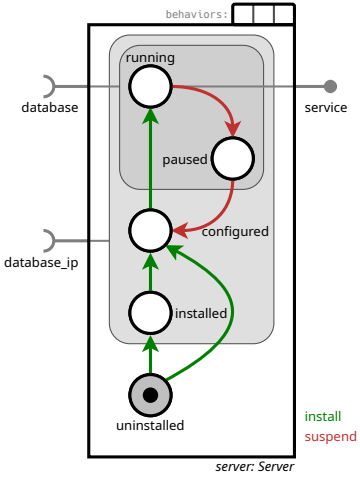
Wait for a given behavior of a component instance

Reconfiguration example - deployment

Written by the reconfiguration developer

Deployment program:

```
1 add(server: Server)
2 add(db: Database)
3 con(server.database_ip,db.ip)
4 con(server.database,db.service)
5 pushB(server, install)
6 pushB(db, deploy)
7 wait(server)
```

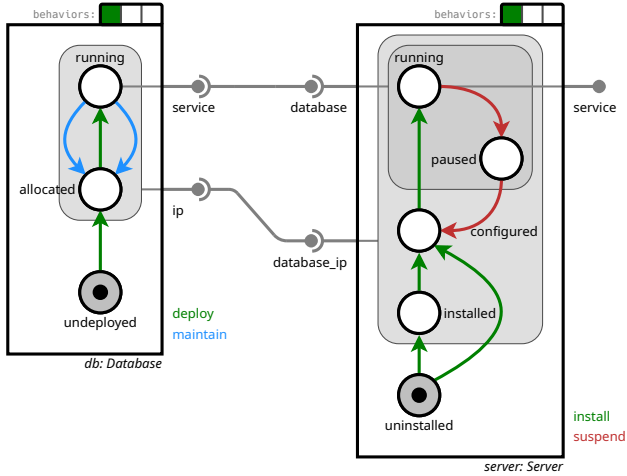


Reconfiguration example - deployment

Written by the reconfiguration developer

Deployment program:

```
1 add(server: Server)
2 add(db: Database)
3 con(server.database_ip,db.ip)
4 con(server.database,db.service)
5 pushB(server, install)
6 pushB(db, deploy)
7 wait(server)
```

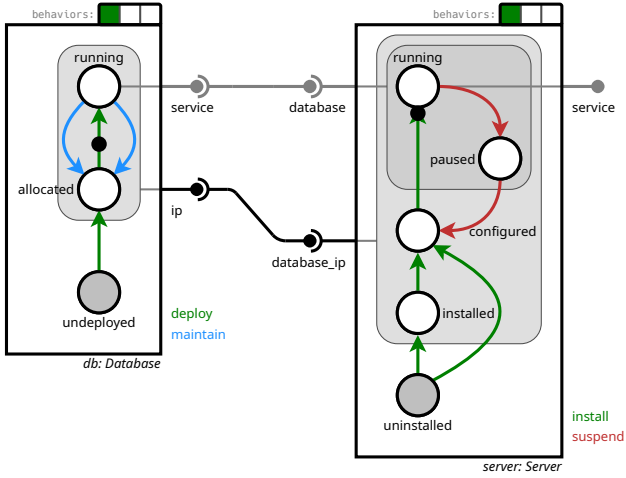


Reconfiguration example - deployment

Written by the reconfiguration developer

Deployment program:

```
1 add(server: Server)
2 add(db: Database)
3 con(server.database_ip,db.ip)
4 con(server.database,db.service)
5 pushB(server, install)
6 pushB(db, deploy)
7 wait(server)
```

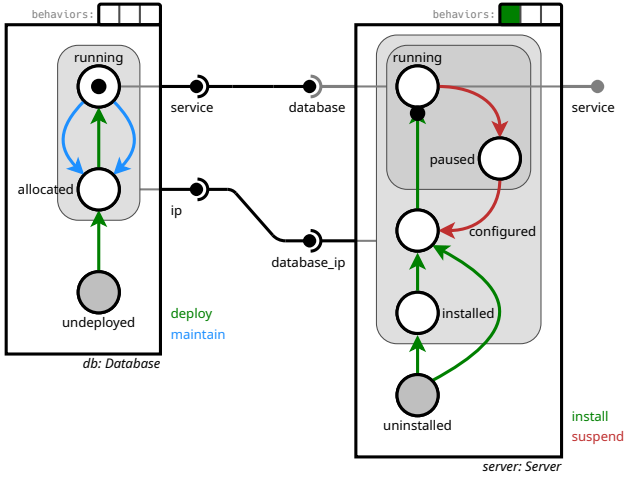


Reconfiguration example - deployment

Written by the reconfiguration developer

Deployment program:

```
1 add(server: Server)
2 add(db: Database)
3 con(server.database_ip,db.ip)
4 con(server.database,db.service)
5 pushB(server, install)
6 pushB(db, deploy)
7 wait(server)
```

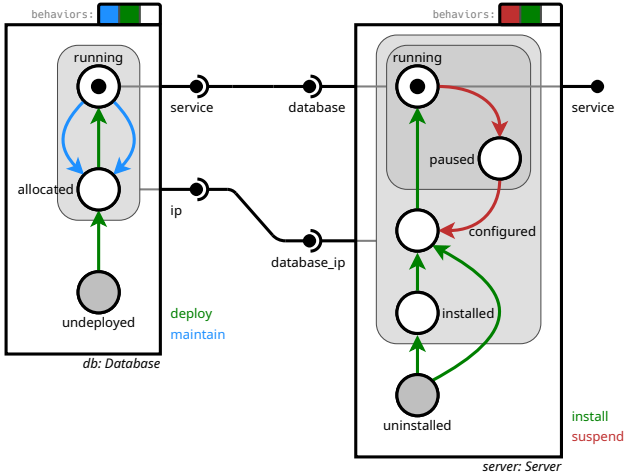


Reconfiguration example - maintenance

Written by the reconfiguration developer

Maintenance program:

```
1 pushB(db, maintain)
2 pushB(db, deploy)
3 pushB(server, suspend)
4 pushB(server, install)
5 wait(server)
```

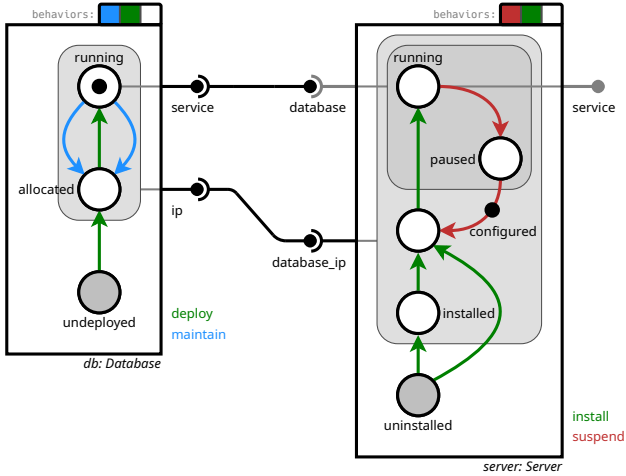


Reconfiguration example - maintenance

Written by the reconfiguration developer

Maintenance program:

```
1 pushB(db, maintain)
2 pushB(db, deploy)
3 pushB(server, suspend)
4 pushB(server, install)
5 wait(server)
```

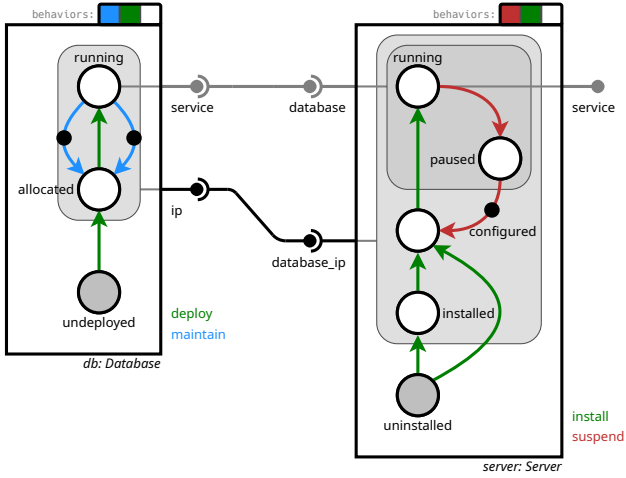


Reconfiguration example - maintenance

Written by the reconfiguration developer

Maintenance program:

```
1 pushB(db, maintain)
2 pushB(db, deploy)
3 pushB(server, suspend)
4 pushB(server, install)
5 wait(server)
```



Reconfiguration language in practice

Written by the reconfiguration developer

```
class ServerClient(Assembly):
    def __init__(self):
        self.server = Server()
        self.database = Database()
        Assembly.__init__(self)

    def deploy(self):
        self.add_component('database', self.database)
        self.add_component('server', self.server)
        self.connect('server', 'database_ip', 'database', 'ip')
        self.connect('server', 'database', 'database', 'service')
        self.push_b('server', 'install')
        self.push_b('database', 'deploy')
        self.wait_all()
```

Reconfiguration language in practice

Written by the reconfiguration developer

```
class ServerClient(Assembly):
    ...

    def maintain(self):
        self.push_b('database', 'maintain')
        self.push_b('database', 'deploy')
        self.push_b('server', 'suspend')
        self.push_b('server', 'install')
        self.wait_all()
```

Separation of concerns and parallelism

Written by the reconfiguration developer

AEOLUS manipulation of internal states

```
1 ...
2 stateChange(z3 , uninstalled , installed)
3 stateChange(z3 , installed , running)
4 bind(mysql_up, z 1 , z3 )
5 stateChange(z2 , installed , running)
```

CONCERTO manipulation of behaviors

```
1 class ServerClient(Assembly):
2     ...
3     def maintain(self):
4         self.push_b('database', 'maintain')
5         self.push_b('database', 'deploy')
6         self.push_b('server', 'suspend')
7         self.push_b('server', 'install')
8         self.wait_all()
```

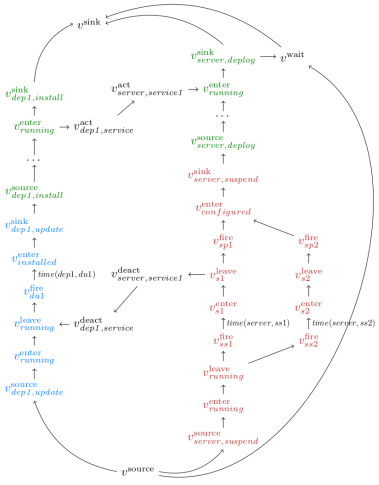
Performance prediction

Inputs:

- reconfiguration program
- time estimations for transitions

Dependency graph generation

- nodes for events such as reaching/leaving place, firing transition
- transition arcs are weighted to reflect execution time
- other arcs are 0-weighted



Performance prediction

Critical path

- length = reconfiguration time (assuming hardware can execute as many concurrent threads as needed)
- highlights the transitions that should be optimized

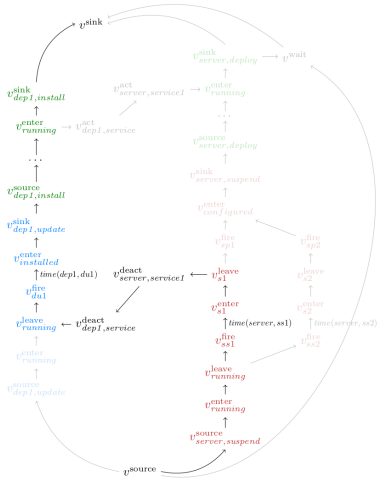


Table of Contents

Evaluation on the reconfiguration of MariaDB

Real use-case extracted from the OpenStack Summit 2018 on a multi-region deployment of OpenSatck

Initial state

- centralized MariaDB running
- additional nodes running some generic components (docker, pip...)

decentralization

- replaces centralized DB with a distributed version with instances on n nodes
- requires a backup of the data, and a restart of the master node

scaling

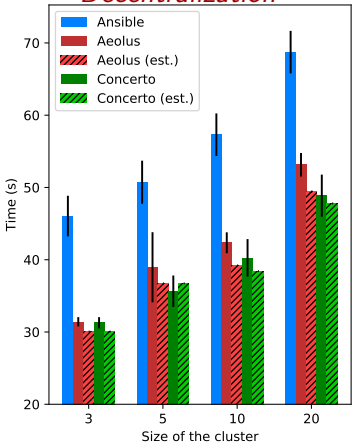
- deploys m new nodes with an instance of the distributed DB

Reproducible experiments

Evaluation on the reconfiguration of MariaDB

Results on nodes of UvB (Sophia) of Grid'5000 (2×6-core Intel Xeon X5670 CPUs, 96 GB RAM, 250 GB HDD, internal 40 Gbps InfiniBand, external 1 Gbps).

Decentralization



Scaling

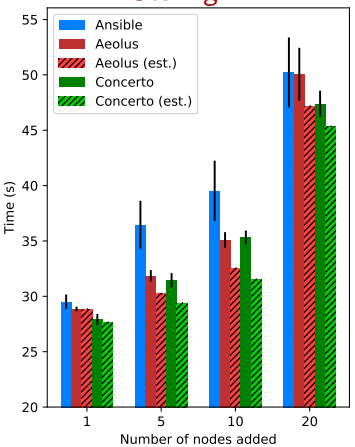


Table of Contents

Conclusion

- Reconfiguration problem and its complexity
- Need for software engineering practices
- Overview of DevOps tools and academic contributions
- Presentation of Concerto
- Evaluation of Concerto

Next talk (April 19)

- CONCERTO is a model and a language to write any reconfiguration program.
- Who writes the reconfiguration program?
- Do we want to write such program or do we want it to be automatically generated?

Autonomic reconfiguration/management

- autonomic decision of a new state to reach,
- autonomic decision of the reconfiguration needed to do it.

Questions?