DevOps deployment tools - Behind the scene

Advanced distributed systems

Hélène Coullon

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Associate professor at IMT Atlantique, France

Inria
INVENTEURS DU MONDE NUMÉRIQUE

Inria researcher, France

NORGES ARKTISKE UNIVERSITET • UIT •

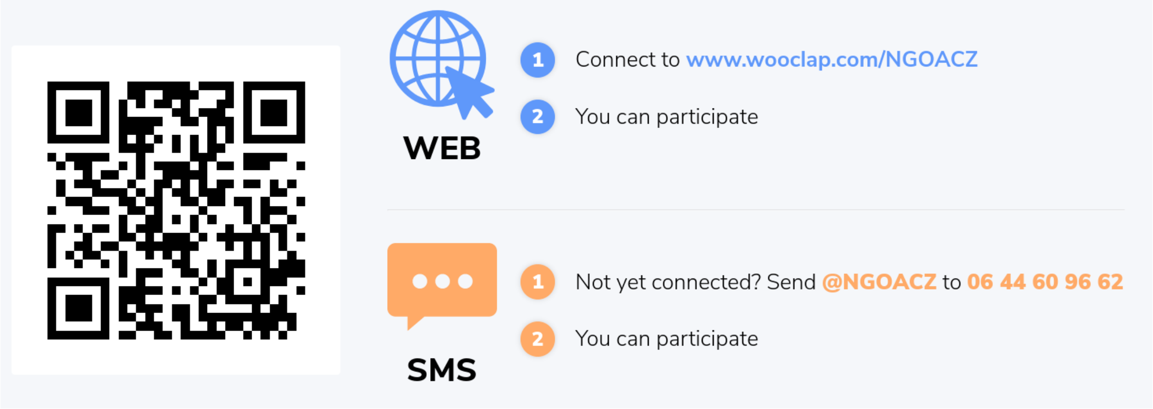
Adjunct professor at UiT, Tromsø, Norway

March 1st, 2021

Outline

1. Introduction
2. Overview of deployment tools
3. Parallelism within deployments
4. The MADEUS deployment model
5. Evaluation
6. Conclusion

Let's start with a few questions on Wooclap



WEB

1 Connect to www.wooclap.com/NGOACZ

2 You can participate

SMS

1 Not yet connected? Send @NGOACZ to 06 44 60 96 62

2 You can participate

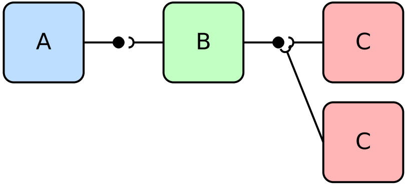
Table of Contents

1. Introduction
2. Overview of deployment tools
3. Parallelism within deployments
4. The Madeus deployment model
5. Evaluation
6. Conclusion

Distributed software systems

General definition

- Non monolithic code,
- modular units of code - **components**,
  - black-box of code,
  - with well-defined provided and required interfaces,
- software system = **architectural assembly** of component instances,
- interactions between components through **communications**.



- Master/workers,
- peer-to-peer,
- dataflow/stream,
- service-oriented,
- layered.

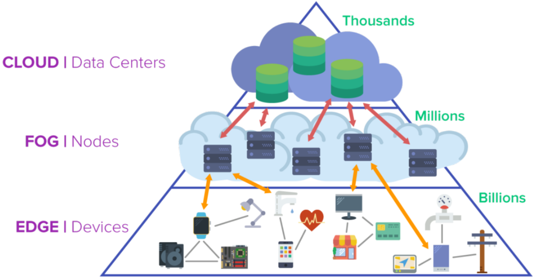
Distributed software systems everywhere

Examples of distributed software systems

- Smartphone applications (e.g., Waze),
- health, bank, tax information systems,
- Netflix micro-services infrastructure (i.e., Netflix OSS),
- operating system of the Cloud (e.g., OpenStack),
- 5G networks (e.g., network function virtualization).

Distributed infrastructures

Computing, storage and network resources everywhere!



- Cloud computing
- Fog and Edge computing
- Internet-of-Things (IoT)
- Cyber-physical systems

Deployment

What is a deployment?

Install, configure, start, test a distributed software system.

Questions raised by deployment

- What do I need to deploy?
- Where do I need to deploy?
- How do I deploy?
- When do I deploy?

[WHAT + HOW + WHERE + WHEN] = DevOps deployment tools

Example - deploying LAMP

[WHAT] LAMP

- **L**inux operating system
- **A**pache web server
- **M**ariaDB database
- **P**HP language

[WHERE]

on **2 nodes**: `machine1`, `machine2`

Example - deploying LAMP

[HOW]

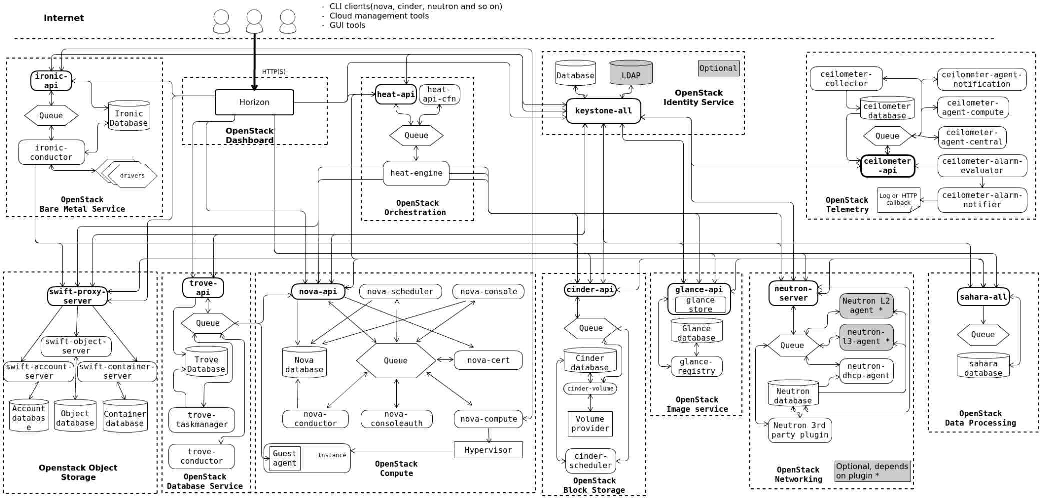
- Let's take a look at a basic documentation
- What if I want to configure Apache and MariaDB?
- What if I deploy on CentOS?

[WHEN]

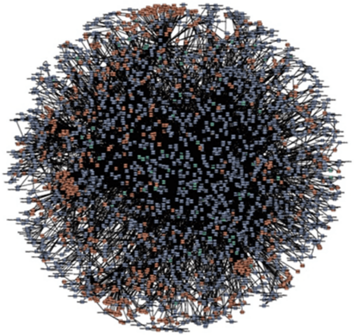
```
linux → apache → mariadb → php
linux → mariadb → apache → php
linux → apache → php → mariadb
```

Not so simple?

What about this?



Or this?



amazon.com

NETFLIX

On this?

Deployment tools and DevOps community

documentation or `README.md` ⟶ ad-hoc scripts ⟶ deployment tools

Software engineering practices applied to deployment

- Automating deployments at scale,
- structuring deployments (languages, models),
- reusing deployment procedures,
- avoiding errors.

Table of Contents

1. Introduction
2. Overview of deployment tools
3. Parallelism within deployments
4. The Madeus deployment model
5. Evaluation
6. Conclusion

Overview of deployment 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)	✓	(✓)	✓	✓

Archiecture of these tools

Agentless, master/workers and heavier software stack (i.e., bootstrap problem).

In practice - combination of tools

4 categories

Configuration management tools

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

Provisioning tools

Initially designed to provision the servers, network, platforms etc.

Docker ecosystem

Solve portability problem and reduce configuration issues through containers.

Orchestration tools

Automated coordination, and management of a set of components that form distributed software systems, on a set of resources (virtual or physical).

Decoupling WHAT / HOW / WHERE / WHEN

Machine 1 **[WHERE]**

Database (DB) **[WHAT]**

[HOW]

1. install prerequisite 1
2. install prerequisite 2
3. install MySQL
4. configure parameters
5. start the service
6. setup the root user
7. add a user
8. create table

Machine 2 **[WHERE]**

Web-server (WS) **[WHAT]**

[HOW]

1. install prerequisite 1
2. install prerequisite 2
3. install Apache
4. configure the firewall
5. restart the firewall
6. download the website content
7. untar the website content
8. configure parameters
9. start the service

[WHEN]: DB $\rightarrow$ WS (components granularity)

Academic contributions

Enhancing **[WHEN]**

- enhancing **[WHEN]** by decoupling
  - **[LIFECYCLE]**
  - **[DEPENDENCIES]**

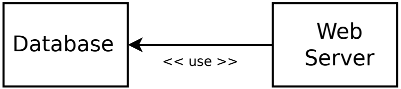
Contributions

Tosca, Deployware, SmartFrog, Engage, Aeolus, Madeus etc.

SmartFrog (1996-2003-2009) → Engage (2012) → Aeolus (2013-2016) → Madeus (2018-2021)

- comparable but complementary to configuration management tools
- generic to any kind of resource and action (configuration, provisioning, management)

Decoupling WHAT / HOW / WHERE / WHEN



Database

<< use >>

Web Server

Machine 1 **[WHERE]**

Database (DB) **[WHAT]**

[HOW] **[LIFECYCLE]**

1. Install
2. Configure
3. Start the service
4. Prepare the service

Machine 2 **[WHERE]**

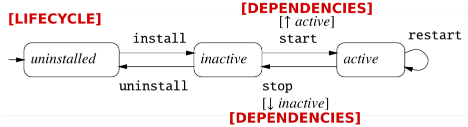
Web-server (WS) **[WHAT]**

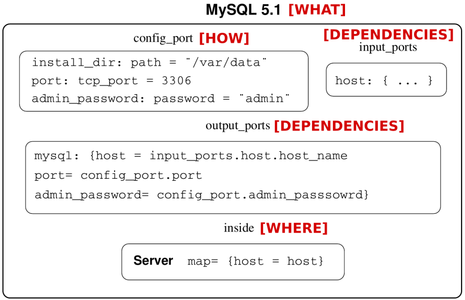
[HOW] **[LIFECYCLE]**

1. Install
2. Configure firewall
3. Download
4. Configure parameters
5. Start the service

[DEPENDENCIES]: WS(4) → DB(3), WS(5) → DB(4) (lifeycle granularity)

Engage





MySQL 5.1 [WHAT]

config_port [HOW]

```
install_dir: path = "/var/data"
port: tcp_port = 3306
admin_password: password = "admin"
```

[DEPENDENCIES]
input_ports

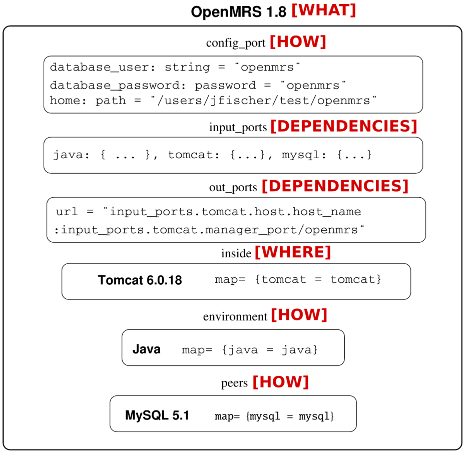
```
host: { ... }
```

output_ports [DEPENDENCIES]

```
mysql: {host = input_ports.host.host_name
port= config_port.port
admin_password= config_port.admin_passsowrd}
```

inside [WHERE]

Server map= {host = host}



OpenMRS 1.8 [WHAT]

config_port [HOW]

```
database_user: string = "openmrs"
database_password: password = "openmrs"
home: path = "/users/jfischer/test/openmrs"
```

input_ports [DEPENDENCIES]

```
java: { ... }, tomcat: {...}, mysql: {...}
```

out_ports [DEPENDENCIES]

```
url = "input_ports.tomcat.host.host_name
:input_ports.tomcat.manager_port/openmrs"
```

inside [WHERE]

Tomcat 6.0.18 map= {tomcat = tomcat}

environment [HOW]

Java map= {java = java}

peers [HOW]

MySQL 5.1 map= {mysql = mysql}

Aeolus

Particularities of Aeolus

- programmable lifecycle
- finer grain to model dependencies
- inspired from state machines
- inspired from component models

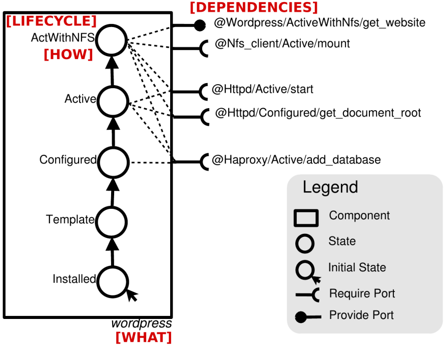


Table of Contents

1. Introduction
2. Overview of deployment tools
3. Parallelism within deployments
4. The Madeus deployment model
5. Evaluation
6. Conclusion

Our goals

Performance related to **[WHEN]** $\equiv$ **[LIFECYCLE, DEPENDENCIES]**

- structured parallelism
- reach quickly a configuration
- avoid disruption time

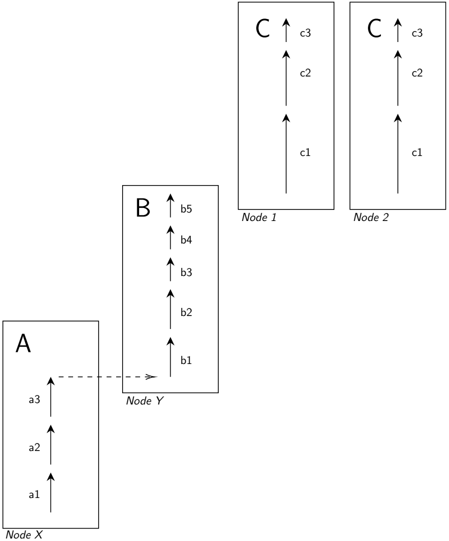
Safety

- formally-defined semantics
- tools to assist during design
- verification of properties

Performance through parallelism and dependencies

level1: multiple nodes, same action

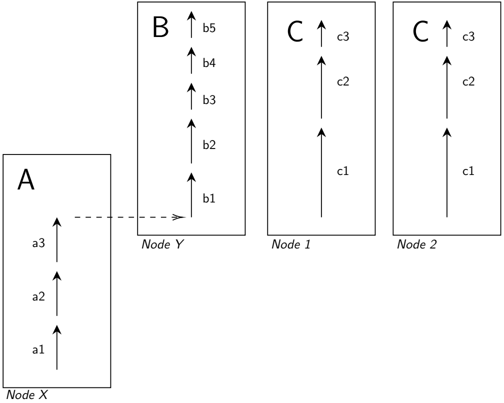
- no dependencies declared
- procedural execution order
- ANSIBLE



Performance through parallelism and dependencies

level2: level1+non-dependent components

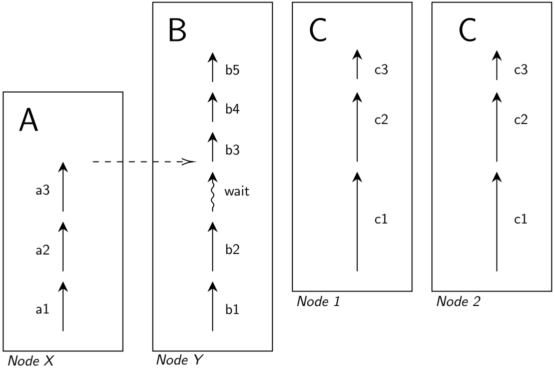
- dependencies at the component level
- Deployware, (basic) Tosca, Engage



Performance through parallelism and dependencies

level3: level1 + level2 + inter-component

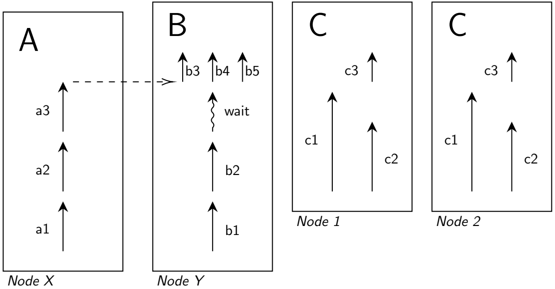
- dependencies at the task level
- (advanced) Tosca, Aeolus



Performance through parallelism and dependencies

level 4: level1 + level2 + level3 + intra-component

- internal task dependencies
- MADEUS



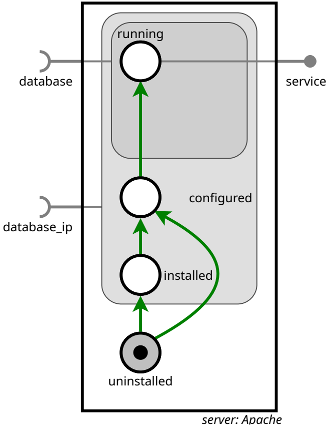
The finer the dependencies granularity is, the better is the efficiency (related to **[WHEN]**)

Table of Contents

1. Introduction
2. Overview of deployment tools
3. Parallelism within deployments
4. The MADEUS deployment model
5. Evaluation
6. Conclusion

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

Interfaces [DEPENDENCIES]

- use ports = requirements
- provide ports = provisions
- during execution: active/inactive

Control components in practice

Written by the component developers

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

        self.initial_place = 'uninstalled'

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

Control components in practice

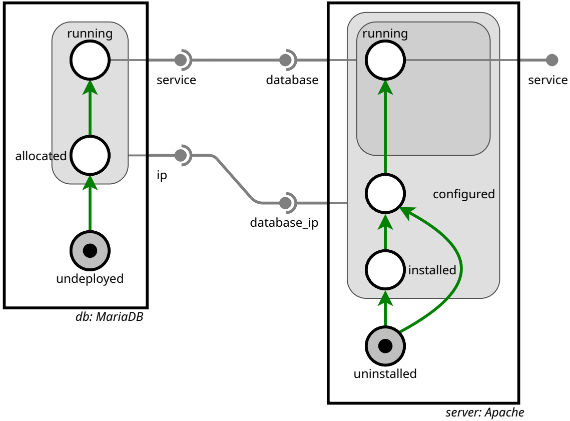
Written by the component developers

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

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

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

Assembly of components

Assembly of components [DEPENDENCIES]

instantiation of component types and connections of ports



Assembly of components in pratice

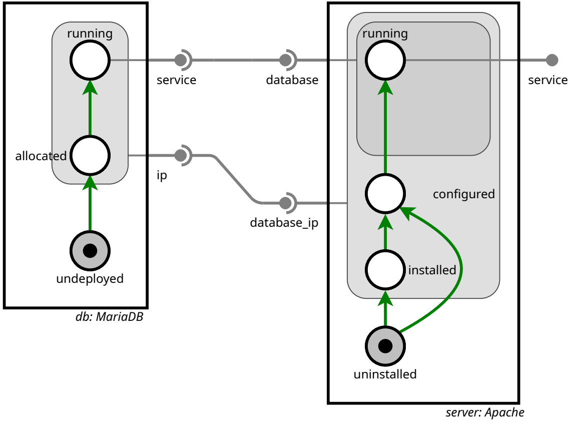
Written by the assembly developers, architect, DevOps

```python
from components.mariadb import MariaDB
from components.apache import Apache

    class ApacheWithDB (MadeusAssembly):
        def create():
        self.components = {
            'server': Apache(),
            'db': MariaDB()
        }
        self.dependencies = [
            ('server', 'database_ip', 'db', 'ip'),
            ('server', 'database', 'db', 'serv')
        ]

    if __name__ == '__main__':
        assembly = ApacheWithDB()
        assembly.run()
```

Execution example

Execution semantics

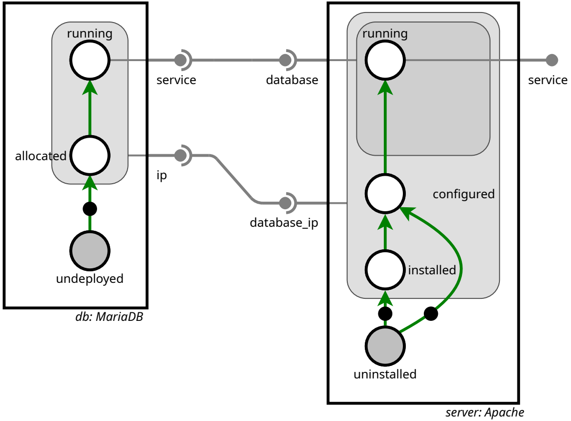
Initial places



Execution example

Execution semantics

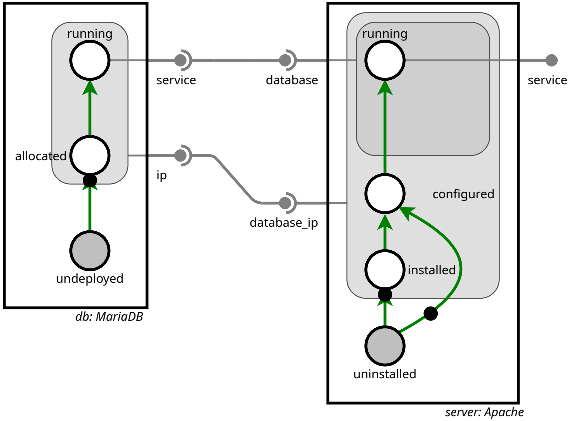
Firing transitions, parallel transitions



Execution example

Execution semantics

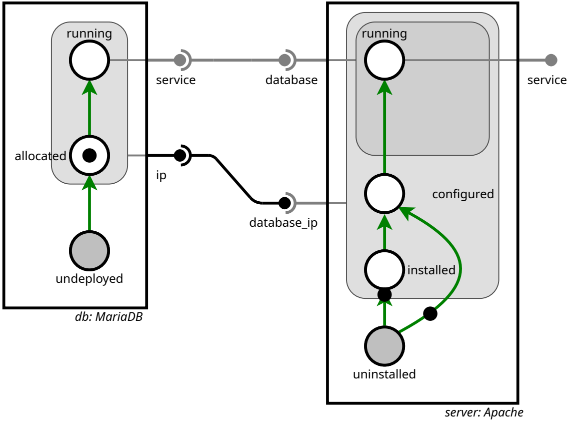
Entering places, inter-coordination through connections



Execution example

Execution semantics

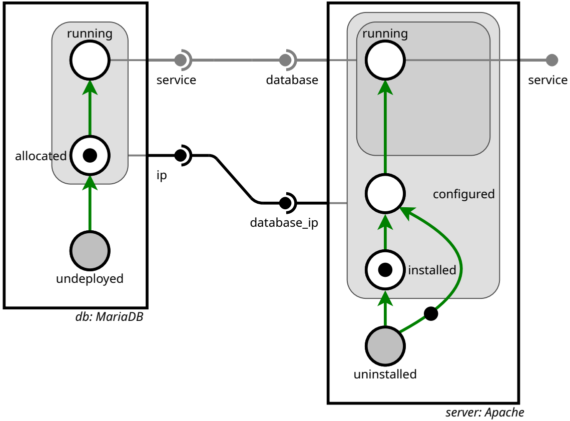
Reaching places, inter-coordination through connections



Execution example

Execution semantics

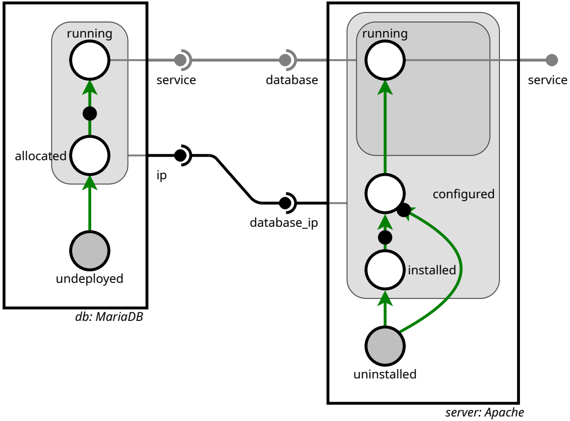
Reaching places, inter-coordination through connections



Execution example

Execution semantics

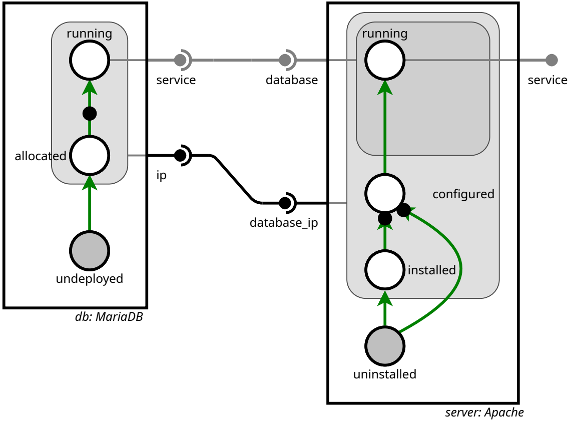
Reaching places, intra-coordination



Execution example

Execution semantics

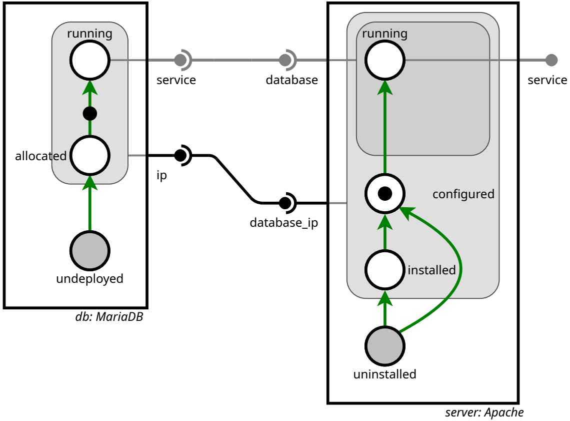
Reaching places, intra-coordination



Execution example

Execution semantics

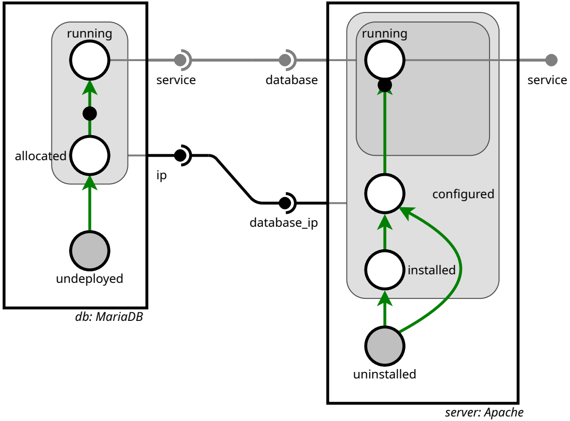
Reaching places, intra-coordination



Execution example

Execution semantics

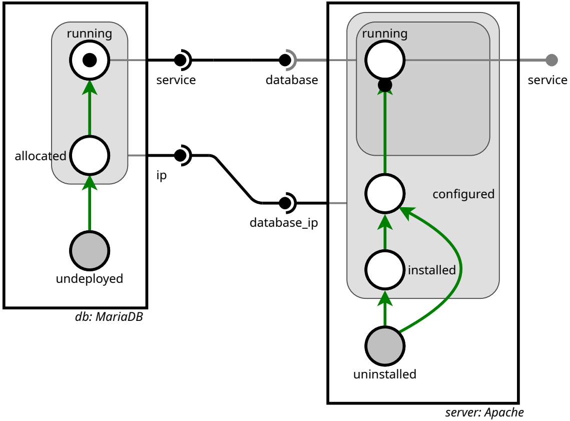
Reaching places, inter-coordination through connections



Execution example

Execution semantics

Reaching places, inter-coordination through connections



Execution example

Execution semantics

Reaching places, inter-coordination through connections

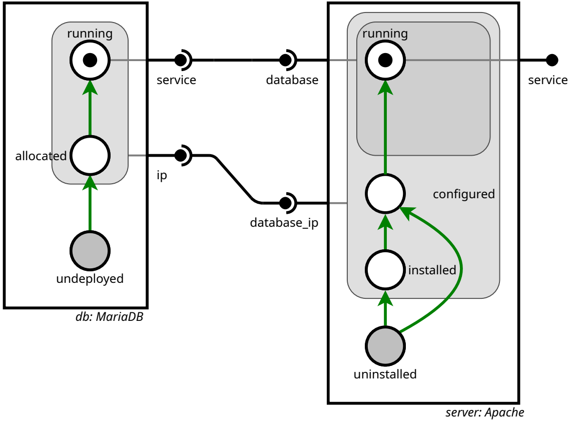


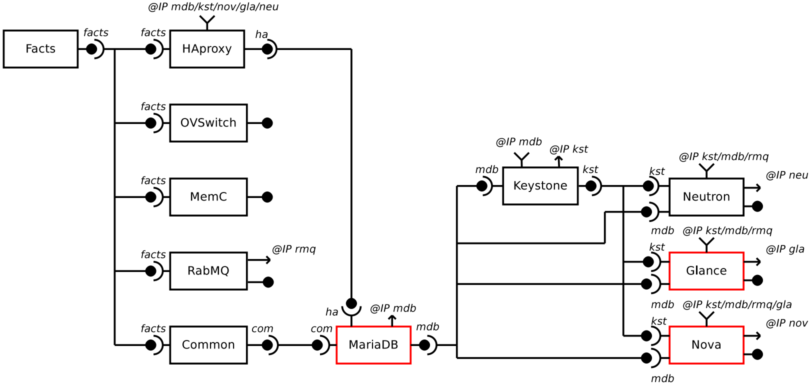
Table of Contents

1. Introduction
2. Overview of deployment tools
3. Parallelism within deployments
4. The Madeus deployment model
5. Evaluation
6. Conclusion

Use-case - deployment of OpenStack

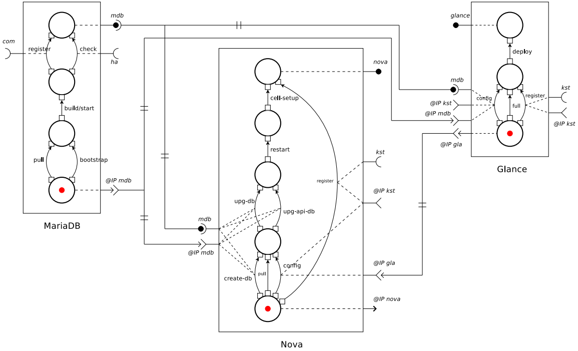
Basic production OpenStack as defined in Kolla-Ansible:

- 11 components, 36 services in total, deployed on three nodes,
- Docker container-based deployment,
- combination of Docker images, Ansible,
- more than 20 minutes to deploy.



Evaluation setup

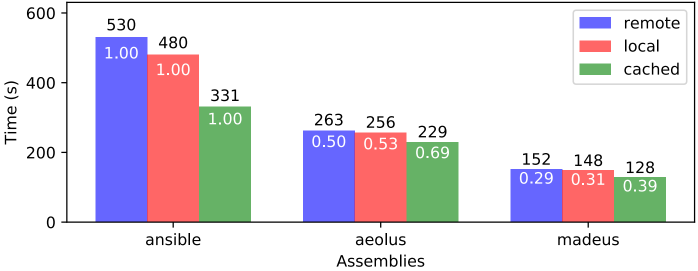
- Comparison to Kolla-Ansible (production tool), and Aeolus (literature),
- emulation of Aeolus (no longer maintained) with Madeus,
- three different versions of the deployment `remote`, `local`, `cached`,
- Reproducible experiments on Grid'5000.



Evaluation on the deployment of OpenStack

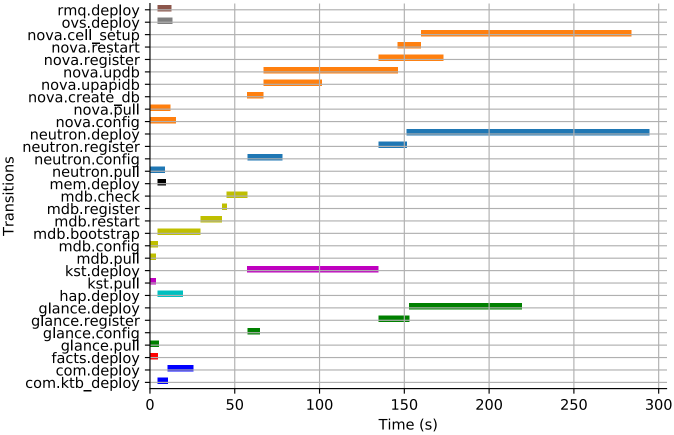
Results on three nodes Ecotype (Nantes) of Grid'5000

Cluster	CPU	Memory	Network
Nantes Ecotype	2× Intel Xeon E5-2630L v4, 10 cores/CPU	128GB	2× 10Gbps

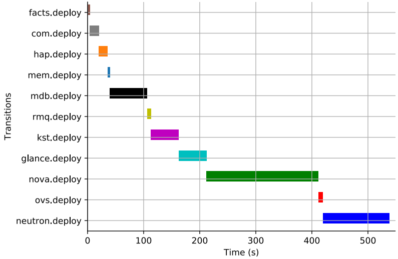


Evaluation on the deployment of OpenStack

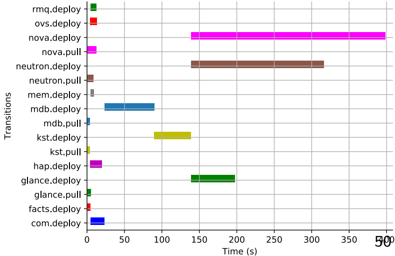
Madeus



Ansible



Aeolus



Evaluation on the deployment of OpenStack

- Traces of the OpenStack continuous Integration platform
- February 19 to February 27 2020
- Exactly 2963 deployments of OpenStack have been recorded (329 runs per day)
- Projection of the gain with deployment times of our experiments in remote mode

	Kolla	MADEUS	gain
reference time(s)	529	150	71%
projection on 9 days(h)	435	123	71%
projection on av./day(h)	48	14	71%

Table of Contents

1. Introduction
2. Overview of deployment tools
3. Parallelism within deployments
4. The Madeus deployment model
5. Evaluation
6. Conclusion

Conclusion

- Deployment problem and its complexity
- Need for software engineering practices
- Overview of deployment tools and academic contributions
- Presentation of MADEUS
- Evaluation of MADEUS

Questions?

Configuration management

Focus of configuration management tools

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

- ANSIBLE: Procedural approach, agentless on top of SSH
  - sequential order of roles and tasks
  - similar to a well structured script
  - abstraction on top of SSH and system commands
- PUPPET: Declarative approach with a master/worker architecture
  - **[HOW]** is mostly hidden for the user
  - specify what you want not how to get it
  - interesting for management

ANSIBLE [WHERE]

Inventory file

ANSIBLE [WHAT]

Playbook, roles, vars

ANSIBLE [HOW]

Tasks, templates, vars, handlers

Example of ANSIBLE Apache role on GITHUB.

Provisioning

Focus of provisioning tools

Initially designed to provision the servers, network, platforms etc.

- Cloud Formation and Heat: Specific to a given Cloud provider
  - resp. AWS, OpenStack
- Terraform, Juju, and Tosca: Generic to any provider
  - write your own providers

Terraform **[WHAT]**

resources, variables

Terraform **[WHERE]**

resources (provisioning)

Terraform **[HOW]**

variables, user_data, local_exec, write custom providers

Example of an AWS EC2 instance provisioning with a webserver with Terraform

Writing custom Terraform providers

Combining Ansible and Terraform

Doing provisioning with Ansible

Docker ecosystem

Focus of configuration management tools

Solve portability problem and reduce configuration issues through containers.

Docker [HOW]

DockerFile and Docker images

Docker [WHAT]

Docker images and Docker Compose

Docker [WHERE]

Kubernetes and Docker Swarm

- lighter than virtual machines (if sharing the same OS kernel)
- the DockerFile still has to be written at some point
- Docker Compose to write an ordered list of Docker images to deploy locally
- Kubernetes and Docker Swarm to manage set of containers, their placement, their replicas

Example of Docker Compose LAMP deployment on GitHub.