Toward efficient and safe deployment and reconfiguration of distributed software

68NQRT seminar - IRISA

Hélène Coullon

2018-12-13

Assistant professor at IMT Atlantique, Inria reasearch chair, LS2N

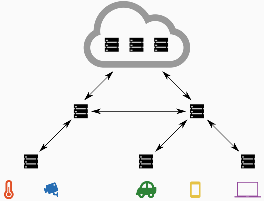
Table of contents

Introduction

Distributed infrastructure

Distributed infrastructure

- Set of distributed interconnected machines
- e.g. HPC clusters, Local servers, Grid (Grid'5000, EGI), private/public Cloud (Amazon), Fog and Edge
- Properties: heterogeneous, large scale



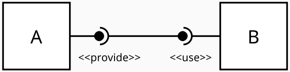
Distributed software

Distributed software

- Software composed of multiple modules
- Dependencies between these modules

Component models

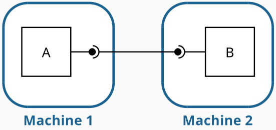
- Set of modules ≈ set of components
- Black box of code
- Explicit *interfaces* through *ports* (e.g. functional dependencies: use-provide)
- *Assembly languages* to define an application
  - instances of components
  - connections between ports
- Separation of concerns, maintainability etc.



What is the deployment?

Deployment

- Placement (mapping modules / resources)
- **Software commissioning**
  - Allocation of resources
  - Creation and configuration of the components
  - Connection of the components
  - etc.



Problem statement

Soft. and infrastructures evolution

- bigger distributed software
  - *e.g.*, OpenStack, Spark, smart-* apps etc.
- massively geo-distributed heterogeneous infrastructures
  - *e.g.*, Fog, Edge, IoT

Problem statement

- deployment automation
- generic deployment
- easy deployment
- efficient deployment
- safe deployment

Deployment

Deployment

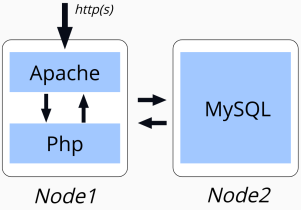
Automated deployment

Automated deployment - Scripts

Installing LAMP (Linux Apache MySQL Php) on a (bare metal) server

Steps

- install an operating system (Linux)
- install (`apt-get install`)
- configuration of Apache, MySQL
- network configuration
- check installation, add plugins etc.

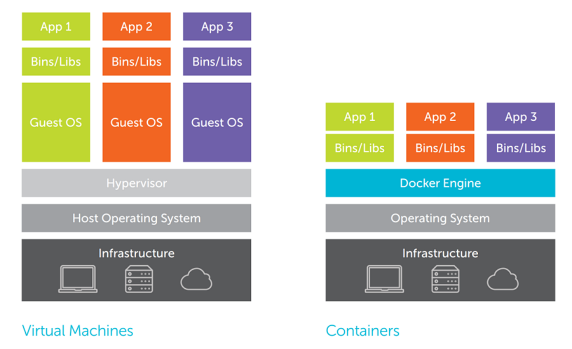


Portability issues and errors

- commands are not portable from one OS to another
- libraries may have different versions on different OSs
- OS specific configurations may be needed
- error prone

Automated deployment - Virtualization

Enhances portability of deployment



Virtual images

- images are needed both for VM and containers
- user can build her/his own images (bootstrap installation)
- user can use existing images (Docker registry, VM templates)

Automated deployment - Virtualization

Installing Apache using Docker and centos basic image

Docker container virtualization

- understand Docker commands
- write or customiza Docker files
- bootstrap problem for IT administrators

Cloud providers

- understand the provisioning API of Cloud providers (AWS Cloud formation, Heat orchestration Template etc.)
- system commands may still be needed to configure or customize the VM
- worse bootstrap problem for Cloud administrators (*e.g.*, OpenStack deployment)

Automated deployment - Tools and models

Ansible, Puppet, Chef

- abstractions above SSH and bash scripts (*e.g.*, yaml, python)
- generic deployment tools (*e.g.*, bare metal, containers or VMs)
- deployment procedure splited in different parts
    - hierarchical view
    - *e.g.*, roles, playbooks, tasks
- data communication between parts
    - *e.g.*, handled by Jinja2
- same set of operations can be applied on multiple hosts
- strict sequential order between different parts (roles, playbooks and tasks)
- bootstrap problem is very limited (ssh, python on nodes)

LAMP deployment in Ansible

LAMP deployment in Puppet

Deployment

State of the art

Classification metrics

Very complex deployment ecosystem: scripts, virtualization, ansible, puppet, chef, kubernetes, juju, etc.

State of the art limited to deployment models and tools

Deployment and components

- 1 module $\approx$ 1 component (*e.g.*, role in Ansible)
- each component has a deployment life-cycle (*e.g.*, stoped, configured, installed etc., plabooks and tasks in Ansible)
- life-cycle management automation is needed

Properties

- Programmable life-cycle (expressivity, safety)
- Life-cycle coordination (automation and safety)
- Parallelism (operations on multiple hosts, inter-component, intra-component)

State of the art

Production tools

Model	Programmable life-cycle	Life-cycles coordination	Parallelism
Ansible/Puppet/Chef	Yes	Yes (sequential)	Same component multiple hosts
Kubernetes	No	No	inter-component
Juju	No	Yes (fixed)	inter-component

Academic research

Model	Programmable life-cycle	Life-cycles coordination	Parallelism
CCM/L2C/Deployware	No	Yes (fixed)	inter-component
Fractal/GCM	Yes	No	inter-component
TOSCA	Yes	No	inter-component
Blender/Aeolus	Yes	Yes	inter-component+

- Madeus is a new component-based deployment model
- Madeus is inpired from Aeolus
- Madeus enhances the efficiency of deployements

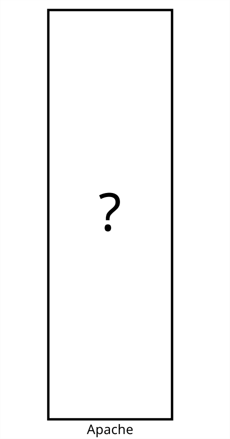
Deployment

Madeus

Madeus - Definitions

Component

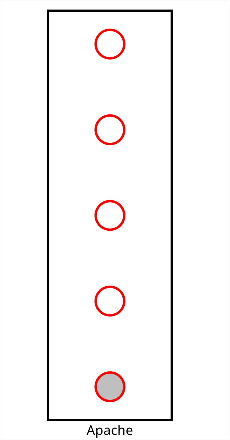
- Usually corresponds to a module of a distributed application
- Has its own life-cycle



Madeus - Definitions

Place

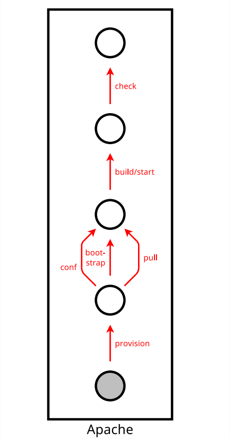
- A “milestone” in the component life-cycle
- Acts as a synchronization mark if multiple actions are performed in parallel



Madeus - Definitions

Transition

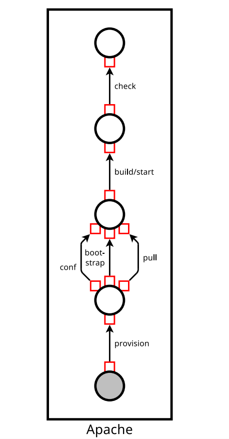
- Bound to an action (i.e. a function)
- From one place to another



Madeus - Definitions

Dock

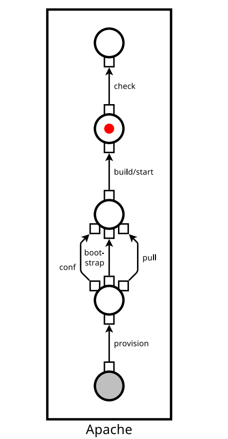
- Allows to handle synchronization of parallel actions with a graphical object
- Attached to places
- Two kinds of docks: input and output
- Connection points for transitions



Madeus - Definitions

Token

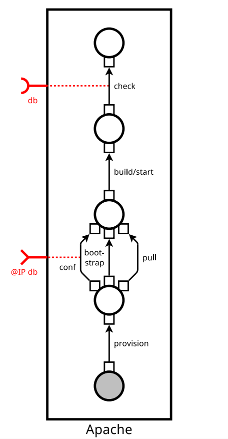
- Represents the state in the life-cycle of the component
- Either present on or absent of each place, dock and transition



Madeus - Definitions

Input port

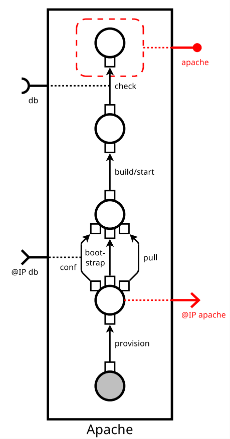
- Bound to transitions that require some data/service
- These transitions can only be triggered when the port is connected



Madeus - Definitions

Output port

- Data output ports: provide data (e.g. IP address)
- Service output ports: indicates that a service is provided by the component



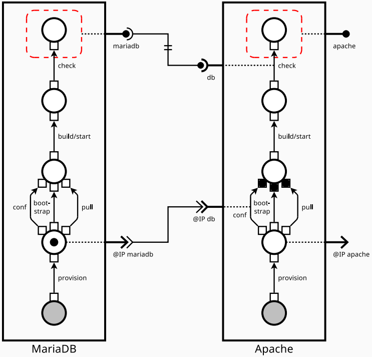
Madeus - Definitions

Assembly

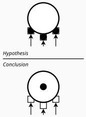
- Set of instances of components
- Connections between their ports
- Similar to a *main* function

Configuration $\langle mk, ebl, val \rangle$

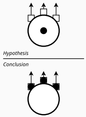
- *mk*: marking = location of tokens
- *ebl*: enabled = whether or not connections are enabled
- *val*: values = values stored in the data output ports



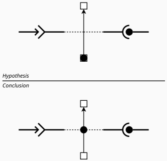
Madeus - Operational Semantics



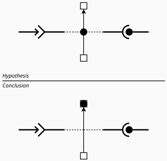
input docks to place



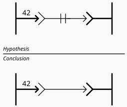
place to output docks



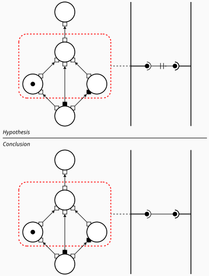
fire transition



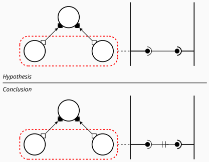
end transition



enable data connection

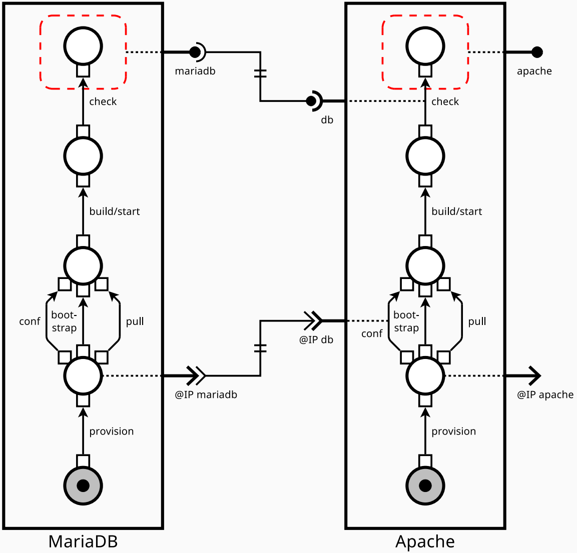


enable service connection

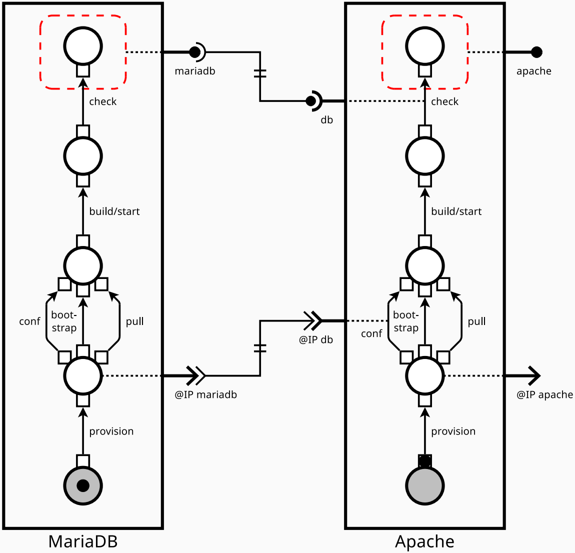


disable service connection

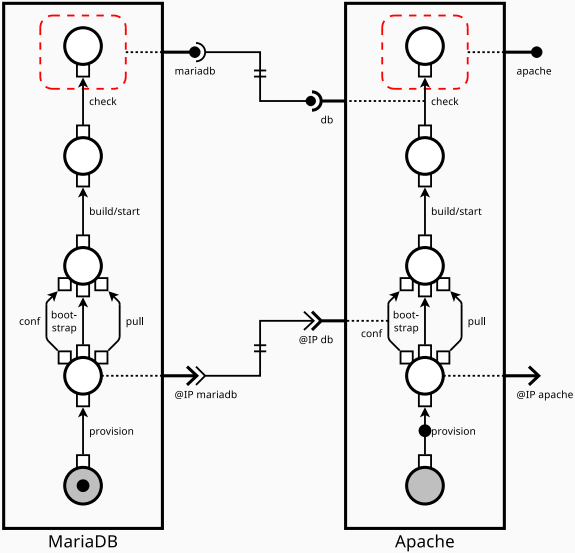
Madeus - LAMP example



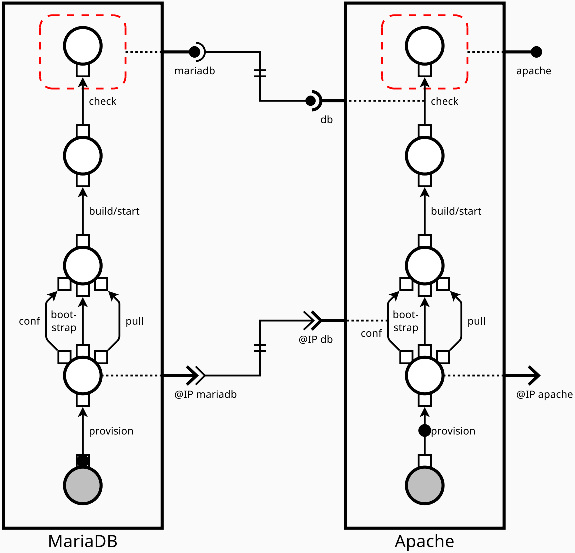
Madeus - LAMP example



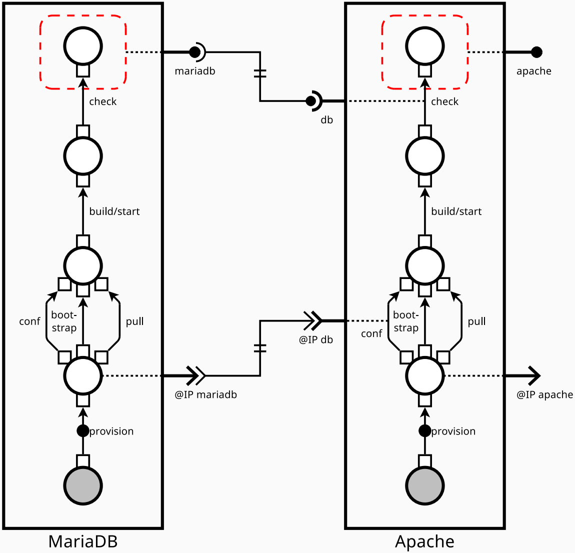
Madeus - LAMP example



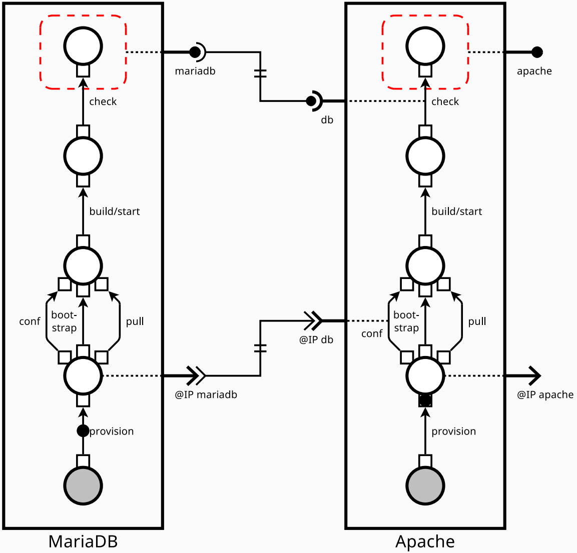
Madeus - LAMP example



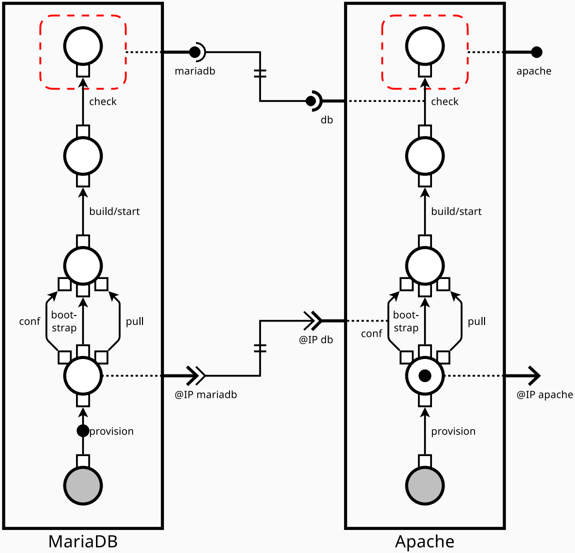
Madeus - LAMP example



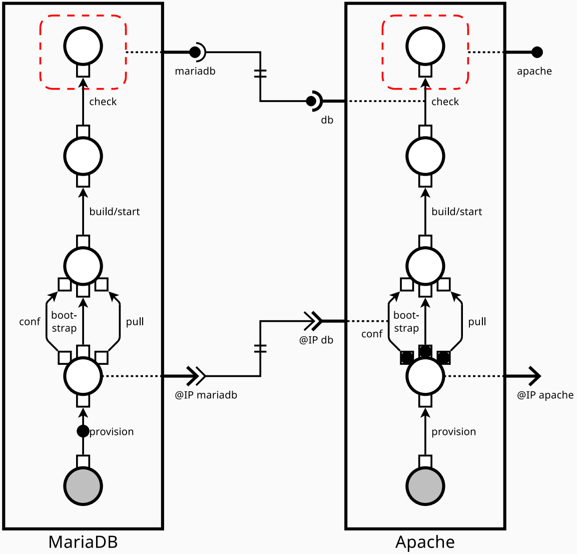
Madeus - LAMP example



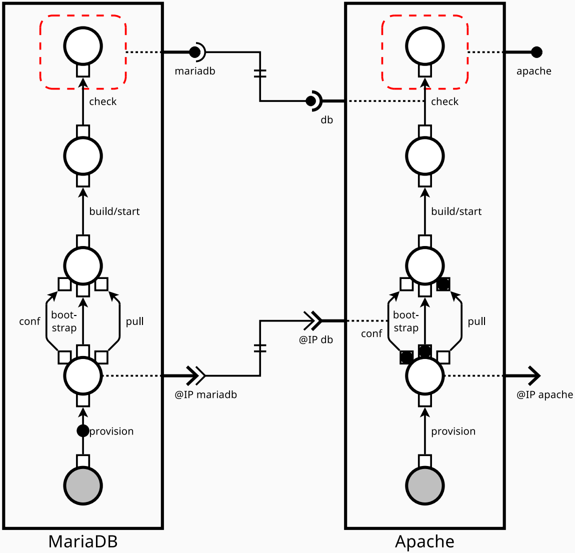
Madeus - LAMP example



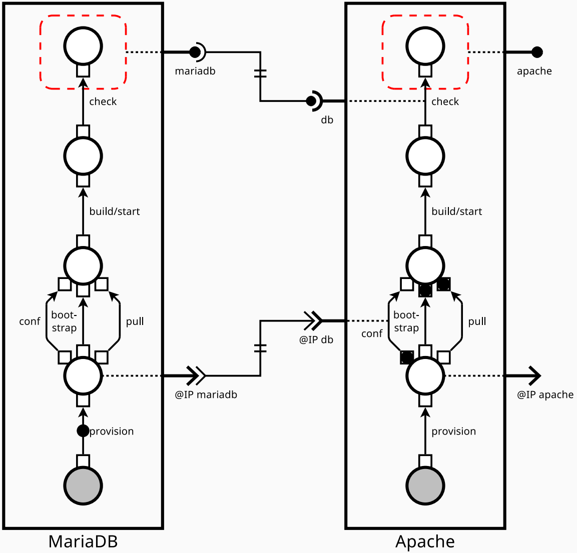
Madeus - LAMP example



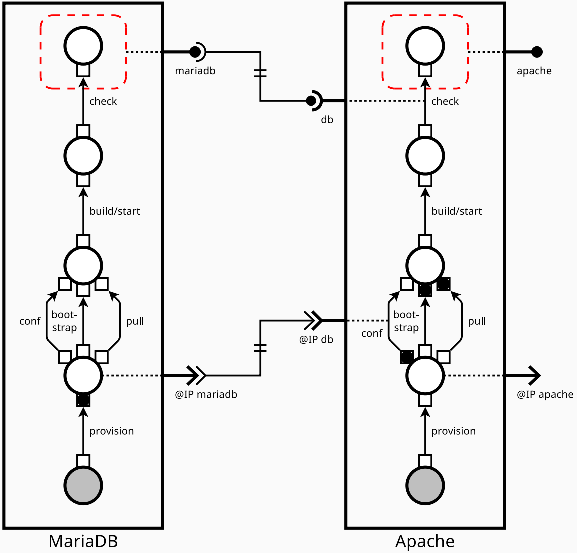
Madeus - LAMP example



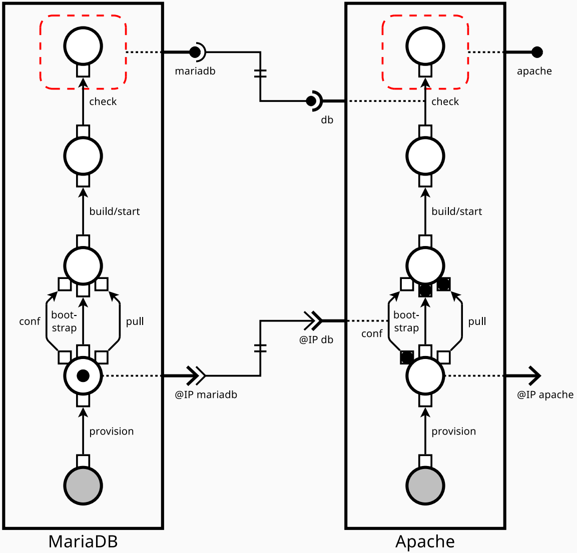
Madeus - LAMP example



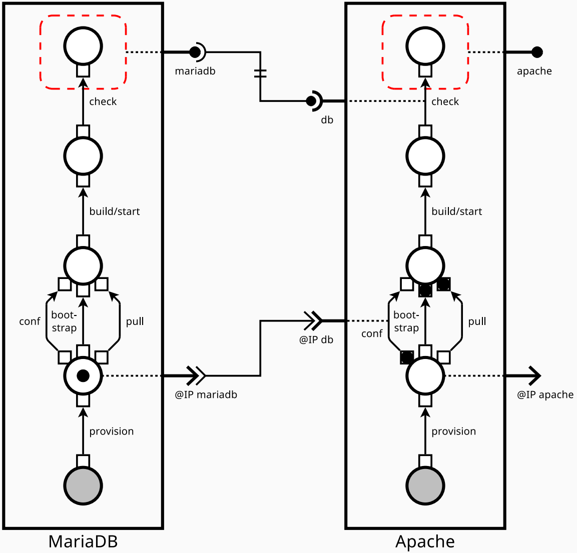
Madeus - LAMP example



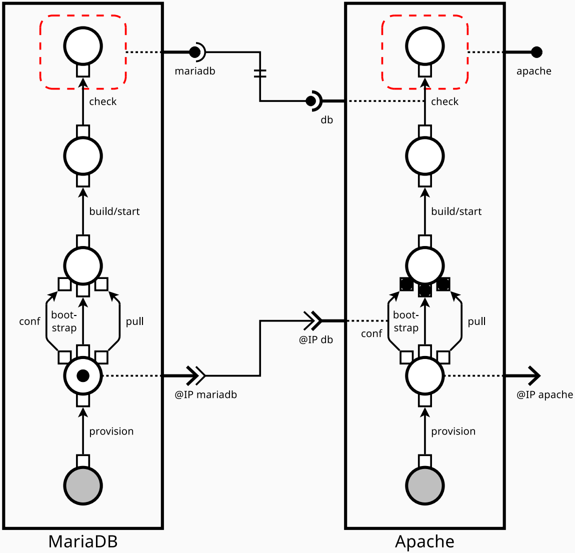
Madeus - LAMP example



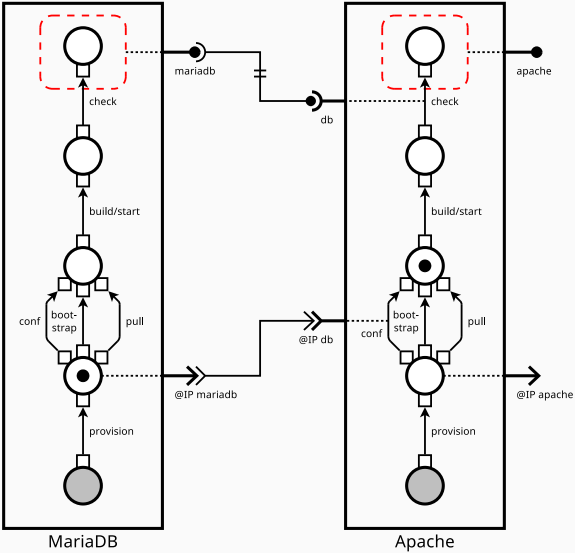
Madeus - LAMP example



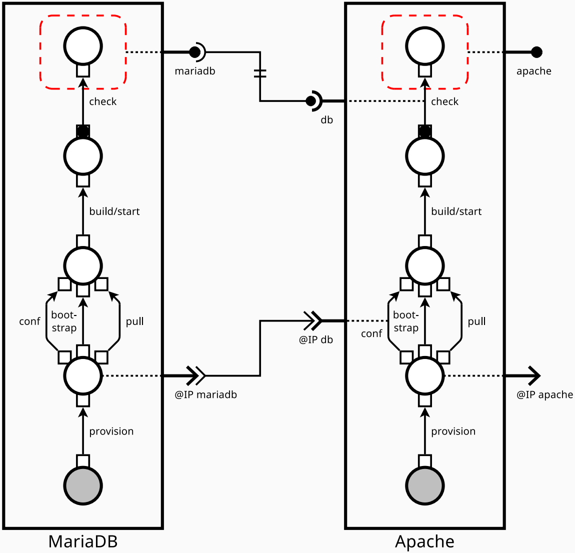
Madeus - LAMP example



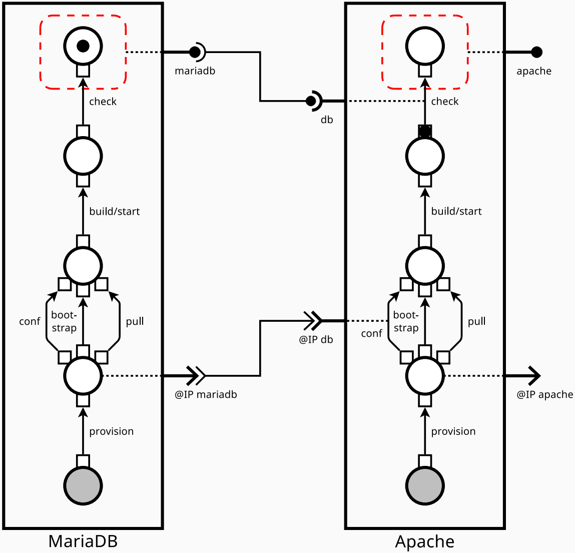
Madeus - LAMP example



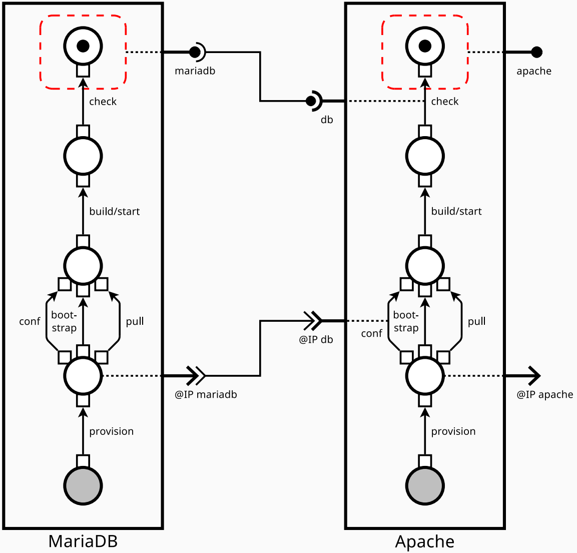
Madeus - LAMP example



Madeus - LAMP example



Madeus - LAMP example



Madeus - Evaluation

OpenStack (bootstrap)

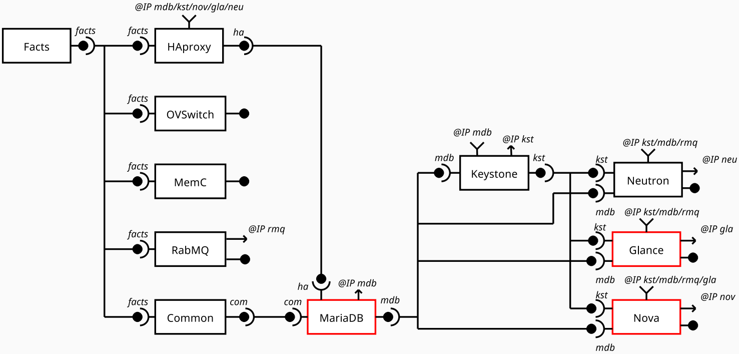
- open source operating system of the Cloud
- large distributed software
- modular architecture composed of more than 30 projects
- more than 150 services

Kolla-Ansible OpenStack Deployment

- our deployment reference is the Kolla project
- production tool to deploy OpenStack ONLY
- deploy a containerized minimal OpenStack by using **Ansible**
- 11 projects, 36 services
- deployment on three nodes: controller (16 services), network (11 services), compute (9 services)

Madeus - Evaluation

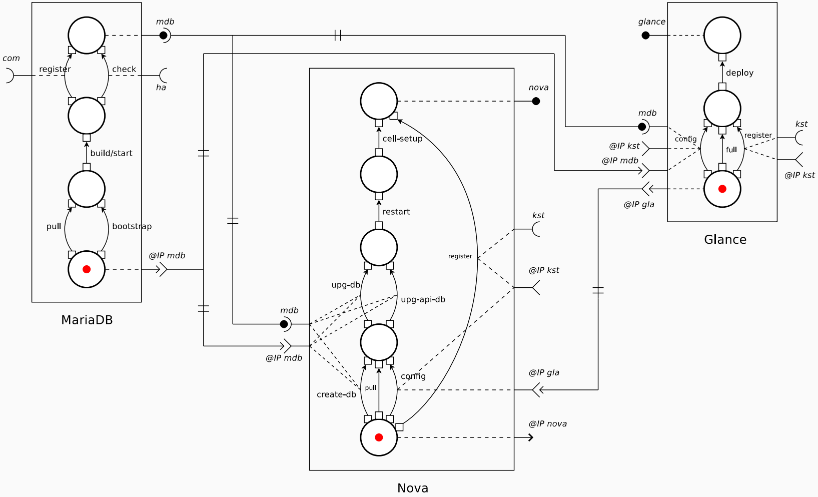
Full coarse-grain view of the Madeus deployment



Places	Transitions	Ports
32	30	47

Madeus - Evaluation

Detailed Madeus components (MariaDB, Nova, Glance)



Madeus - Evaluation

Deployment versions

- spmd-1t = Kolla-ansible
- dag-2t = Aeolus (simulated with Madeus, no parallel transitions)
- dag-nt = Madeus

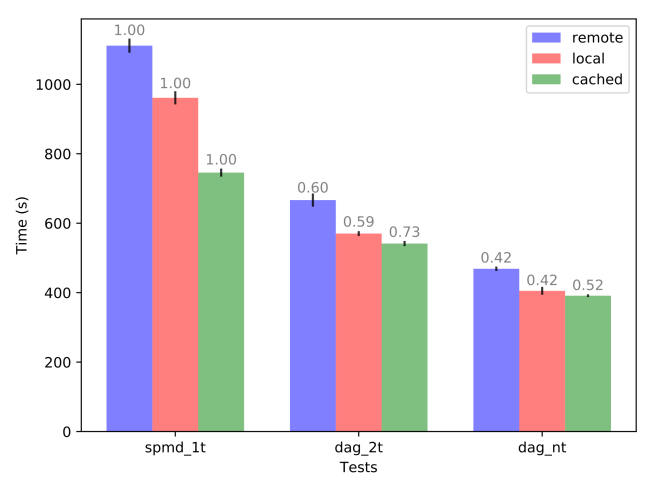
Docker image management

- remote (Docker Hub)
- local (dedicated local registry in the cluster)
- cached (all nodes already store docker images)

Cluster	CPU	Memory	Network
Taurus (g5k)	2 x 6 cores/CPU	32GB	10 Gbps

	Compute	Network	Control
Number of images	9	11	16
Total size (MB)	2767	2705	4916

Madeus - Evaluation

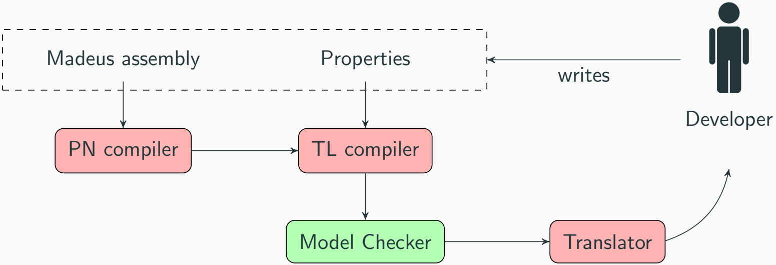


- 58% faster than Kolla-Ansible
- 32% faster than Blender-Aeolus

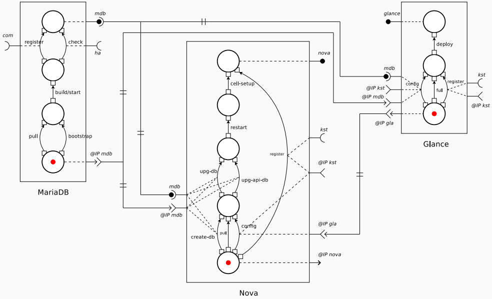
Deployment

Verification and Madeus

Madeus and Petri nets

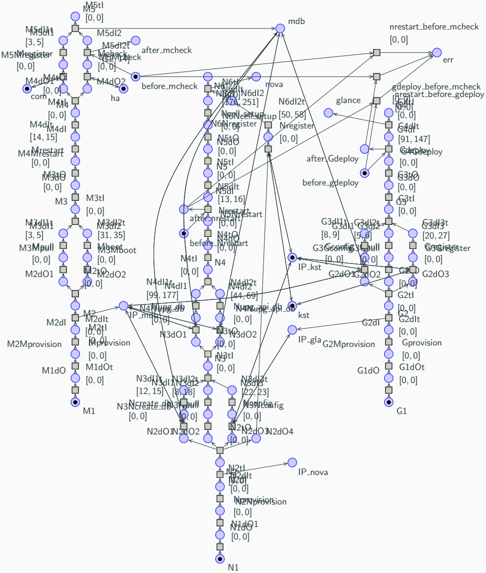


Madeus and Petri nets



- Madeus places ⟶ Petri net places
- Madeus docks ⟶ Petri net places
- Madeus transitions ⟶ Petri net places
- Madeus connections ⟶ Petri net places
- places and transitions of the Petri net connected such that the same semantics is applied
- specific case for groups, not detailed in this talk

Madeus and Petri nets



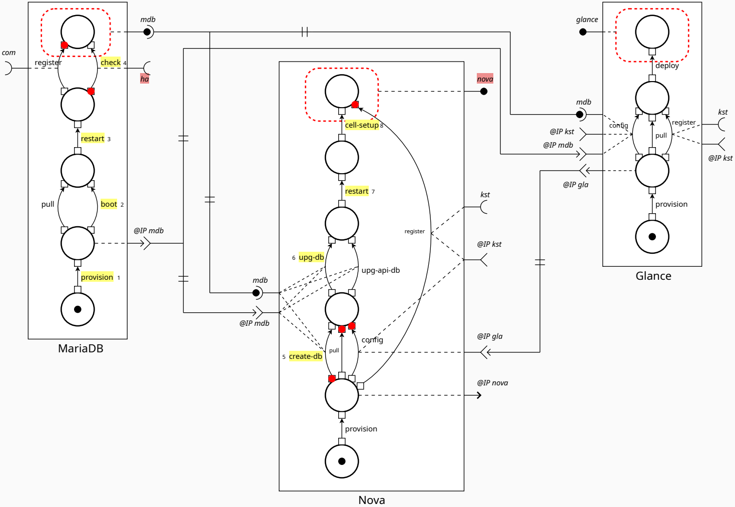
Property language

```
def addInterval(self, transition, min, max)
def addDeployment(self, name, list_places)
def deployability(self, name_deployment, with_intervals,
                  traces)
def sequentiality(self, transition1, transition2, ...)
def parallelism(self, full_assembly, list_components)
def boundaries(self, traces)
```

Properties to temporal logic

- deployability ⟶ inevitability
- sequence ⟶ observer subnet + invariant
- parallelism ⟶ max($\sum$(reachable markings))
- boundaries: min/max costs + critical path

Traces and debug



Perspectives

Perspectives

- Proof of semantic equivalence between Madeus and the Petri net
- Conditions and errors in Madeus
- Probabilistic model
- Game theory

Other ongoing work

Coq modelization of Madeus (proofs on the model)

Reconfiguration

Reconfiguration

Context

Context

Reconfiguration

- Deployment = specific reconfiguration
- Rolling upgrade
- Dynamic resources (add/remove nodes, failures)
- Dynamic software topology (add/remove/replace/new configuration)
- Other dynamic information (security/energy etc.)

State of the art

Metrics

- *Performance*: as fast as possible
  - minimize downtime
  - minimize execution time
- *Expressivity*: handling many kinds of reconfiguration
- *Separation of concerns* between developers and reconfiguration designers
  - each actor does what is in their area of expertise

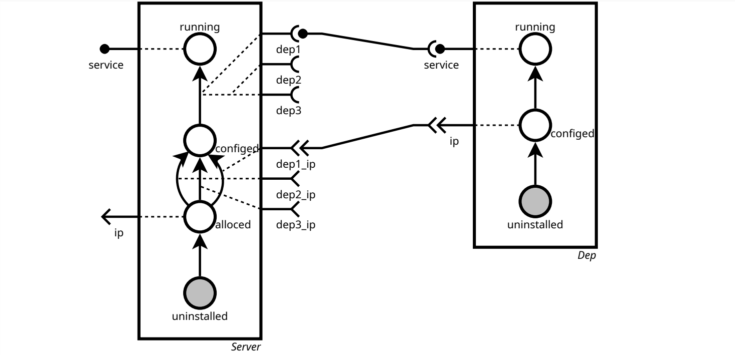
Objectives

- Extend Madeus with reconfiguration to inherite its efficiency
- Increase separation of concerns compared to Aeolus

Reconfiguration

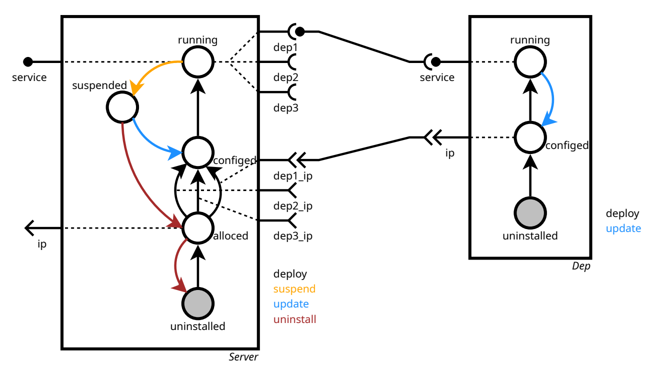
Madeus++

Madeus assembly



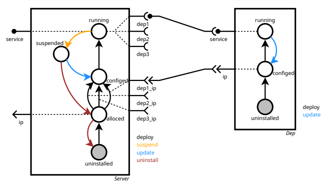
- Efficient deployment (programmable life-cycle, parallelism)
- No reconfiguration

Madeus++ assembly



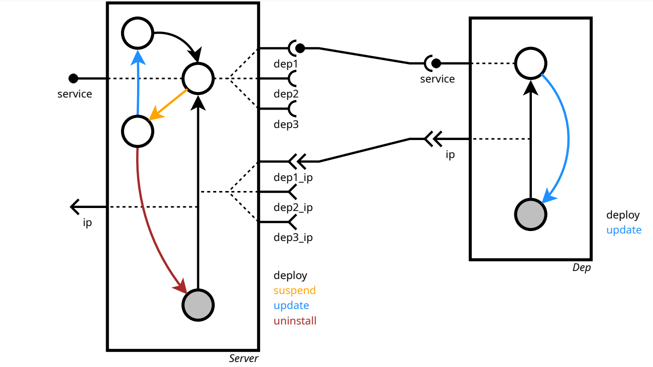
- Introduction of behaviors within Madeus
- Add a reconfiguration language composed of 6 operations: *add*, *del*, *connect*, *disconnect*, *changeBehavior*, *wait*

Madeus++ reconfiguration



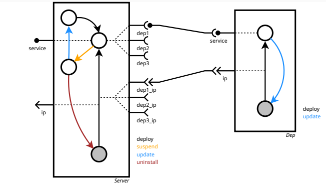
```
changeBehavior(server,suspend)
changeBehavior(dep,update)
wait(server)
disconnect(server,dep1,dep,service)
changeBehavior(server,update)
wait(dep)
changeBehavior(dep,install)
wait(server)
changeBehavior(server,install)
```

Behavioral interfaces



- Simplified interfaces for the reconfiguration designer
- Increases the separation of concerns

Madeus++ reconfiguration



```
changeBehavior(server,suspend)
changeBehavior(dep,update)
wait(server)
disconnect(server,dep1,dep,service)
changeBehavior(server,update)
wait(dep)
changeBehavior(dep,install)
wait(server)
changeBehavior(server,install)
```

Madeus++ ongoing work

- A prototype of MAD++ has been implemented in Python
- Experiments on real case study (database migration)
- Proof of equivalence between madeus++ and behavioral interfaces

Reconfiguration

VeRDi project

VeRDi project

Verified Reconfiguration Driven by execution

Automated reconfiguration execution

- programmable reconfiguration protocols
- efficient reconfiguration (parallelism)
- safe reconfiguration
- decentralized reconfiguration

A few challenges

- programmable protocols
- high level of parallelism
- static and dynamic verifications
- verification of decentralized reconfiguration (local knowledge)
- use verification as a tool to help the developer

Conclusion

Conclusion

- Deployment and Madeus
    - efficiency
    - evaluation on OpenStack
- Madeus and Petri nets
    - transformation of a Madeus assembly to a Petri net
    - transformation of the property language to temporal logic
    - use a model checker for verification and debug
- Reconfiguration and Madeus++
    - efficiency
    - separation of concerns
- The VeRDi project

People involved

Madeus and Madeus++

Dimitri Pertin

Former postdoc

Maverick Chardet

Ph.D. student

Christian Perez

DR Inria, Lyon

Team leader Avalon Inria, LIP

Madeus and Petri nets

Didier Lime

HDR Ecole Centrale de Nantes

Team leader STR LS2N

Claude Jard

Professor Université de Nantes

Head of LS2N

Madeus and Coq

Frederic Loulergue

Professor at NAU USA