Towards Decoupling Application Descriptions from their Execution Environments

Journées TARANIS, Lyon

Quentin GUILLOTEAU[1], Hélène COULLON[2], Christian PEREZ[1]

2025-10-17

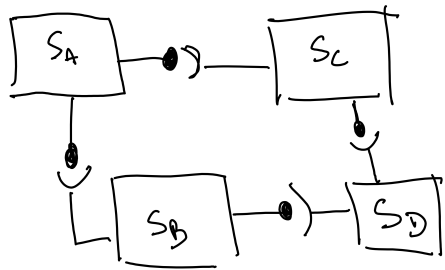
[1]Inria
[2]IMT Atlantique

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030
PROGRAMME DE RECHERCHE
CLOUD

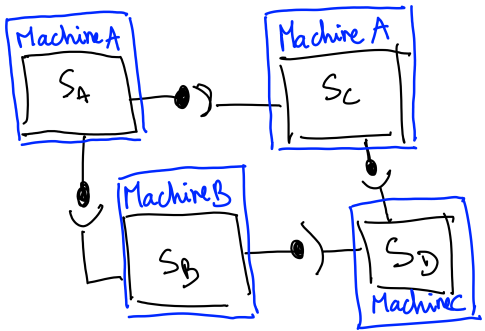
Context – Deployment of Cloud Applications

- deployment of cloud applications
- application = set of (micro)services with interfaces, interconnected
- can be modeled as black box components with ports (e.g., `USE`/`PROVIDE`) for better composability
- connected components = assembly



Context – Deployment of Cloud Applications

- deployment of cloud applications
- application = set of (micro)services with interfaces, interconnected
- can be modeled as black box components with ports (e.g., `USE`/`PROVIDE`) for better composability
- connected components = assembly



↪ Software components need to be deployed onto "**machines**"

Resources, Provisioning, and Config. Management

What even are "machines"?

Does not have to be machines $\rightsquigarrow$ VMs, containers, baremetal, etc.

$\hookrightarrow$ "Resources" $\rightsquigarrow$ Execution units

Usually:

1. Provisioning (e.g., via Terraform):
   - Descriptions of the resources
   - Manual, depends on the tool used
2. Configuration Management (e.g., with Ansible):
   - Installation/configuration/etc.
   - Might depend on the resources

Resources, Provisioning, and Config. Management

What even are "machines"?

Does not have to be machines $\rightsquigarrow$ VMs, containers, baremetal, etc.

$\hookrightarrow$ "Resources" $\rightsquigarrow$ Execution units

Usually:

1. Provisioning (e.g., via Terraform):
   - Descriptions of the resources
   - Manual, depends on the tool used
2. Configuration Management (e.g., with Ansible):
   - Installation/configuration/etc.
   - Might depend on the resources

$\hookrightarrow$ Mapping between the resources and the components?

Research Statement

How to model component assemblies to support their deployment in **various execution contexts**?

Motivations and Research Statement

Research Statement

How to model component assemblies to support their deployment in **various execution contexts**?

Motivations – Portability

- Application developer not responsible to execution context
- Avoiding vendor-locking
- Multi-cloud deployments
- Resource heterogeneity

Motivations and Research Statement

Research Statement

How to model component assemblies to support their deployment in **various execution contexts**?

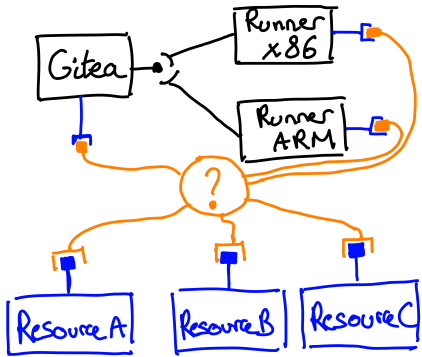
Motivations – Portability

Motivations – Unavailable Services / Urgent Computing

- Resource provider has some issue
- Some resources or services are not available
- but need to deploy the application right now, with whatever resources available

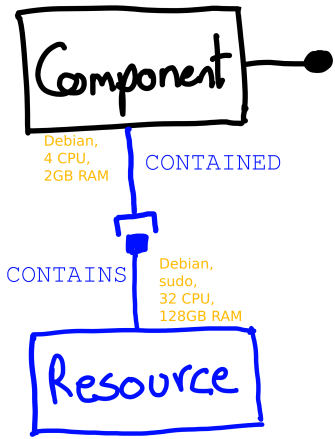
Modelisation

⇝ Model resources as components



Model – Components, ResourcePorts, and Attributes

- Components have the classical `USE/PROVIDE` ports
- but also ResourcePorts
  - port types (eq. to use/provide): `CONTAINED/CONTAINS`
  - models where a component is deployed
  - One port = one resource
  - One component can model several resources
- Set of **functional** attributes linked to the ports (e.g., `CPU`, `MEMORY`, etc.)
- Set of non-functional attributes linked to the ports (e.g., FLOPs, CO2/W, etc.)



Model – Instances and Assemblies

Instances

Instance = Component Type + Identifier

(For both applicative and resource components!)

Component Type = Ports + ResourcePorts + Attributes

Model – Instances and Assemblies

Instances – Component Type + ID

Assembly

- Set of component instances
- Port Connections
  - between application ports
  - but also between ResourcePorts

Model – Instances and Assemblies

Instances – Component Type + ID

Assembly – Set of Component Instances + Port Connections

Incomplete Assembly

Assembly where at least one ResourcePort of type `CONTAINED` is not connected

Complete Assembly

Assembly where all the ResourcePorts of type `CONTAINED` are connected

$\hookrightarrow$ A complete assembly is "deployable"

Model – Instances and Assemblies

Instances – Component Type + ID

Assembly – Set of Component Instances + Port Connections

Incomplete Assembly

Assembly where at least one ResourcePort of type `CONTAINED` is not connected

How to go from an incomplete to a complete assembly?

Complete Assembly

Assembly where all the ResourcePorts of type `CONTAINED` are connected

$\hookrightarrow$ A complete assembly is "deployable"

Mapper – Definition

Mapper

$$\text{Mapper} : \textit{Assembly} \rightarrow \textit{Assembly}$$

- Only manipulates ResourcePorts
- Does not have to return a complete assembly (i.e., can solve a sub-problem)

Mapper – Definition

Mapper

$$\text{Mapper} : \textit{Assembly} \times \textbf{Universe} \rightarrow \textit{Assembly}$$

- Only manipulates ResourcePorts
- Does not have to return a complete assembly (i.e., can solve a sub-problem)

Universe

- Set of resource component instances: resources already available
- Set of resource component types: resources that can be instantiated
- A relation to model if a resource can be deployed onto another

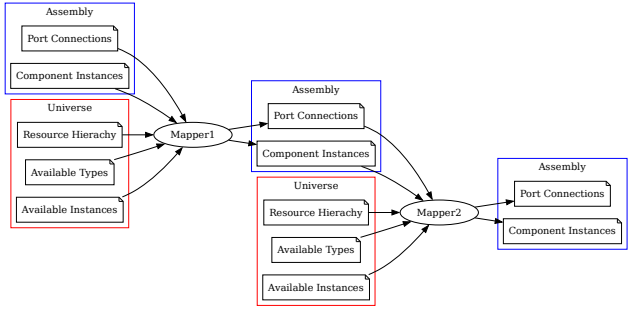
↪ Can be generated based on the state of the platform(s)

Mapper – Definition

Mapper

$$\text{Mapper} : \textit{Assembly} \times \textbf{Universe} \rightarrow \textit{Assembly}$$

Universe – Set of resource instances and types to complete assemblies



Mapper – Examples of Mappers

Warning!: We are **not** proposing new placement algorithms!

Simple Greedy Mapper

- for each unconnected `CONTAINED` ports
  - go through the set of resources instances and see if one can connect
  - if not, go through the set of resources types, and see if one can connect, if so instantiate this resource
- continue until no `CONTAINED` port is left unconnected (might need to backtrack)

Integration with Constraint Solvers (via Minizinc)

- Difficult to express our problem as a simple variant of knapsack/bin-packing ...
- A "flat" representation of the problem
- Have to "instantiate" the resource types to feed the solver: how many?

Mapper – Examples of Mappers

Warning!: We are **not** proposing new placement algorithms!

Simple Greedy Mapper

- for each unconnected `CONTAINED` ports
  - go through the set of resources instances and see if one can connect

Ok, we have a complete assembly, now what ?

- continue until no `CONTAINED` port is left unconnected (might need to backtrack)

Integration with Constraint Solvers (via Minizinc)

- Difficult to express our problem as a simple variant of knapsack/bin-packing ...
- A "flat" representation of the problem
- Have to "instantiate" the resource types to feed the solver: how many?

Assembly – Transformation into Concerto

We need a way to **execute/deploy** our assemblies

Assembly – Transformation into Concerto

We need a way to **execute/deploy** our assemblies

$\hookrightarrow$ We choose Concerto

Assembly – Transformation into Concerto

We need a way to **execute/deploy** our assemblies
$\hookrightarrow$ We choose Concerto

Why Concerto?

- Performance!
- and support for reconfiguration (cf. perspectives)

Transformation into Concerto

- We transform the Assembly in our model into a Concerto assembly
- All of our components have a Concerto implementation in Python
- (behavior support = "only `pushB(start)`")

Implementation / Evaluation

Implementation / Evaluation

- Realization of our Model in Nickel[1] (JSON + types + contracts + λ)
- Slight changes of Concerto (new port type, add parameters to components)
- Implemented several resource components (i.e., in our model and in Concerto)
  - Grid'5000 (basic env.), Kadeploy,
  - QEMU (`x86`, `arm64`), VM on Grid'5000,
  - Docker,
  - K3S cluster (control node + agents)
  - preallocated resource (e.g., laptop)

[1] `https://nickel-lang.org/`

Scenario – Gitea and CI Runners

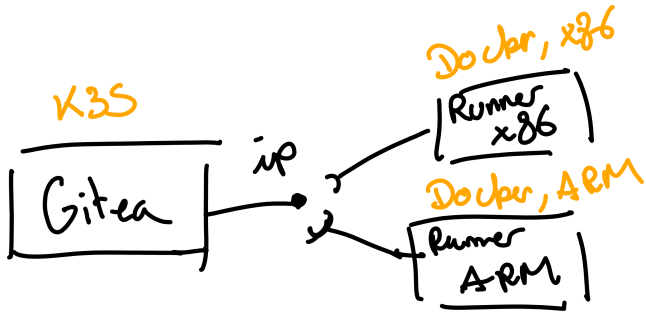
Assembly

- Gitea: Git hosting service
  - K3S (http, DB, storage)
- CI runners
  - Docker container
  - `x86` and `arm64`

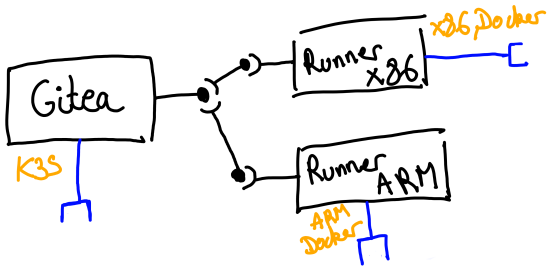
Scenario

Deploy the same assembly ...

- ... in production (Grid'5000)
- ... in "incomplete" prod.
- ... on my laptop (dev)

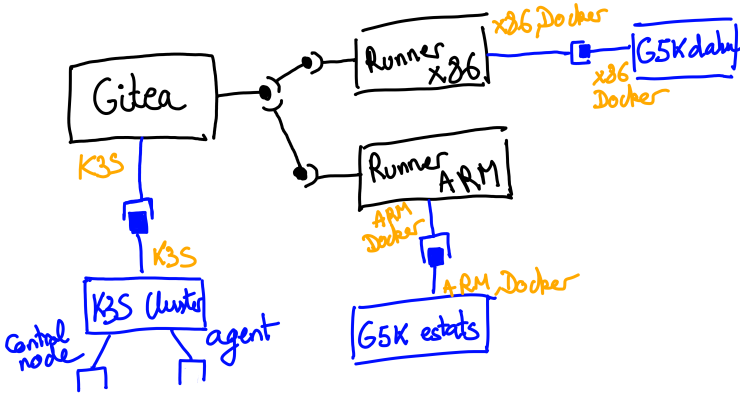


Scenario – Gitea and CI Runners – Production



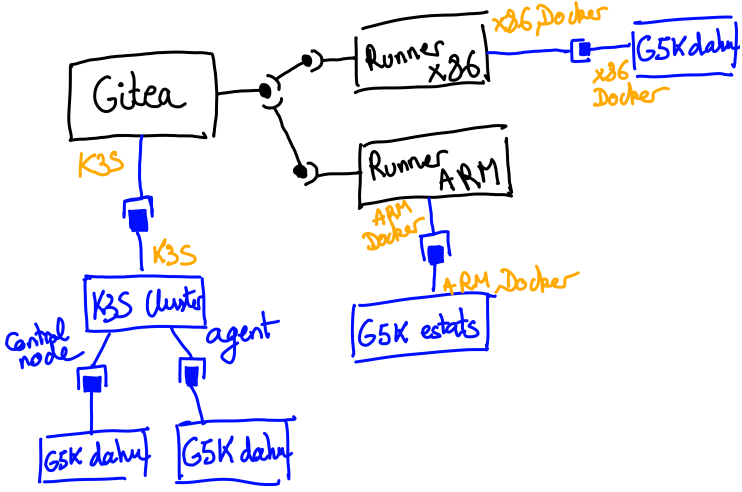
- Had to deploy own K3S cluster
- using one `x86` node (`dahu`) for runner
- using one `arm64` node (`estats`) for runner

Scenario – Gitea and CI Runners – Production



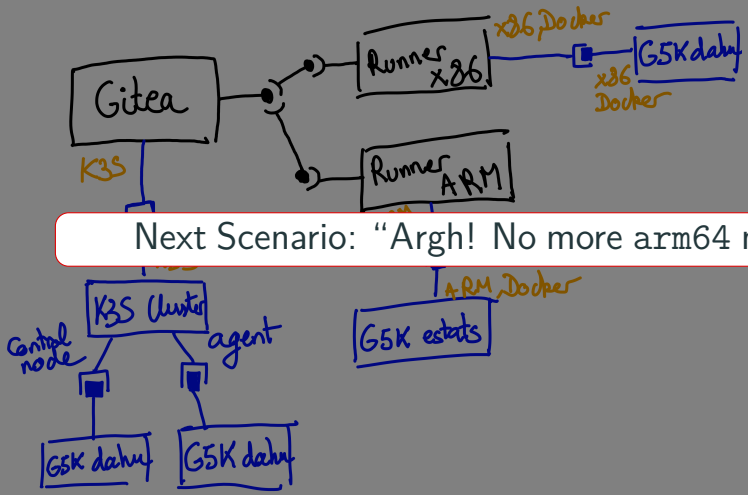
- Had to deploy own K3S cluster
- using one x86 node (`dahu`) for runner
- using one `arm64` node (`estats`) for runner

Scenario – Gitea and CI Runners – Production



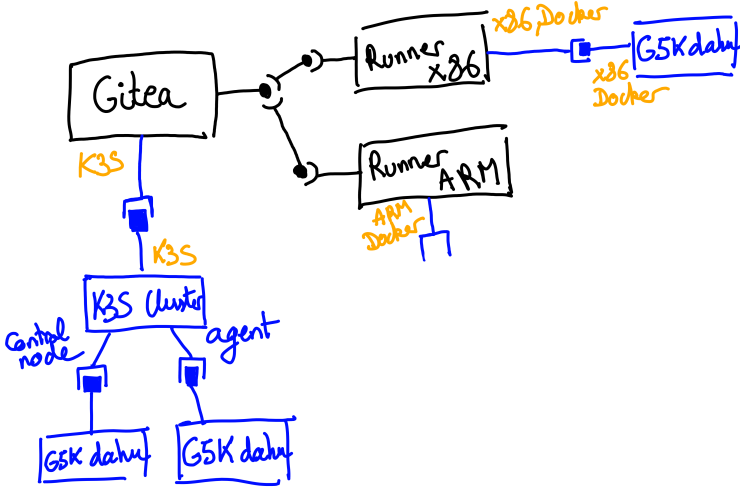
- Had to deploy own K3S cluster
- using one `x86` node (`dahu`) for runner
- using one `arm64` node (`estats`) for runner

Scenario – Gitea and CI Runners – Production



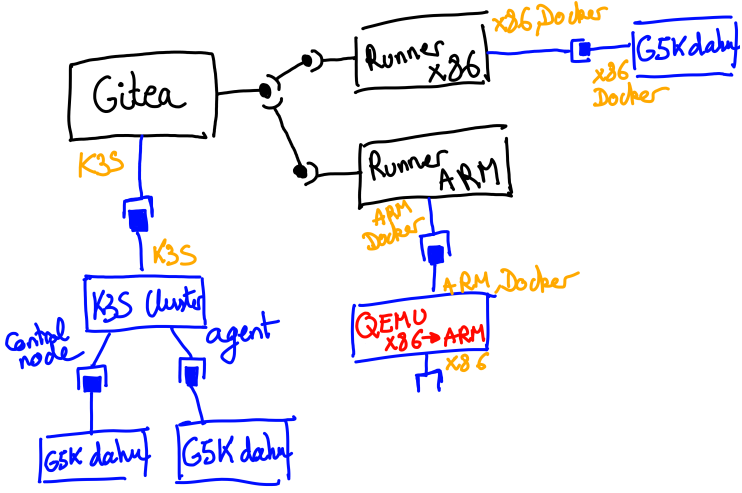
- Had to deploy own K3S cluster
- using one x86 node
- using one `arm64` node (`estats`) for runner

Scenario – Gitea and CI Runners – Incomplete Production



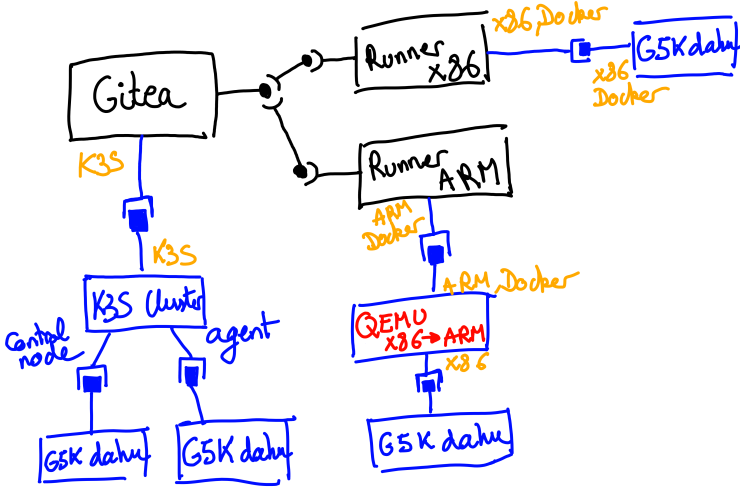
- no `arm64` node available!
- use a x86 node + **QEMU**
- "deep encapsulation"!

Scenario – Gitea and CI Runners – Incomplete Production



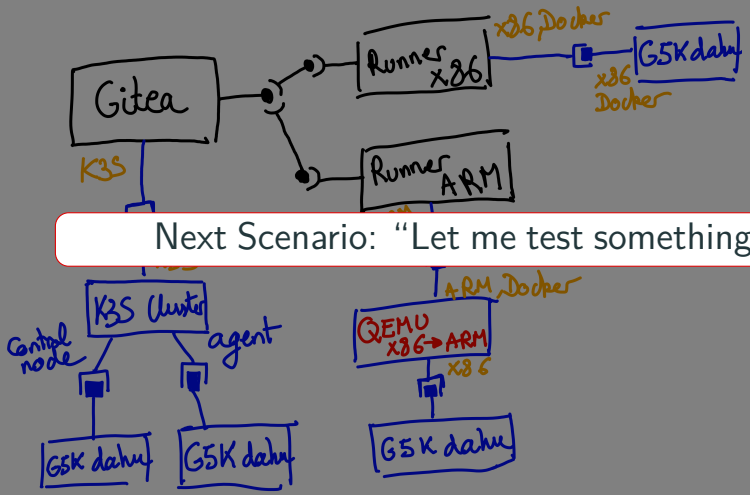
- no `arm64` node available!
- use a x86 node + **QEMU**
- “deep encapsulation”!

Scenario – Gitea and CI Runners – Incomplete Production



- no `arm64` node available!
- use a `x86` node + **QEMU**
- "deep encapsulation"!

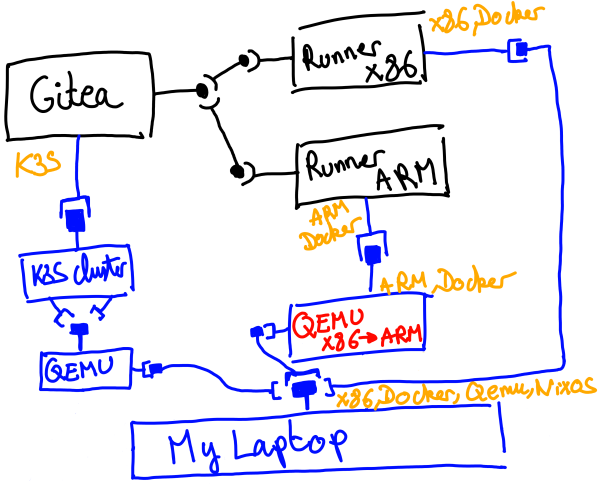
Scenario – Gitea and CI Runners – Incomplete Production



- no `arm64` node available!
- QEMU
- “deep encapsulation”!

Next Scenario: “Let me test something on my laptop”

Scenario – Gitea and CI Runners – Development



- need to reduce the resource requirements (functional attributes) to fit on laptop
- K3S needs Debian
- VM for K3S cluster and VM for arm worker

Conclusion & Perspectives

Conclusion & Perspectives

Pb: "How to model assemblies to support their execution in various contexts?"

Conclusion and Analysis

- ✓ Model to express application assemblies independently from the resources
- ✓ Allow for portability of assemblies, and heterogeneous deployments
- ✓ Able to model complex deep "encapsulation levels"
- ✓ Can generate problem for a solver, and transform the output into Concerto
- ≈ Resource components not (yet) too difficult to implement in Concerto
- ≈ Might be difficult for components to communicate if "too deep" in encapsulation

Conclusion & Perspectives

Pb: "How to model assemblies to support their execution in various contexts?"

Conclusion and Analysis

It kinda works!

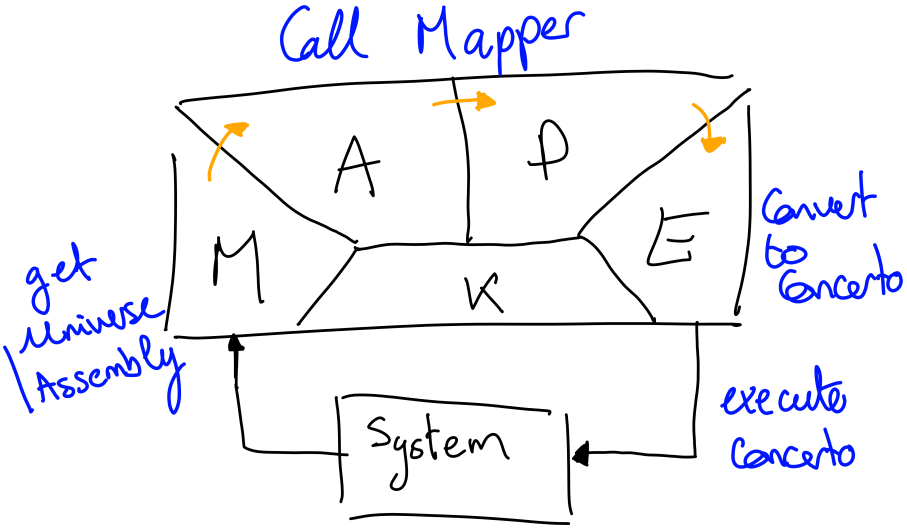
Perspectives

- **Reconfiguration!**
- Support variants of components, composites, auto-wiring (?)
- Software environments
- Network, storage elements as resources ?
- Allow the Mapper to change the parameters of the resource components

By making deployments easier/optimized, we also made **more** deployments possible...

$\hookrightarrow$ Are we participating in the acceleration of the Cloud?

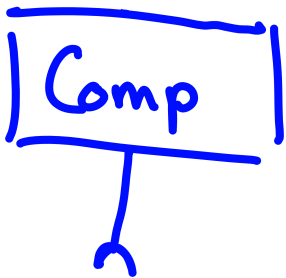
The Global Picture



Categories of Resource Components

Applicative Components

- Uses one, or more, resources
- Does not provide any resource

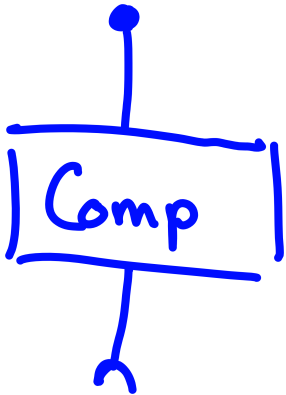


Categories of Resource Components

Applicative Components

Transformers/Converters

- "Transform" one resource into another one
- Use one resource, provide one resource
- e.g., Virtual Machines, Containers,



Categories of Resource Components

Applicative Components

Transformers/Converters

Resources Groups

- Gather several resources into a single interface
- e.g., Kubernetes

Categories of Resource Components

Applicative Components

Transformers/Converters

Resources Groups

Allocations

- Already allocated resources
- Resources acquired via allocation
- Only provides resource(s)
- e.g., local machine, Grid'5000, AWS