Geo-distribute Cloud applications for the edge

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

Inria
inventeurs du monde numérique

LS2N
LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

Marie Delavergne under the supervision of
Adrien Lebre and **Ronan-Alexandre Cherrueau**

Context

Existing cloud apps are going to the Edge

- **What is the Edge?**
  - Bring compute and storage equipments as close as possible to discrete data sources, physical elements or end users (to deal with latencies)
  - Limited network connections with disconnections between (edge) sites

Existing cloud apps are going to the Edge

- **How to deal with disconnections?**
  - Distribute cloud application instances on every involved sites
  - Each instance works **autonomously**, but should be able to ***collaborate*** with others when needed

Existing cloud apps are going to the Edge

- **Unfortunately, most Cloud applications do not follow these principles**
  - e-commerce, web-services, stream processing, …, OpenStack

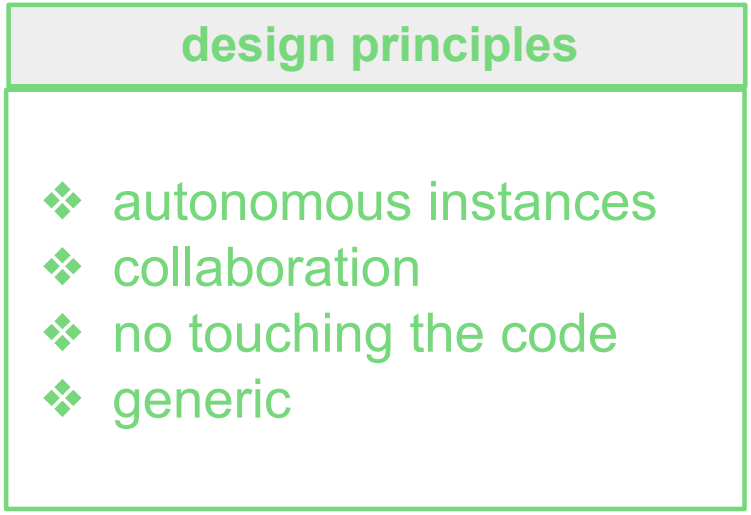
- **Intrusive modifications, when possible, are tedious** [1,2]
  - Thousands of LoCs: ShareLatex, Kubernetes, …, OpenStack

⇒ We do not want to change their code

[1] Revising OpenStack to Operate Fog/Edge Computing infrastructures https://hal.inria.fr/hal-01273427
[2] ShareLatex on the Edge [...] https://dl.acm.org/doi/10.1145/3286685.3286687

Problem

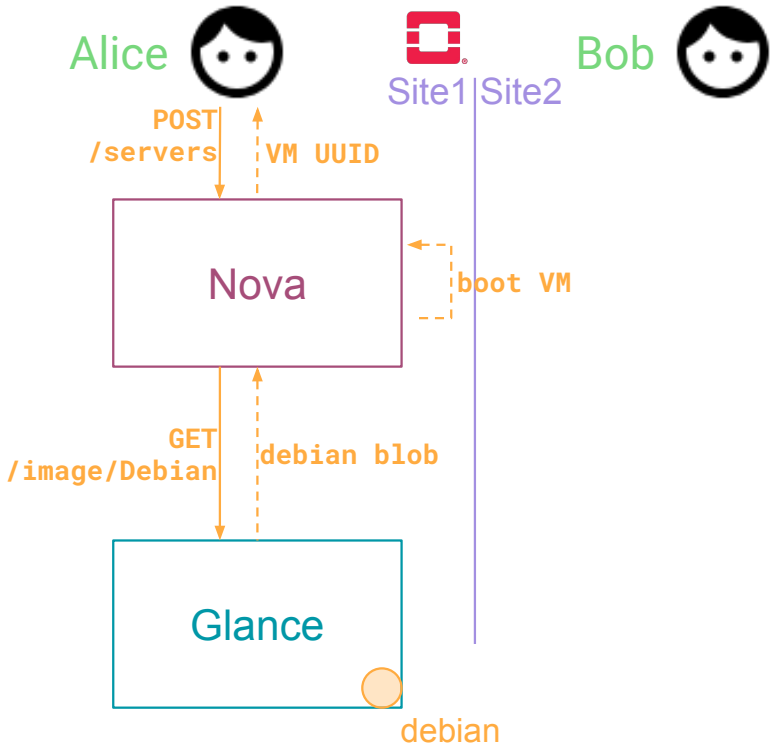
How to make a Cloud App edge compliant?



My Cloud application, example: OpenStack

Alice and Bob use the same Openstack, even though Bob is far

```
server a = openstack
           server create my-vm
           --image debian
```



How to make a cloud app edge compliant with our design principles?

- genericity ?
- no touching the code ?
- autonomous instances?
- collaboration ?

- The answer lies -in part- in service-oriented application modularity
- Those applications are composed of modules/services that:
  - allows separation of concerns (application domain vs deployment, monitoring, etc.)
  - are generic and can be used in other applications
  - expose an API to communicate with each other

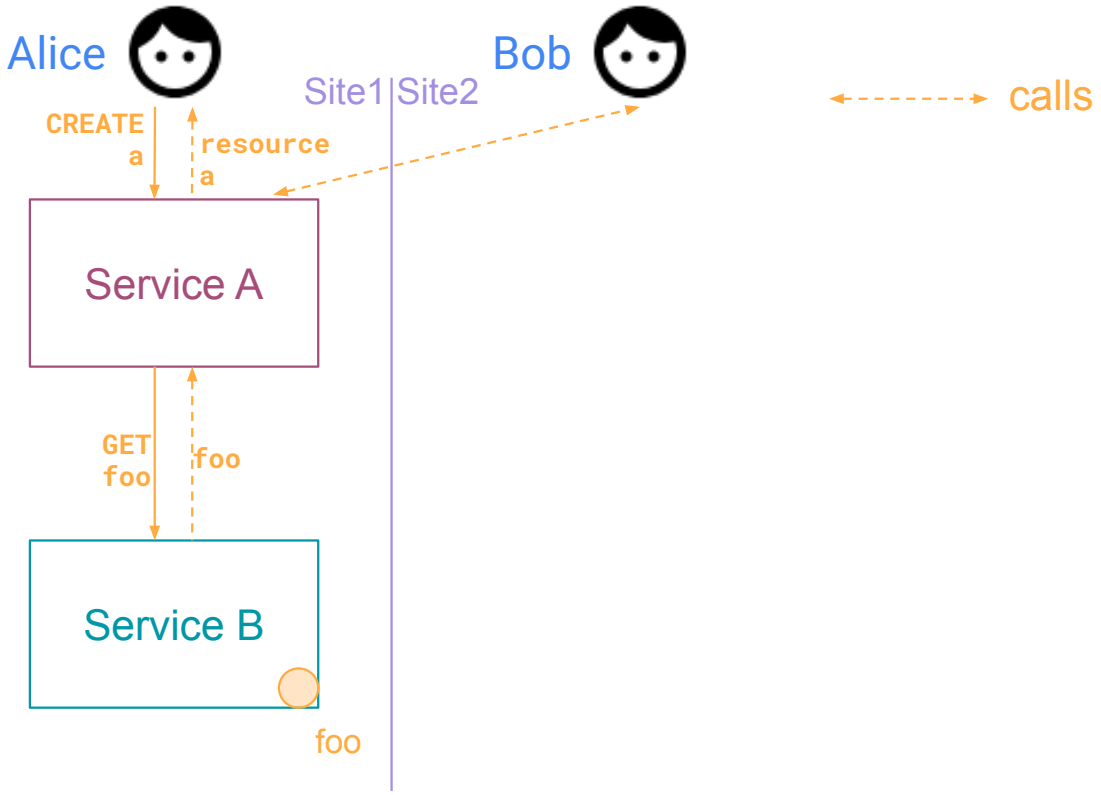
Solution

My Cloud application

- generic
- no touching the code
- autonomous instances ✖
- collaboration ✖

Alice and Bob use the same application, even though Bob is far

```
resourceA a = application
              resourceA create
              --sub-resourceB foo
```

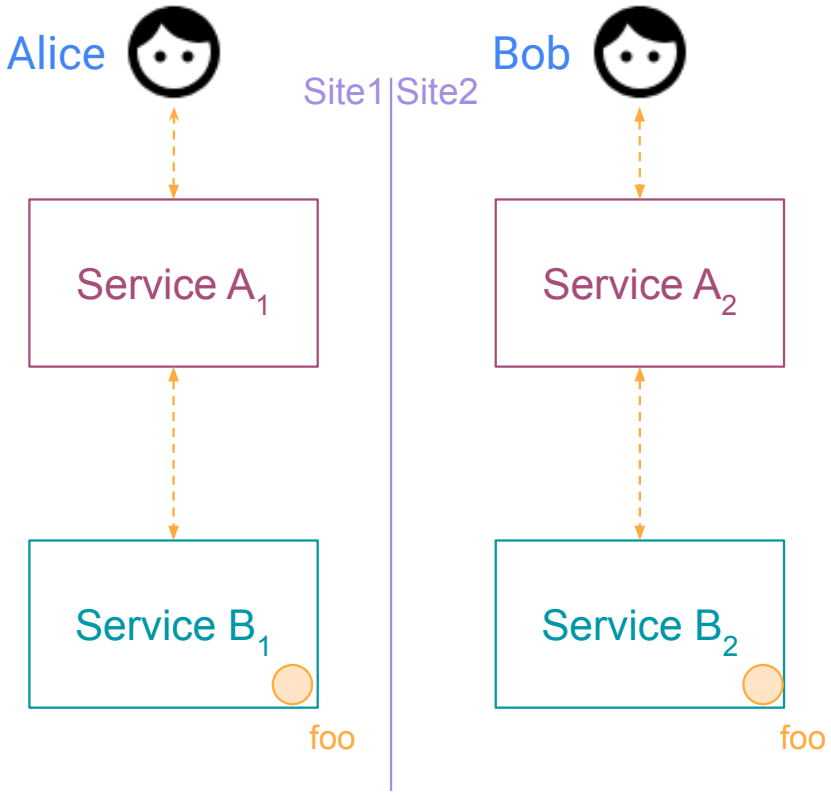


My Cloud application *instantiated everywhere*

- generic ☐
- no touching the code ☐
- autonomous instances ☐
- collaboration ✖

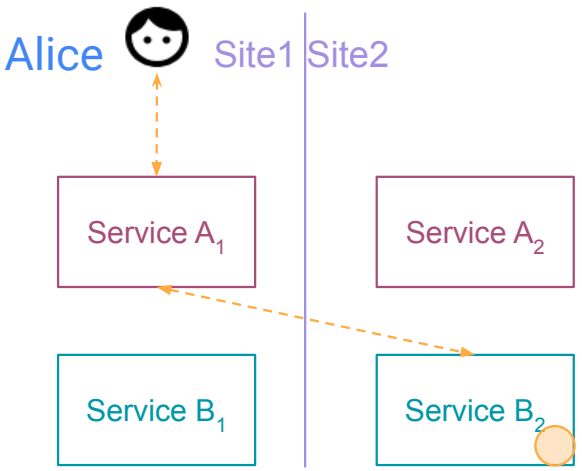
Alice and Bob use their own application, closer to them

```
resourceA a = application
              serviceA create
              --sub-resourceB foo
```

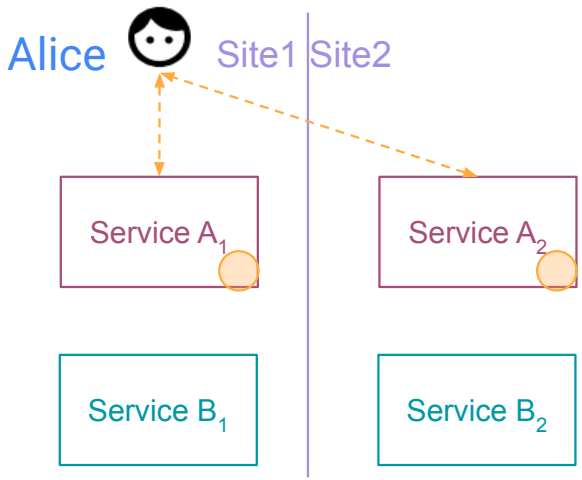


Focus on collaboration (3 types)

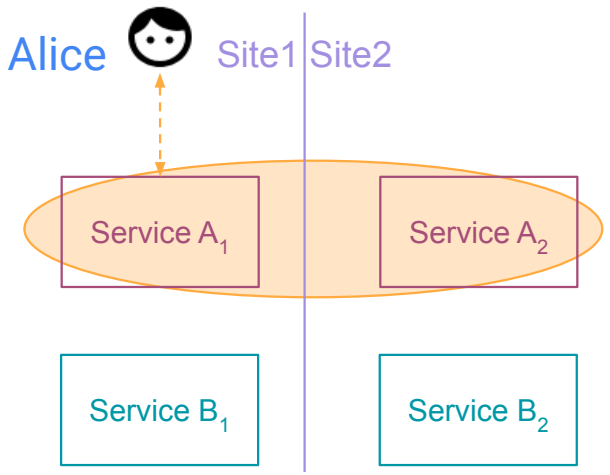
- Between services of different instances for **sharing**
- Resource **replication**
- Resource spanning a**cross** different instances



Sharing:
service B from Site2 has the required resource



Replication:
Alice creates identical resources on different sites



Cross:
Alice creates a resource that span on every involved sites

Design principles, updated

- genericity
- no touching the code
- autonomous instances
- collaboration
  - sharing
  - replication
  - cross

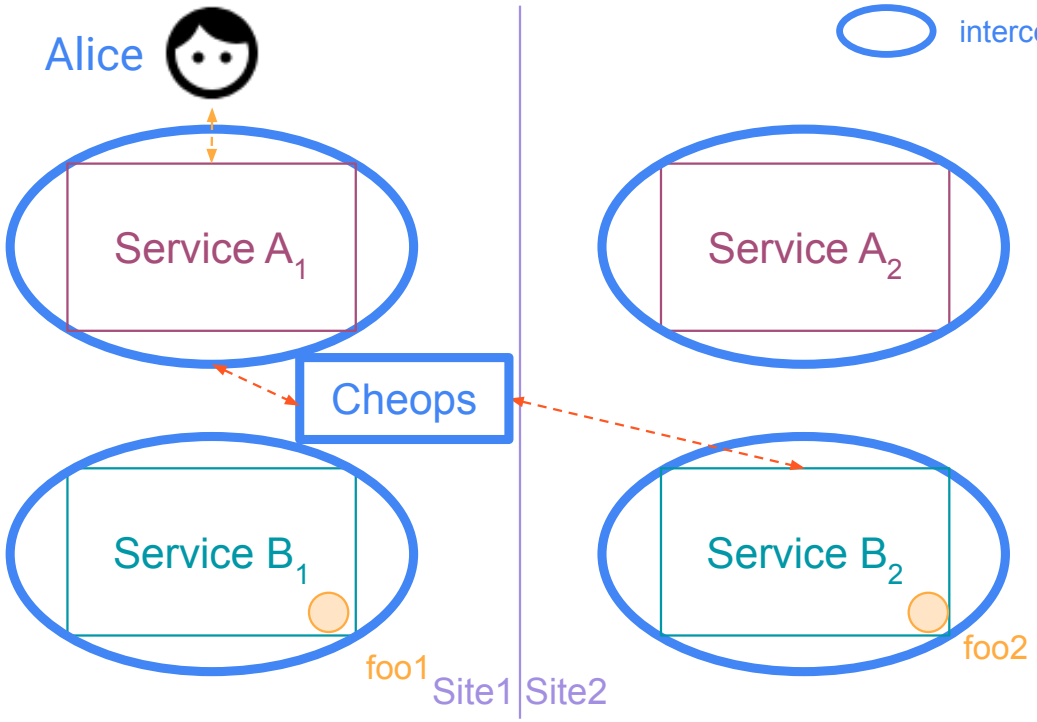
My cloud application *with sharing*

- generic
- no touching the code
- autonomous instances
- collaboration
  - sharing
  - replication ✖
  - cross ✖

Alice defines the scope of the request into the CLI. The scope specifies **where** the request applies.

```
resourceA a = application
              resourceA create
              --sub-resourceB foo2
              --scope { serviceA: Site1 ,
                        serviceB: Site2 }
```

Cheops: yet another proxy? Yes and no.[1]



[1] Edge Computing Resource Management System: Two Years Later! https://hal.inria.fr/hal-02527366/

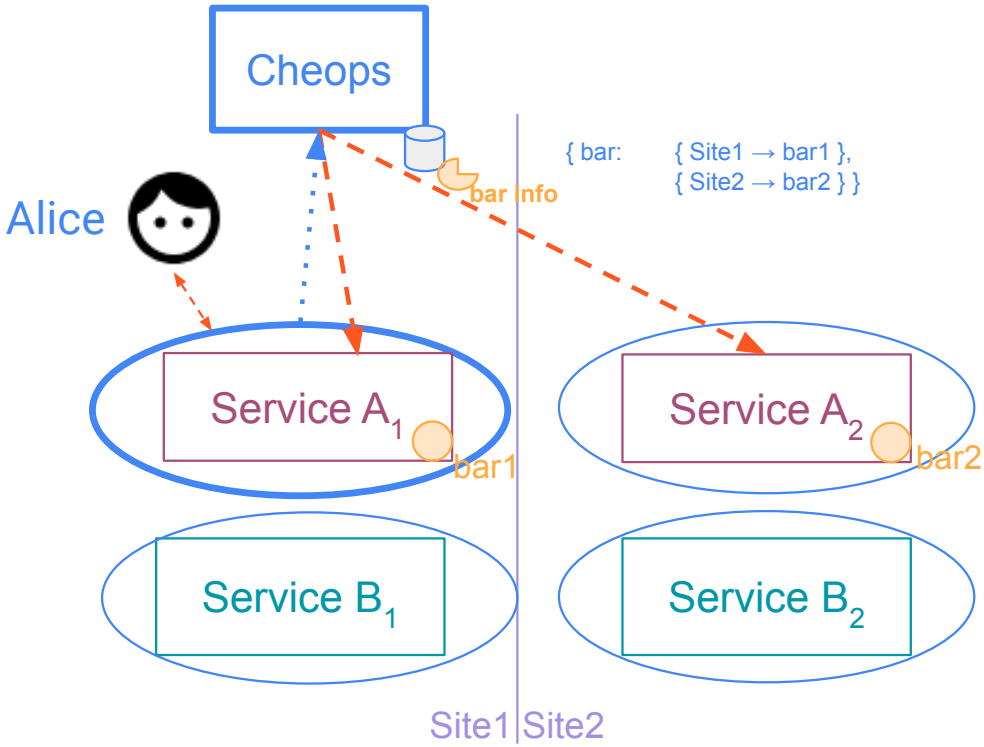
My cloud application *with replication*

- generic
- no touching the code
- autonomous instances
- collaboration
  - sharing
  - replication ?
  - cross ✖

Alice defines the scope of the request into the CLI. She defines that the resource (managed by Service A) will be created on both sites.

```
ressourceA bar  = application
                  resourceA create
                  --scope { serviceA: Site1 &
                                      Site2 }
```

- Stores only generic information about the resources (e.g. its unique id, where is it located, information to retrieve it locally)
- resource: {meta-uid, site-uid, local-uid}

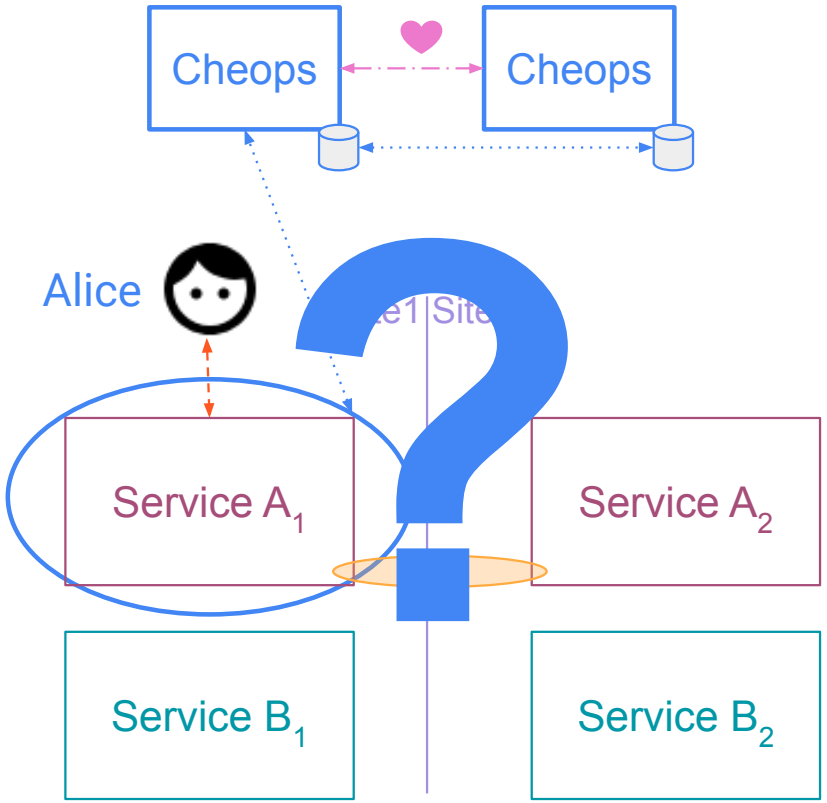


My cloud application *with cross*

- generic
- no touching the code
- autonomous instances
- collaboration
  - sharing
  - replication
  - cross

Done for Neutron[1] but requires additional work to be generic to any kind of service.

Future work

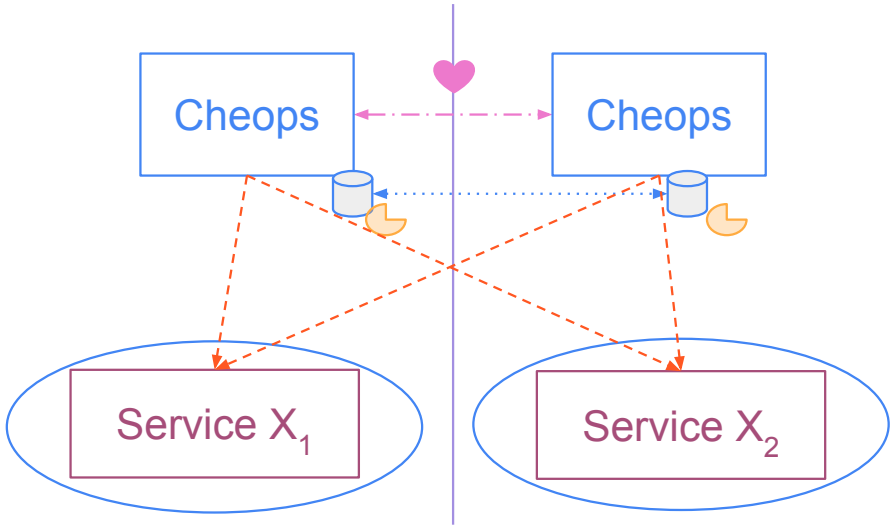


[1] Multi-site Connectivity for Edge Infrastructures : DIMINET: DIstributed Module for Inter-site NETworking

Cheops as a building block to deal with geo-distribution

- To forward requests between services
- To manage creations, updates and deletions on multiple sites

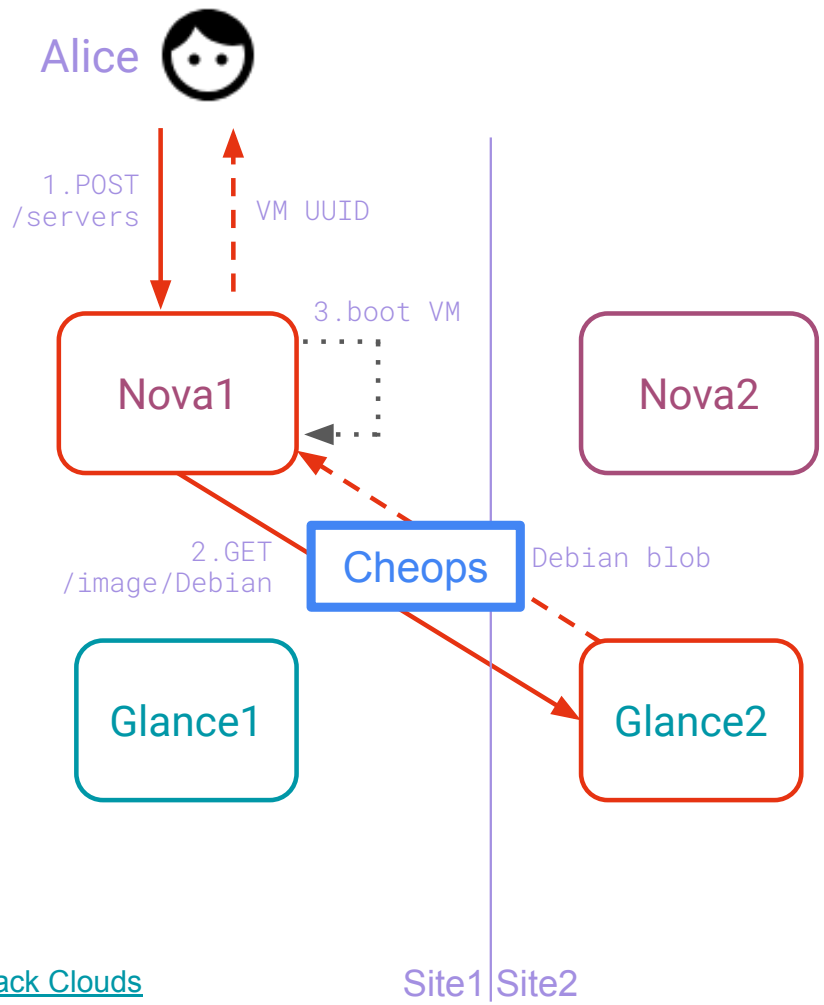
- Cheops intercepts API requests (services are black boxes)
  - Agents are located on each site
  - Relies on the forward operation
  - Uses heartbeat to check if sites are up and in the network
  - Uses a geo-distributed like database to have resource information only where relevant
  - A PoC on top of the service mesh Consul and Envoy (early stage of development)



Going further

OpenStack/*-Oïd: a first Poc

- Presented at the Open Infrastructure Summit in Denver[1]
- Laid the groundwork of a modular way of geo-distributing with *scope-lang*
- Works with the Nova and Glance example (*server create*)
- Uses HAProxy to intercept requests and Lua code to extract the scope



[1] Implementing Localization into OpenStack CLI for a Free Collaboration of Edge OpenStack Clouds

Cheops, for a fine-grained control

- **Vanilla request**
  - `openstack server create my-vm --image debian`
- **The same, with scope**
  - `openstack server create my-vm --image debian --scope { compute: Site1, image: Site1 }`
- **Sharing**
  - `openstack server create my-vm --image debian --scope { compute: Site1, image: Site2 }`
- **Replication**
  - `openstack image create debian --file ./debian .qcow2 --scope {image: Site1 & Site2}`
- Extend to any kind of multi-sites operations
  - **otherwise** operator, **around** operator
    - `server create --scope { Nova: Site1 ; Site2 }`
    - `server create --scope { Nova: around(Site1, 10ms) }`

Thanks for your attention

marie.delavergne@inria.fr

http://stack.imt-atlantique.fr