Cheops: Can A "Service-Mesh" Be The Right Solution For The Edge?

Geo Johns Antony
Marie Delavergne
Baptiste Jonglez
Adrien Lebre

Cheops Public repo: https://gitlab.inria.fr/discovery/cheops

Context

Existing Cloud Apps Are Going To The Edge

Deployment of micro and nano data centers at the Edge is taking off.

Challenges for applications:

- ○ High latency
- ○ Disconnection between (edge) sites

Existing Cloud Apps Are Going To The Edge

Intrusive modifications for existing Cloud Apps, when possible, are tedious: [1,2]

Thousands of LoCs: ShareLatex, Kubernetes, OpenStack, etc.

⇒ We do not want to change their code

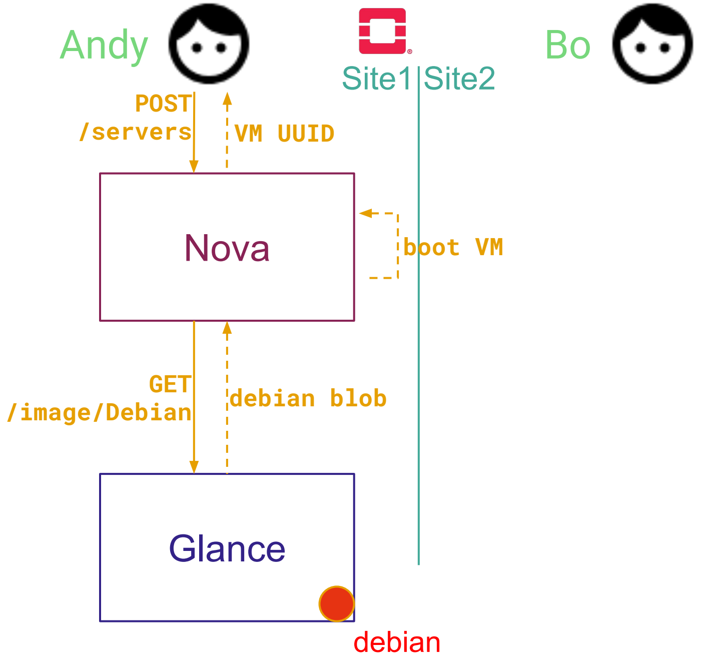
[1] Revising OpenStack to Operate Fog/Edge Computing infrastructures https://hal.inria.fr/hal-01273427
[2] ShareLatex on the Edge [...] https://dl.acm.org/doi/10.1145/3286685.3286687

Problem overview

Cloud Application Example: Openstack

Andy and Bo use the same Openstack, even though Bo may be far

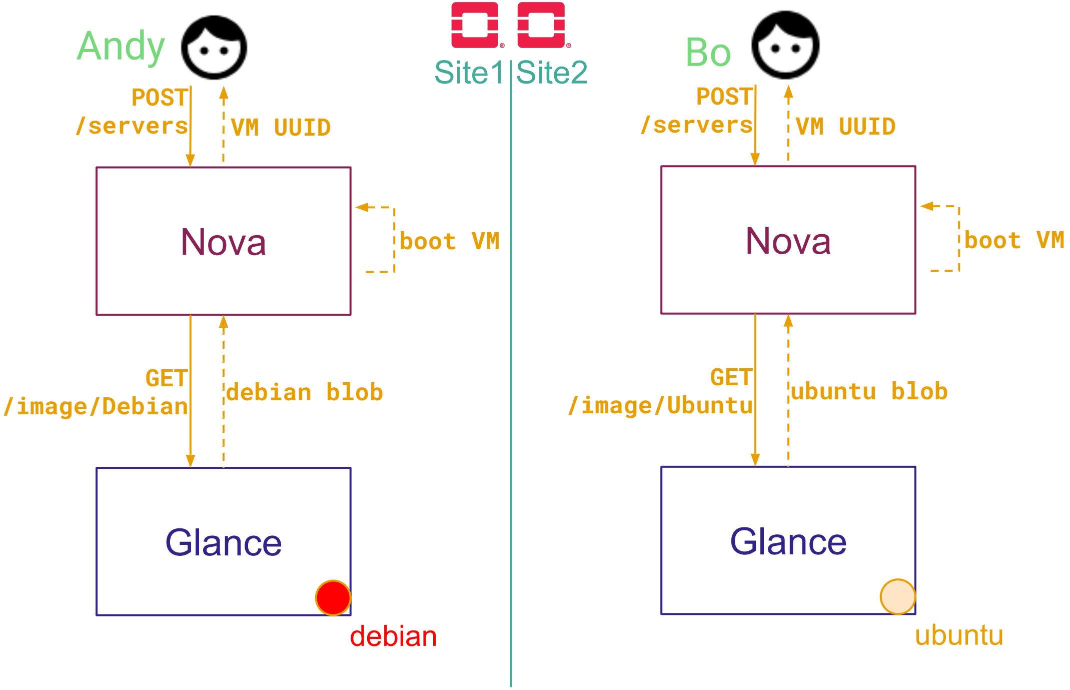
```
server a = openstack
           server create my-vm
           --image debian
```



Cloud Application Example: Openstack

Autonomous instances: provides **locality** and **robustness**

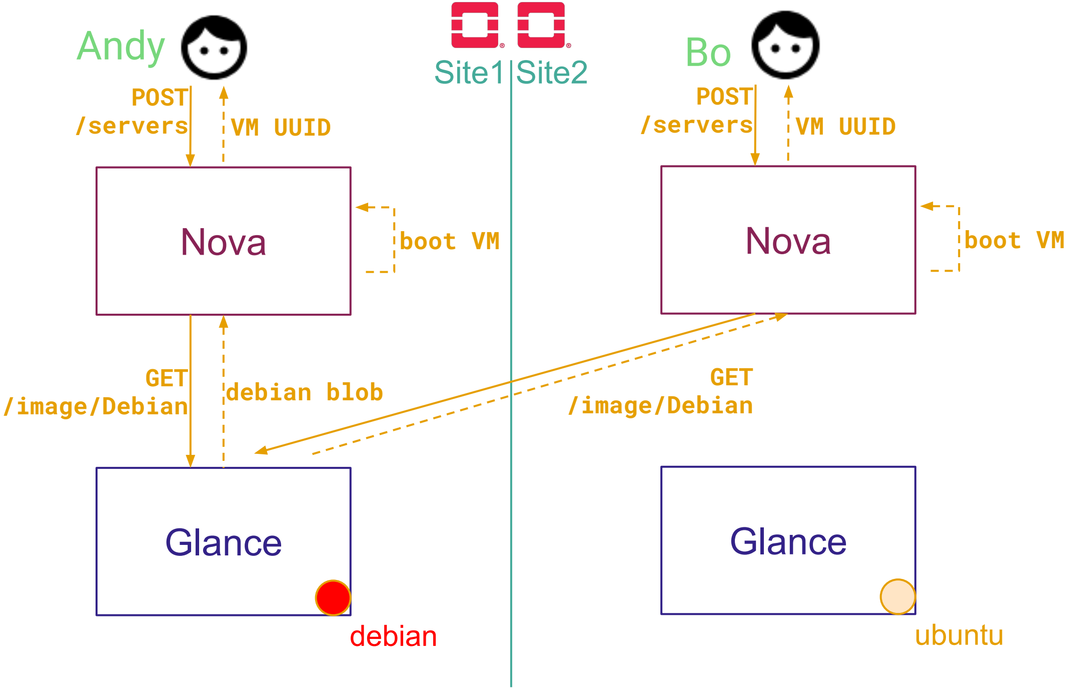
```
server a = openstack
           server create my-vm
           --image debian

server b = openstack
           server create my-vm
           --image ubuntu
```



Cloud Application Example: Openstack

Collaboration may be required

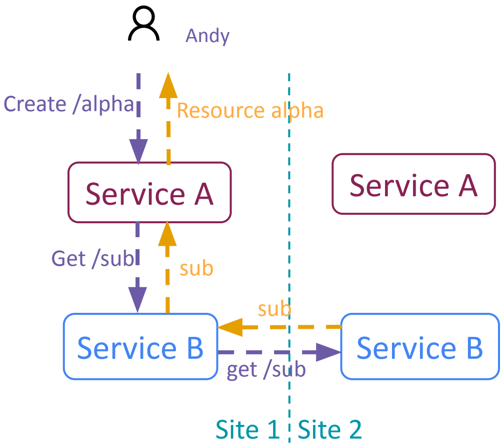
```
server b = openstack
           server create my-vm
           --image ?
```



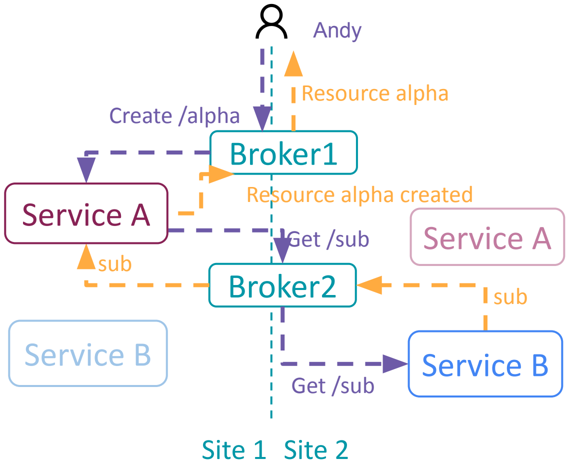
Collaboration For Service-Based Applications: state of the art

Request: Andy wants to create a resource alpha in site 1 using a resource foo from site 2

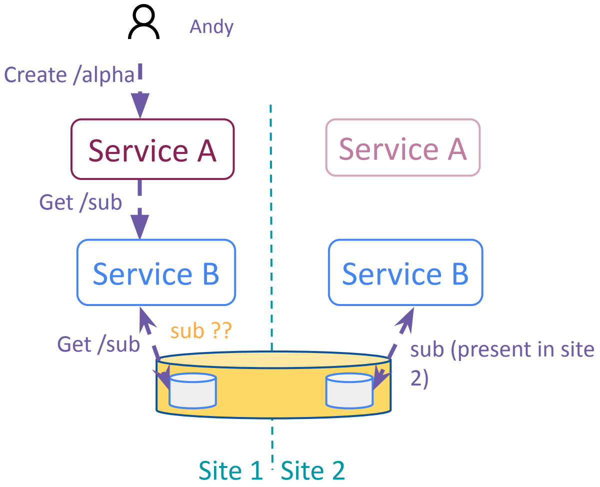
Service to Service Collaboration



Broker Based Collaboration



Database Collaboration



Problems:

The collaboration code is tangled in the business code

Broker needs to be developed for each service

Resources have a context and a scope, sometimes side effects

Approach envisioned and presented in OpenStack Summit in Vancouver 2018

Principles Of A Service-Based App For The Edge

- **Autonomous instances**: local-first for robustness
- **Collaboration** (on demand/if needed): leverage available resources
- **Generic:** the approach should work with multiple applications
- **No touching the code**: no extra efforts (intrusive) to existing code

Can A Service Mesh Be The Solution?

Premises Of Our Proposal

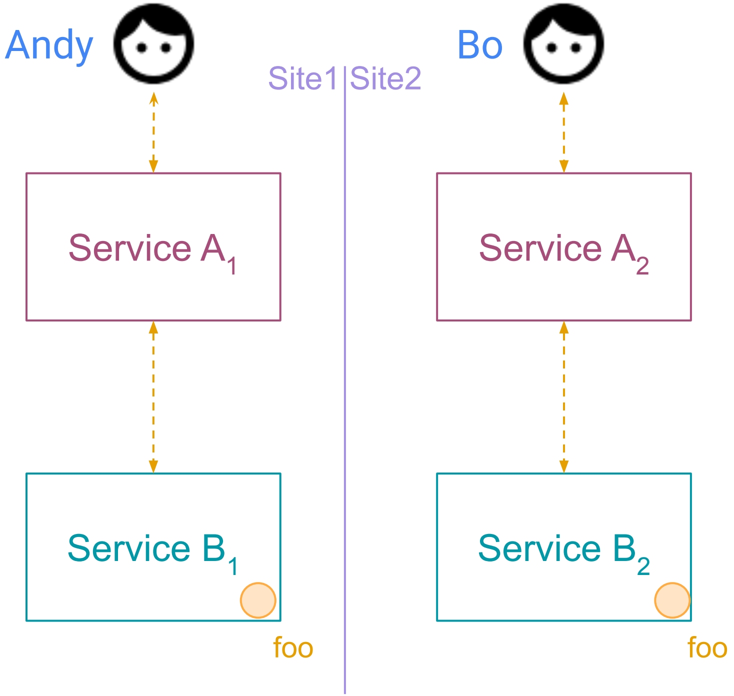
Same Cloud Application *Instantiated Everywhere*

Andy and Bo use their own application, closer to them

Ensuring **Local-first** principle

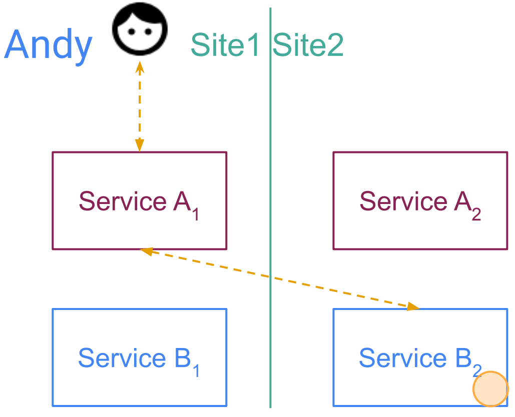
- Always able to serve local requests
- Minimize communications between DCs

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



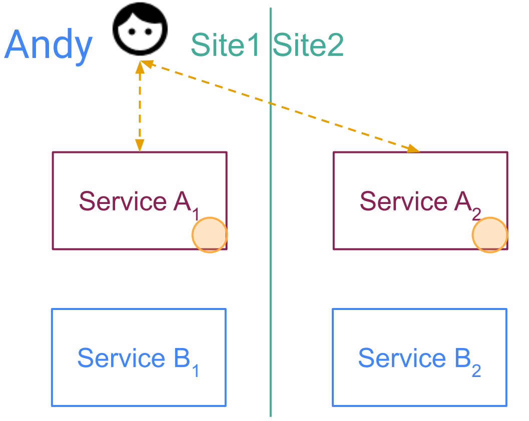
Focus On Collaboration (3 Types)

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



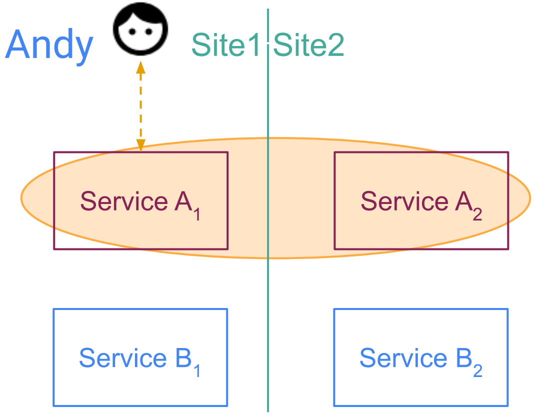
Sharing:

service B from Site2 has the required resource



Replication:

Andy creates identical resources on different sites



Cross:

Andy creates a resource that span on every involved sites

My Cloud Application *With Sharing*

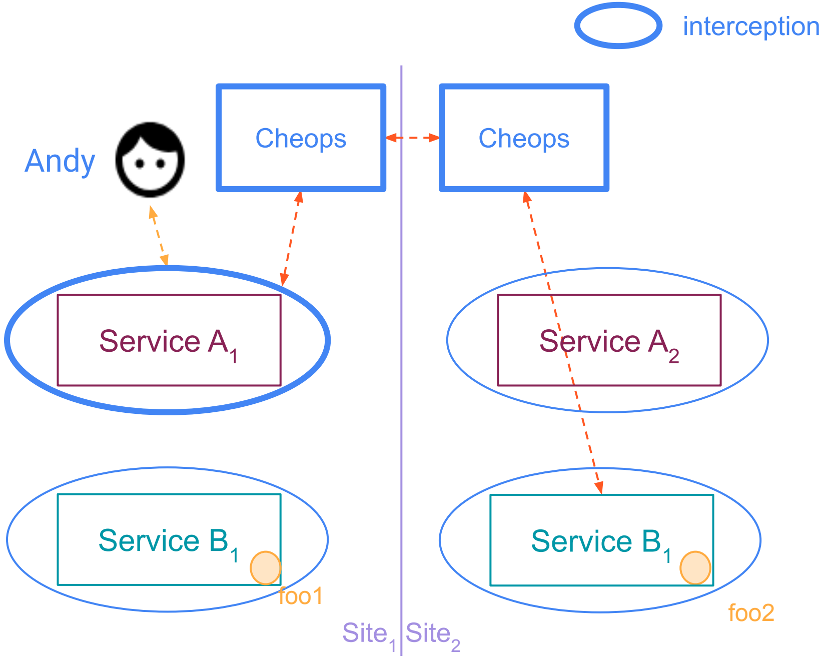
Service B from Site2 has the required resource

Andy 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 }
```

Sharing allows the user to share a resource which is located on a remote geo-located site.

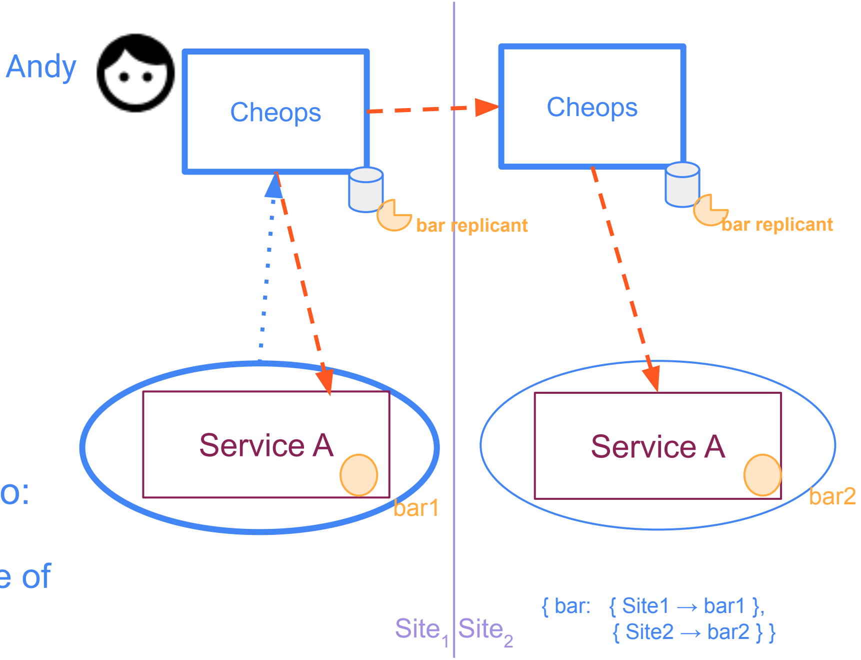
- Create a resource using a service from a remote location
- Provides flexibility for services to be geo-distributed rather than each site having all required services (to create a resource)



My Cloud Application *With Replication*

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

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



Cheops allows to replicate resources on different sites to:

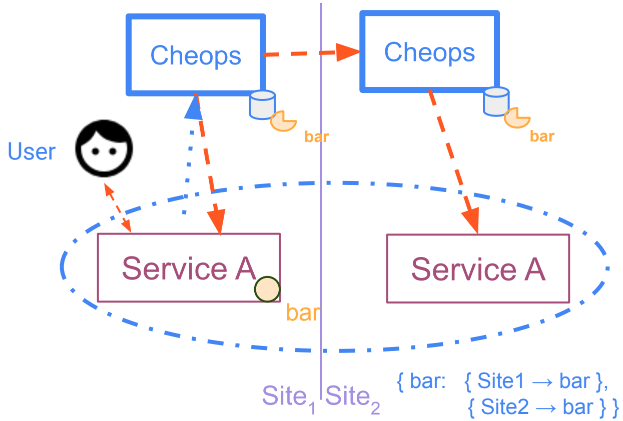
- lower latency when using a resource close
- increase robustness towards partition by allowing the use of the resource locally

The replication follows an eventual consistency, and the consistency is maintained **through the API**

My Cloud Application *With Cross*

Cross creates the illusion of a **Single Service Resource** across the involved sites.

- Service is made available across geo-distributed sites **without creating Independent instances**.
- A Single instance of an application is created throughout the geo-distributed sites.
- Cross will manage the availability, Network Partition, local-first scenarios.
- Cross is based on **Aggregation** & **Divisibility** principles we identified



Initial proposal made by Sarmiento et al. (STACK team).

- Development efforts for Openstack was made

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

Collaborations At The Devops Level

Scope-lang[1] :

- A DSL to manage resources & integrated with native CLI
- Irrespective to any platform
- A scope-lang expression contains
  - Collaboration & service information
  - Location Information

Scope-lang expression:

- `resourceA a = application serviceA create --sub-resourceB foo –scope {ServiceA: Site1, ServiceB: Site2}`
  - `openstack server create [...] myvm --scope {Nova: Berlin & Paris}`
  - `Kubectl apply -f nginx.yml --scope {Berlin & Paris}`

[1] https://hal.inria.fr/hal-03212421/

Collaborations and Scope-lang

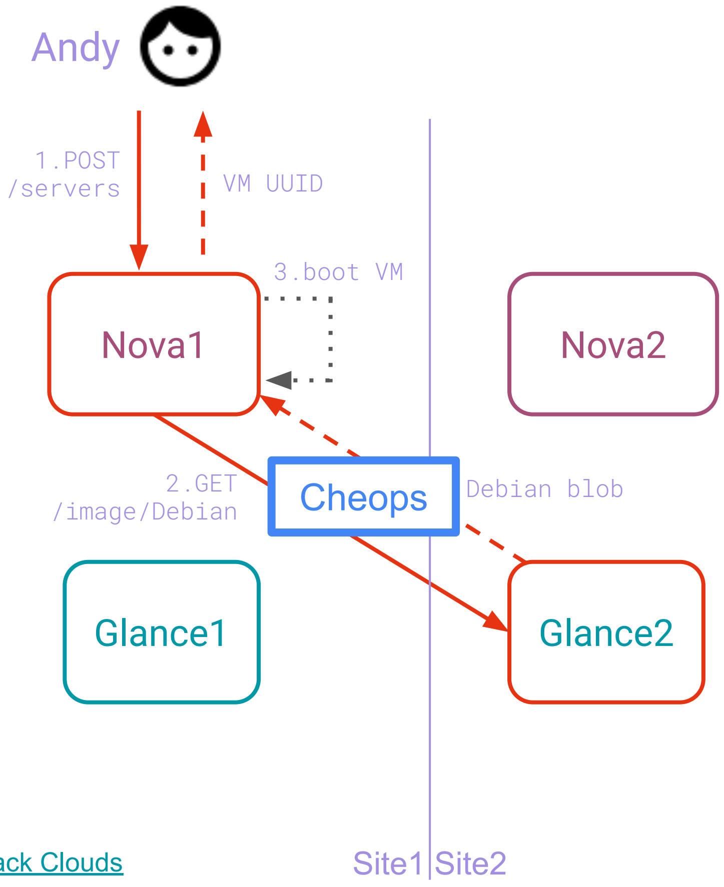
- Sharing
  - `application resourceA create --sub-rscB foo --scope { serviceA: Site1, serviceB: Site2 }`
- Replication
  - `application resourceA create --scope { serviceA: Site1 & Site2 }`
- Cross
  - `application resourceA create --scope { serviceA: Site1 % Site2,prime: Site1 }`
- Or
  - `application resourceA create --scope { serviceA: Site1 || Site2 }`
- Around
  - `application resourceA create --scope { around: Site1, 50ms }`

Berlin Hackathon

Back to 2018

OpenStack/*-Oïd: A First PoC

- Implemented during the hackathon in berlin and then presented for the first time 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

A Resource Is Not Just A Black Box

How To Deal With Dependencies Between Resources?

A Generic Pattern

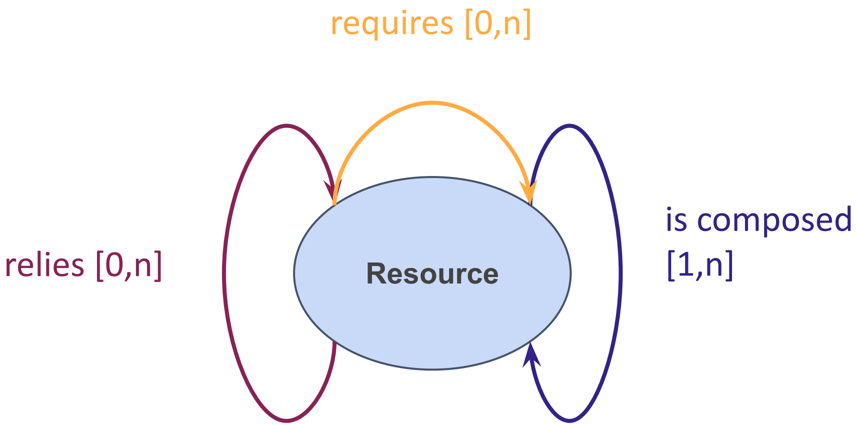
An application consists of a vast heterogeneous micro services.

Microservices relies on resources like VM, Pod, Services, Networks, etc.

Geo-distributing applications without explicit change in code:

- Involves handling dependencies between these micro-services
- Since micros-services are distributed, dependencies from multiple platforms need to be handled
- Can we find a generic pattern across multiple frameworks to solve these **dependencies**?

Patterns In Resources



Example:

A vm **requires** an image

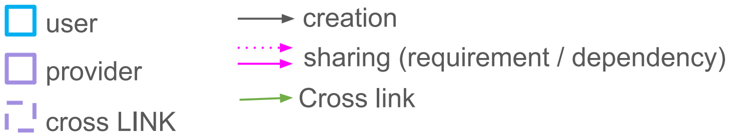
Example:

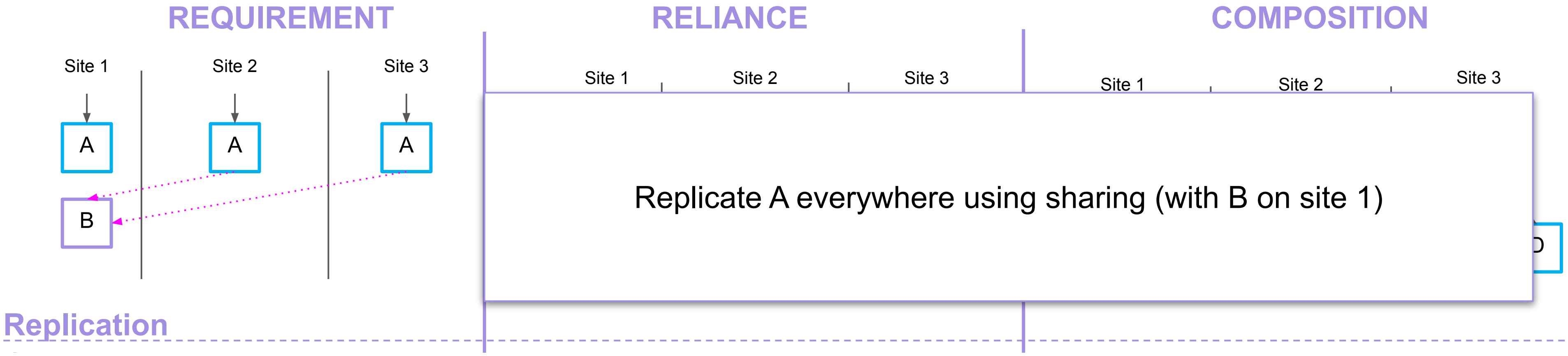
A vm **relies** an IP

Example:

A deployment **is composed of** pods

Behaviors / Patterns





Replicate A everywhere using sharing (with B on site 1)

Behaviors / Patterns

□ user
□ provider
▭ cross LINK

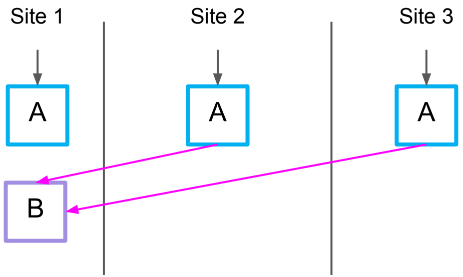
→ creation
→ sharing (requirement / dependency)
→ Cross link

REQUIREMENT

Site 1 Site 2 Site 3

Replicate A everywhere using sharing (with B on site 1).
A heartbeat will warn the user that B is no longer available for the other sites.

RELIANCE



COMPOSITION

Site 1 Site 2 Site 3

B D

Store the dependency information on B to warn against/prevent deletion.

Replication

Behaviors / Patterns

□ user

□ provider

⬚ cross LINK

→ creation

→ sharing (requirement / dependency)

→ Cross link

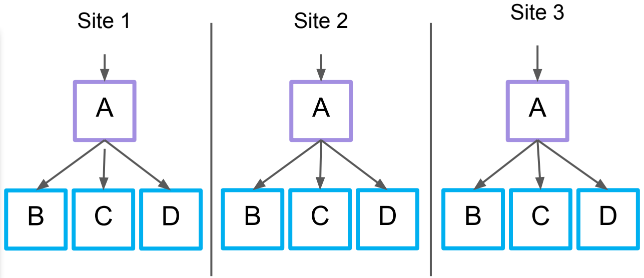
REQUIREMENT

Site 1 | Site 2 | Site 3

RELIANCE

Site 1 | Site 2 | Site 3

COMPOSITION



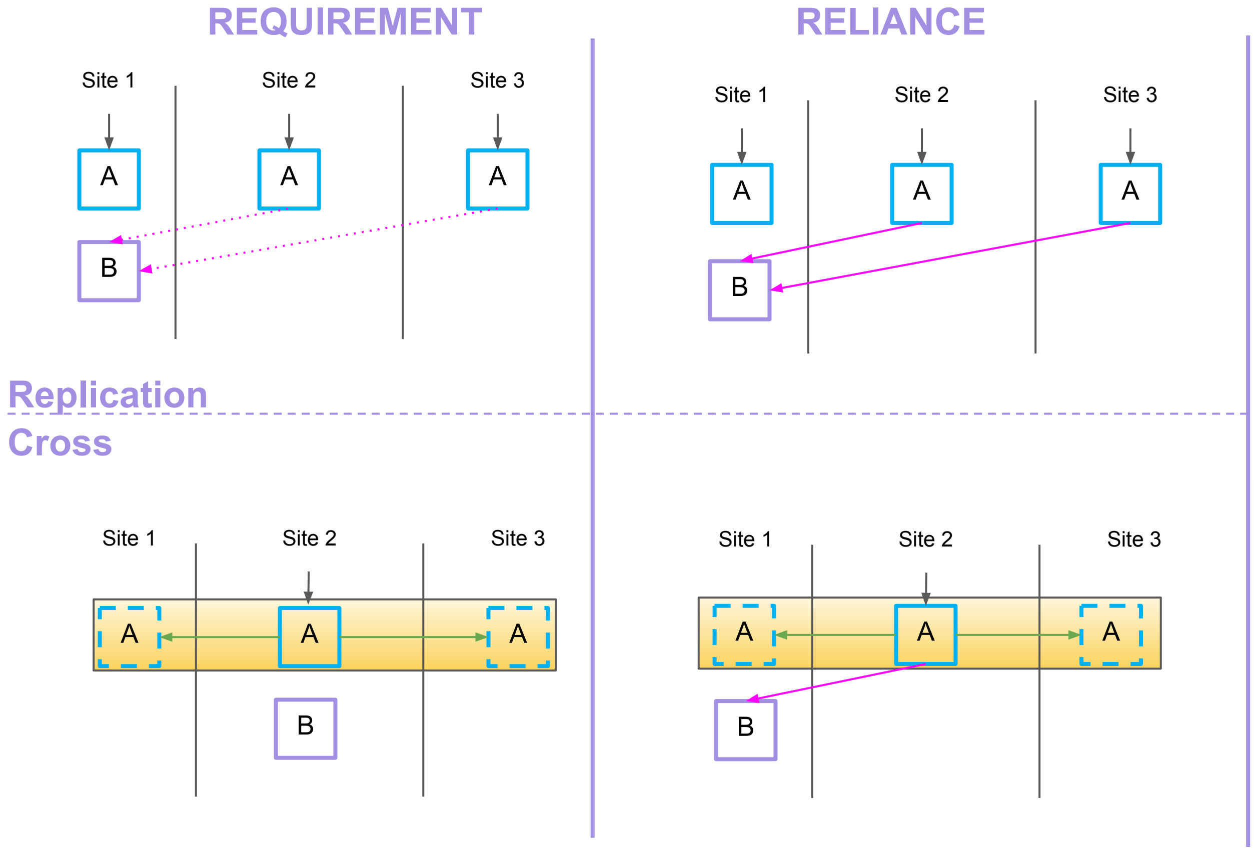
Replicate A everywhere, as an elementary resource, the application will create the sub-resource in the process

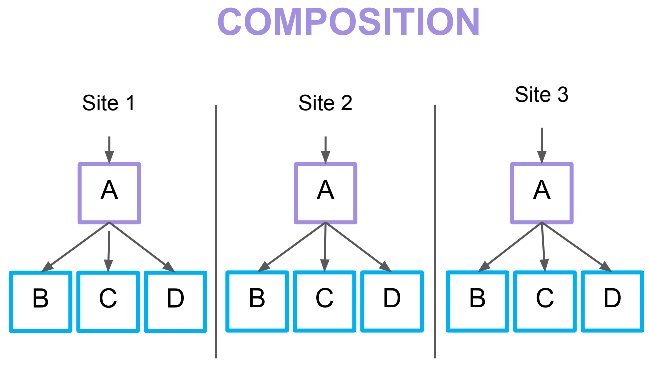
Replication

Behaviors / Patterns

- □ user
- □ provider
- ⬚ cross LINK

- → creation
- ⇢ sharing (requirement / dependency)
- → Cross link





Extend Resource A to new site (with direct link to resource A from site2)

Behaviors / Patterns

□ user

□ provider

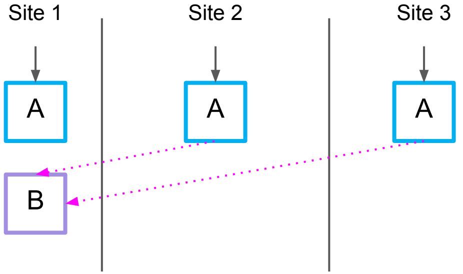
⬚ cross LINK

→ creation

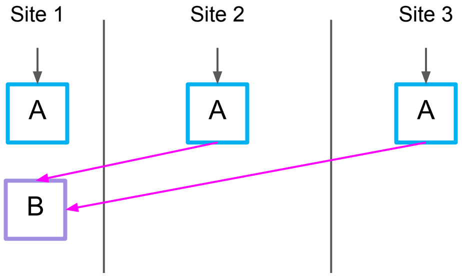
→ sharing (requirement / dependency)

→ Cross link

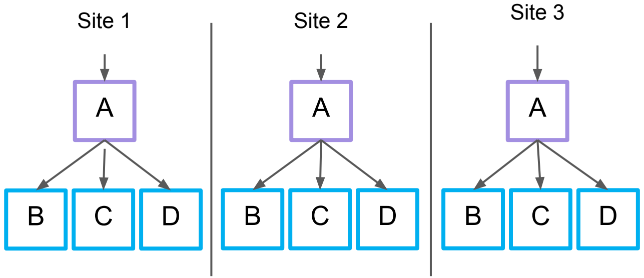
REQUIREMENT



RELIANCE



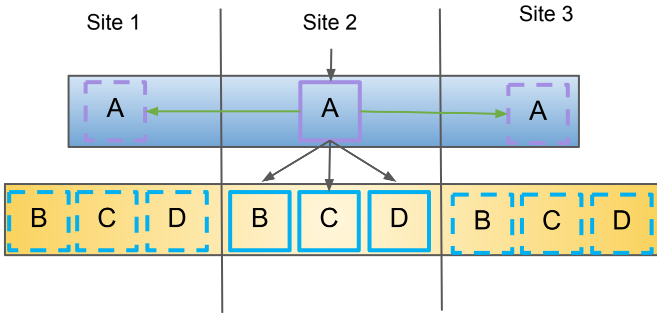
COMPOSITION



Replication

Cross

Extend the sub resources of Resource A into Different sites & create a cross link between them



Behaviors / Patterns

□ user

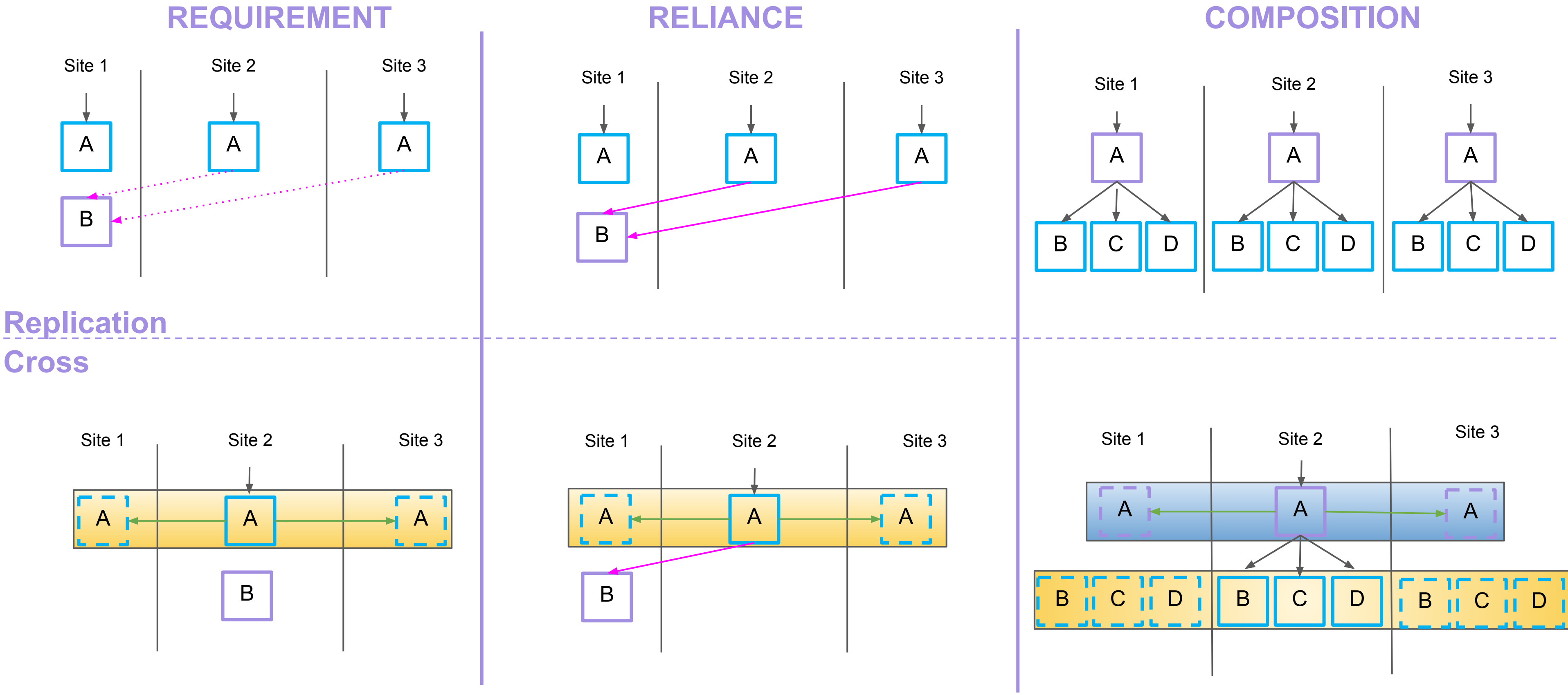
□ provider

⬚ cross LINK

→ creation

⇢ → sharing (requirement / dependency)

→ Cross link



REQUIREMENT

RELIANCE

COMPOSITION

Replication

Cross

Current Prototype

Cheops: A Generic Approach For Applications

A New Architecture

Our specification/requirements for the architecture:

- A generic resource handling across platforms
- Decentralized & P2P
- Modular and Scalable
- Ability to integrate our collaboration methods

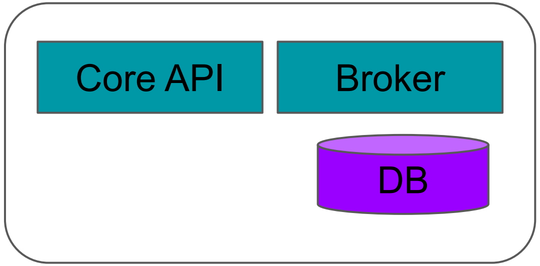
Primary target platforms: Openstack, Kubernetes

Surveyed multiple existing architectures, no suitable candidate found

Our Framework

Divided into 2 major modules:

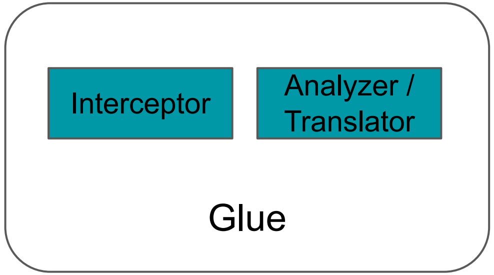
- Cheops Core:
  - Management layer responsible for the deployed application metadata & p2p interaction
  - Generic across platform
- Cheops Glue:
  - Intermediate layer responsible for translating interactions with the platform
  - Dependent to a platform



Core API

Broker

DB

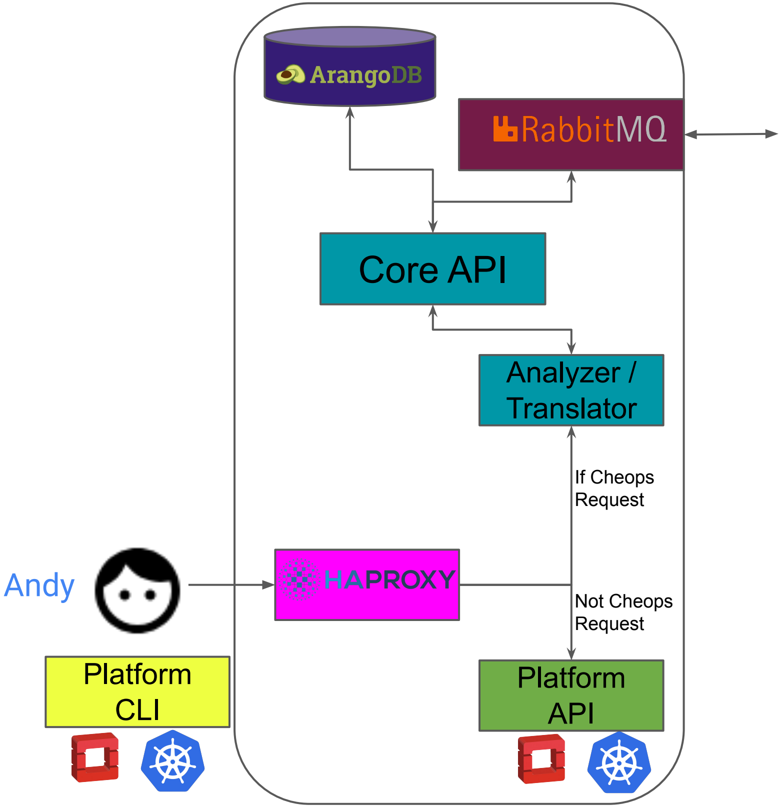


Interceptor

Analyzer / Translator

Glue

Cheops Framework



Cheops Site

Broker:

- P2p AMQP broker
- RabbitMQ

DB:

- NoSql Document based
- ArangoDB

Interceptor:

- Reverse-proxy for request capture
- HAProxy

Current building blocks can change in future.

Current Status

Cheops Development efforts are ongoing

- Initial efforts were made for the POC
  - Sharing, replication & cross collaborations feasibility is tested
  - Developed replication & cross POC under Kubernetes

Public repo: https://gitlab.inria.fr/discovery/cheops

Takeaway + What’s Next

Takeaway And Future Work

Can we go from Cloud to Edge without intrusive changes in the business logic?
Yes! Cheops demonstrates it.

We are **open for collaborations!!!** (for Cheops on various levels)

Multiple developments are scheduled, with a focus on Framework & Collaboration:

- Autonomous loops similar to a Kubernetes CRD
- Autonomous Site optimisation
- Better network partition handling
- More development on Scope-lang & Cheops Framework

Thanks For Your Attention!

You can contact us:
geo-johns.antony@inria.fr,
marie.delavergne@inria.fr,
baptiste.jonglez@inria.fr,
adrien.lebre@inria.fr

Cheops Public repo: https://gitlab.inria.fr/discovery/cheops

http://stack.imt-atlantique.fr