POLITECNICO DI TORINO
Repository ISTITUZIONALE

Enabling compute and data sovereignty with infrastructure-level data spaces

Original
Enabling compute and data sovereignty with infrastructure-level data spaces / Marino, J., Camiciotti, L., Cheinasso, F., Olivero, A., Risso, F.. - ELETTRONICO. - (2023), pp. 77-85. (3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum Ludwigsburg (DE) October 17, 2023) [10.1145/3624486.3624509].

Availability:
This version is available at: 11583/2982281 since: 2023-09-19T07:25:33Z

Publisher:
ACM

Published
DOI:10.1145/3624486.3624509

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

13 September 2026

Enabling Compute and Data Sovereignty with Infrastructure-Level Data Spaces

Jacopo Marino
Politecnico di Torino
Torino, Italy
jacopo.marino@polito.it

Leonardo Camiciotti
TOP-IX Consortium
Torino, Italy
leonardo.camiciotti@top-ix.org

Francesco Cheinasso
ArubaKube
Torino, Italy
francesco.cheinasso@arubakube.cloud

Alessandro Olivero
ArubaKube
Torino, Italy
alessandro.olivero@arubakube.cloud

Fulvio Risso
Politecnico di Torino
Torino, Italy
fulvio.risso@polito.it

ABSTRACT

Data is a critical asset in today's world. When multiple actors are involved, ensuring that data is accessible only to authorized parties and protecting it against theft present significant challenges. A potential solution to these issues is creating data spaces that interconnect clusters managed by different actors. The latter can securely exchange data under specific constraints and terminate connections when needed. This paper aims to show how infrastructure-level data spaces, which support both *access* to data, and *processing* of it, can facilitate secure data exchange and limit data theft. Furthermore, we investigate how data sovereignty can be maintained through cluster data exchange, which is crucial in an era where data is increasingly regulated and controlled. Additionally, we explore how offloading applications from the data consumer into the data producer cluster can match data gravity patterns, improving overall system efficiency. Finally, this paper presents the potential integration of the proposed solution within the framework of IDSA and Gaia-X, serving as promising option for implementing their proposed functionalities.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Networks** → *Cloud computing*.

KEYWORDS

Data Spaces, IDSA, Data sovereignty, Gaia-X, Liqo

ACM Reference Format:
Jacopo Marino, Leonardo Camiciotti, Francesco Cheinasso, Alessandro Olivero, and Fulvio Risso. 2023. Enabling Compute and Data Sovereignty with Infrastructure-Level Data Spaces. In *3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum (ESAAM 2023), October 17, 2023, Ludwigsburg, Germany*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3624486.3624509

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ESAAM 2023, October 17, 2023, Ludwigsburg, Germany
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0835-0/23/10.
https://doi.org/10.1145/3624486.3624509

1 INTRODUCTION

Data is a key asset of our modern digital society. However, the actors in charge of data production and consumption are often different, posing non-trivial challenges to control who can access what, and (even more important) to guarantee that data is not stolen and/or copied illegally, particularly when sensitive data is concerned. These problems can be analyzed with two concepts: *data sovereignty* and *data gravity*.

Data sovereignty involves, or can be identified with, the control of data flows and the related infrastructure via national jurisdiction [10]. For example, governments are worried about data sovereignty when their information is stored in the cloud and are questioning how to ensure confidentiality and prevent government data from being subject to another jurisdiction if hosted abroad [14]. These concepts can be clearly extended to universities and companies, where data can be hosted outside their facilities. In this context, a mention of the European General Data Protection Regulation (GDPR) has to be made. It imposes obligations onto organizations anywhere, so long as they target or collect data related to people in the European Union [17]. Regarding Europe, it is worth mentioning two additional pieces of legislation, known as the Data Governance Act [7] and the Data Act [6]. The former seeks to increase trust in data sharing, strengthen mechanisms to increase data availability, and overcome technical obstacles to the reuse of data . The latter complements the Data Governance Act by clarifying who can create value from data and under which conditions.

Data gravity is the ability of data to attract applications, services, and other data. Each data chunk has a mass and density, and when these two characteristics become large, it becomes difficult in terms of time, logistics, and cost-effectiveness to move the data from one location to another over the network. It is similar to the scenario of a large physical mass exhibiting gravitational pull to its surrounding objects: the larger the mass, the stronger the attraction [19].

Data spaces allow us to solve those problems by setting up a restricted area in which third parties can consume data and access only what the producer wants to share with them. This paper aims to demonstrate a cloud-based technology that can dynamically create flexible data spaces upon request, potentially spanning multiple administrative domains, leveraging Liqo [15]. This enables a data producer to offer its data to potential consumers, without giving up on security and data ownership/sovereignty rules, and without affecting the possibility of consumers to read and process

arbitrary data. Differently from current data spaces, in which only *communication* (i.e., data transfer) is involved among (data) producer and consumer actors, the proposed solution allows the data consumer cluster to borrow also *processing* resources in the data producer cluster, associating them to its cluster and creating a virtual continuum that spans both. While being more flexible (e.g., data can be processed on the data producer cluster, without requiring long-distance and expensive data movements), it introduces several advantages in terms of data protection. Any service running on the virtual continuum is controlled by the owner of the virtual cluster (hence, data consumer), which therefore has complete control over the lifecycle of its own services, and no control over others that are not owned. On the other side, the data producer does not give up full control of its resources and it may impose additional policies that enable *controlled access* to all the *guest* services running on its domain. If the data producer finds that the data usage deviates from the agreed-upon terms, it retains the right to terminate the partnership. It is noteworthy that the control at this level should be executed in tandem with oversight of the *guest* services. This dual oversight is critical because the data provider needs to ensure that the algorithm being executed within its domain adheres strictly to the terms agreed upon and that no variations from the approved algorithm are being implemented.

Our solution can automatically wrap the application and enforce the defined privacy, security, and access policies, facilitating data producers to offer their data to arbitrary actors, counting on a more solid technical background than simple trust (and legal agreements), hence facilitating the sharing of relevant data among multiple parties. It can dramatically simplify the operations of data producers and consumers (which are often different actors), as well as processing multiple data sources, thanks to a data-gravity approach in which processing is (securely) moved close to data sources.

The remainder of this paper unfolds as detailed below. Section 2 introduces the concept of data space and its associated requirements. Section 3 showcases the Liqo open-source project, which represents the ground for our solution. Section 4 presents the architecture of the proposed solution, followed by its implementation specifics in Section 5. Section 6 details the testing protocols and their respective outcomes. Potential integration of our solution within the Gaia-X architecture is delineated in Section 7. The paper draws to a close with Section 8, summarizing key takeaways and highlighting avenues for future research directions.

2 DATA SPACES

The International Data Spaces Association (IDSA) [11] aims to create a global standard for International Data Spaces (IDS), as well as to foster technologies and business models that will drive the data economy of the future in Europe and around the globe. The EU-funded Open DEI project [16], which is part of this ecosystem, published a position paper defining a data space as a decentralized infrastructure for trustworthy data sharing and exchange in data ecosystems based on commonly agreed principles. Users of such data spaces are enabled to access data in a secure, transparent, trusted, easy, and unified fashion. These access and usage rights can only be granted by those persons or organizations who are entitled to dispose of the data.

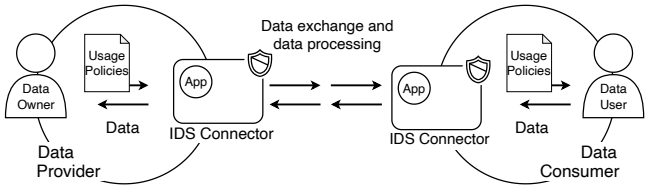


Figure 1: IDSA architecture for data spaces.

The IDS standard is adopted by projects like Gaia-X [9], which aims to create an ecosystem whereby data is shared and made available in a trustworthy environment, and employed by Catena-X [1], which can be seen as a data space inside that data ecosystem. Gaia-X defines a data space as a type of data relationship between trusted partners who adhere to the same high-level standards and guidelines in relation to data storage and sharing within one or many vertical ecosystems (i.e. Health, Infrastructure, Tourism, etc.).

2.1 Architecture requirements

The IDS Reference Architecture Model (IDS-RAM) [12] is the beating heart of the IDS: it comprises the standards for secure and sovereign data exchange, certification, and governance across Europe and around the world. Open DEI identified seven architecture requirements that data spaces must fulfill to be considered as such. These requirements include data-sharing empowerment, trustworthiness, publication, economy, interoperability, and data space engineering flexibility, and community. The aim of these measures is to ensure appropriate stakeholder control, security, privacy, data exchange, customization, and community support in data spaces. In Section 4, more detailed and lower lever requirements will be explained.

The IDS Connector is the key technical component to achieve secure and trusted exchange of arbitrary data, dealing with the authentication of the entities involved in the data exchange, and the attachment and enforcement of usage policies to data. Moreover, it acts as an application-level gateway (e.g., protocol translator), hence providing a uniform access to any data and by hiding specific protocols used internally towards the actual data source. This enables the exchange of data between clusters while ensuring the enforcement of all necessary security measures and policies.

The connector sends the data directly to the recipient from the device or the database in a trusted, certified data space, so the original data provider always maintains control over the data and sets the conditions for its use, as shown in Figure 1 [11]. The connector uses technology that puts data inside a sort of *virtual container*, which ensures that it is used only as agreed upon per the terms set by the parties involved. IDSA specifies the connector architecture, but then there are several implementations, which are interoperable (i.e. they can communicate with each other seamlessly) thanks to the common standard they adhere to, such as the Eclipse Dataspace Connector [3], TRUE (TRUsted Engineering) Connector [4], and the Dataspace Connector (DSC) [8].

3 LIQO

Liqo [13, 15] is an open-source project that enables dynamic and seamless Kubernetes multi-cluster topologies, supporting heterogeneous on-premise, cloud, and edge infrastructures. It allows to share resources and services coming from different clusters, which are aggregated into a seamless computing continuum called *virtual cluster*, as depicted in Figure 2, where *Cx* are clusters, *Px* are pods, and *Sx* are services. The contributing clusters are completely autonomous and can decide the size of the resources to share each other; furthermore, some resources can be kept outside the virtual cluster (such as pod *P*6 and service *S*1 in Figure 2, which are accessible only from non-offloaded pods of *C*2).

Within the virtual cluster, each remote cluster is abstracted by a new virtual node: each pod scheduled on that node will be offloaded and executed in the remote cluster, and it is reachable from any pod, either local or remote, belonging to the virtual cluster (full pod-to-pod connectivity). Furthermore, Liqo supports also *service* offloading: if a service (e.g., a Kubernetes ClusterIP) is created in the virtual cluster, it is reachable from any pod belonging to the computing continuum.

Liqo's capability to support arbitrary clusters, with varying parameters and components such as CNI plugins, poses a challenge to ensure non-overlapping pod IP address ranges or *PodCIDR* in the different clusters. This may necessitate address translation mechanisms, given that NAT-less communication is preferable whenever address ranges are disjointed. During the peering phase, Liqo exchanges relevant network configurations between peers (i.e., *PodCIDR*, *ServiceCIDR*, etc.) and possibly enables NATting rules in the Liqo gateway (the component in charge of the inter-cluster tunnel) in case overlapping IP addresses are detected, before setting a secure tunnel between the two clusters. This enables full pod-to-pod connectivity across the newly created virtual cluster.

More recent Liqo versions ($\geq$ 0.10) include also an initial support for security policies, which allow to segment traffic between the clusters, hence avoiding that a pod in one cluster could be able to communicate with a random pod in the other cluster, restricting the communication only to allowed pods.

The main advantage of Liqo is that each organization keeps control over its own resources, being able to revoke access to them at any time, effectively securing the shared infrastructure built on top of real clusters. In fact, each cluster can terminate the peering connection at any time, thereby revoking access its shared resources and tearing down all offloaded pods (and services).

4 ARCHITECTURE

In a traditional interaction between two administratively distinct actors, one acting as data producer and the other as data consumer, each actor controls its own cluster, while (raw) data is transferred between the parties. Each actor has a vested interest in maintaining control over the interactions among data and processing services in its cluster, with a specific emphasis on the owner of sensitive data, which may give access to sensitive data while, at the same time, preventing unauthorized data exfiltration.[1]

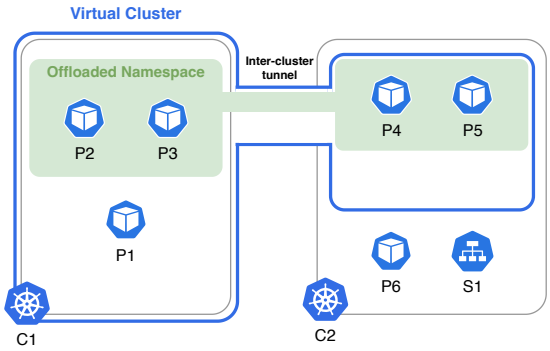


Figure 2: Liqo architecture overview.

To illustrate this concept in a straightforward manner, we consider a scenario depicted in Figure 3, with two clusters. A first cluster is owned by a pharmaceutical company (Pharma) that needs to execute its newest algorithms on the patient data, while a second is owned by a hospital (Hospital) that possesses the data to be processed (e.g., medical records). Hospital would like to run the Pharma algorithm on the patient data, but it must prevent that this sensitive data is returned and read by Pharma. Currently, the only solution consists in transferring the data to the Pharma application, and making sure (e.g., through a legal agreement) that data is not stolen. However, no technical methods are available to control that this agreement is actually respected by Pharma.

In our architecture, Pharma seeks to outsource the execution of its workloads to Hospital, while retaining the ability to govern the offloading process using the capabilities provided by Liqo. In other words, Pharma algorithms are running in the Hospital cluster, but those are still in control of the Pharma actor, which handles the lifecycle of this application as it was running locally. On the other hand, Hospital wishes to review offloading requests before accepting or rejecting them. Upon receiving a valid request, Hospital subjects it to a series of additional security measures (e.g., how the offloaded service can communicate with the Pharma cluster; more details in the next Sections), thereby ensuring the secure execution and monitoring of third-party workloads.

The management of these clusters, including their associated complexities, is seamlessly handled by Liqo through standard declarative configurations, thus providing a vanilla Kubernetes experience. Thanks to its capabilities, Liqo simplifies the deployment and management of a robust and secure multi-cluster environment, allowing users to focus solely on the Kubernetes resources required to create the data space playground. However, the proposed solution must adhere to several constraints that establish trust among the involved parties regarding their handling of data and computational resources. These constraints concern network connectivity and privileges restrictions, to prevent pods from bypassing imposed ones. To provide a clear understanding of their fulfillment, we present a concise description of these constraints in Table 1, based on the current capabilities of Liqo at the time of writing.

[1]With data exfiltration we refer to the capability to provide sensitive data to external partners, which has to be consumed by processing services that must produce (and return) only aggregated data, without disclosing all the original details present in the original data.

Building upon the aforementioned scenario, we can further extend it to manage more intricate situations involving multiple data producers and consumers, potentially overlapping in a way where a producer can also act as a consumer when considering other data.

4.1 Workflow

This subsection outlines the process of creating a secure infrastructure-level data space between Pharma and Hospital clusters, enabling data consumption through pod offloading while ensuring the implementation of security measures.

This workflow assumes that the two clusters, Pharma and Hospital, have already been configured and are operational, with Liqo already installed. At this stage, the clusters remain disconnected, and a peering connection must be established between them. Since the Pharma cluster intends to access data in the other cluster, it initiates a Liqo peering towards the Hospital cluster. If the Hospital cluster accepts the connection, it pushes security rules to its Liqo Gateway, granting Pharma access to the Hospital services running in the "data-space" Kubernetes namespace (named *data-ns* in Figure 3), which hosts the sensitive data and it is *external* to the Pharma virtual cluster. Within its virtual cluster, Pharma can then read data or offload one or more pods in the Hospital cluster, allowing for data gathering.

When Pharma initiates the offloading process, Hospital detects it and automatically applies a set of security measures to securely host those pods within its cluster. These measures include implementing Kubernetes Network Policies, performing a mutating operation on the offloaded pods, and enforcing firewall rules on the connecting inter-cluster gateway. Offloaded pods are granted access to the protected data, allowing them to manipulate and aggregate the data, thereby adding a new layer to the data producer cluster.

The above security rules ensure that all Pharma pods running on the Pharma cluster can only connect to Pharma pods on the Hospital cluster. Specifically, the set of Network Policies ensures that the offloaded pods can only communicate with selected services while blocking all other requests within the cluster. This is achieved using the standard Kubernetes APIs. The processed data can then be transmitted to Pharma through the Liqo inter-cluster tunnel. The Hospital Liqo Gateway ensures that incoming traffic from the other cluster can only reach its offloaded pods. The process of pod offloading is managed by a Mutating Webhook, which adds an *init* container and a *sidecar* to the main application container, which create the necessary iptables rules to forward all the traffic of offloaded pods to the Sidecar container, which acts as a proxy and monitors all the pod's communications, allowing only the desired ones. This mechanism introduces a strong barrier that safeguards the Hospital cluster against data exfiltration. In fact, this Sidecar intercepts all traffic from offloaded pods and directed outside the Hospital cluster, inspecting the data at the application level through a transparent Application-Level Security Gateway (ALSG) functionality. It enables all Pharma pods running on the Hospital cluster to establish uncensored communications with a list of selected destinations within the Hospital cluster, without ALSG. Finally, it allows service communications from all offloaded Pharma pods to local Kubernetes services in the Hospital cluster, such as DNS queries.

5 IMPLEMENTATION

As depicted in Figure 3, the Liqo Gateway acts as the final destination of the inter-cluster tunnel that has successfully completed the peering phase. Given that the data exfiltration property has to be guaranteed for Hospital, this component (in the Hospital cluster) will deserve a special attention in our architecture. In fact, a new Kubernetes controller (the *gateway controller*) running in the Hospital cluster watches all pods that have been offloaded from the Pharma cluster, leveraging a proper label that is automatically attached by Liqo to all offloaded pods.[2] Following identification, the controller collects their IP addresses and includes them in a whitelist, thereby allowing them to be accessed exclusively by pods belonging to the Pharma cluster (either local or remote), the original offloader, while denying connections from all other clusters. These whitelisted IP addresses are translated into `iptables` rules and pushed on the Liqo gateway of the Hospital. Upon cessation of offloaded pods, their respective IP addresses are removed from the whitelist, and the corresponding `iptables` rules are deleted. Since the Liqo gateway may also act as a NAT between clusters, `iptables` rules are appended outside the secure tunnel, hence are applied after traffic has been decapsulated from the tunnel. Consequently, the rules also lie outside the NAT, thereby making the NAT transparent in our implementation. The IP addresses utilized in the rules are local to the Hospital cluster.

Within the Hospital cluster, another controller, referred to as the *namespace controller*, oversees the reconciliation of offloaded namespaces. Similar to the gateway controller, it identifies offloaded namespaces through labels added by Liqo and appends a Network Policy (NetPol) and a ConfigMap to each identified namespace. Should a namespace be deleted, the associated NetPol and ConfigMap are likewise removed. Given that the NetPol is a resource, its deletion ensures the removal of the namespace and all encompassed resources. The NetPol governs the inbound and outbound traffic of the namespace in which it is deployed. It differentiates traffic flow, either from or to namespaces or specific resources, via a specific label. This label, ascribed by the Hospital cluster, allows the Hospital to discern which traffic should be permitted or blocked. If the label corresponds, traffic is allowed; otherwise, it is blocked. Both the sender and receiver must match the label for traffic to flow. Although the label is contained within the NetPol, matching occurs with respect to external resources to the offloaded namespace that intend to exchange traffic with it. The ConfigMap hosts the configuration parameters necessary for directing traffic from each offloaded pod through the Sidecar container.

The final *Mutating Webhook* component performs several operations in the event of pod offloading. An Init container is introduced into the pod prior to the main container when the offloaded pod is scheduled. This container accesses the configuration stored in the namespace ConfigMap and establishes rules that mandate all main container traffic to go through the Sidecar container. Upon the termination of the Init container, the main and Sidecar containers are created. The enforcement by the Init container ensures that traffic from the main container is routed via the Sidecar. As of the current

[2]It is worth mentioning that the number of offloaded pods can change over time (e.g., new replicas are created), which has to be detected by the above custom controller.

Table 1: Properties of secure data spaces.

Property	Description	Implementation
Restricted privileges	The offloaded pod must adhere to restrictions set by the hosting cluster and peering can be ceased at any time.	The pods need to be started as non-privileged and isolate the host as much as possible in terms of networking and process-level information and data. It is essential to prevent the pods from disabling or bypassing the imposed restrictions, meaning they should not have permission to remove the Sidecar or disable/modify the iptables rules.
Restricted Pod-to-pod connectivity	The full pod-to-pod connectivity within the virtual cluster should be limited to avoid offloaded pods to communicate with arbitrary other pods.	To restrict pod-to-pod connectivity within a cluster, two components are utilized: Network Policy and iptables. The Network Policy governs the communication between local and offloaded pods of different entities within the cluster. On the other hand, iptables rules are added to the Liqo Gateway to limit pod interactions by allowing only a select few. While pods of the same entity can interact freely, local and offloaded pods can only communicate with authorized service endpoints, including all endpoints of the reflected service in Liqo's implementation, to ensure the proper functioning of the service.
No network data exfiltration	The offloaded pods must communicate with the external world and within the host cluster in a controlled fashion, adhering to pre-determined data representation models and encryption requirements.	A Sidecar, acting as a proxy, is added to each offloaded pod and it is set by a ConfigMap, ensuring that all the traffic of the pod is forced to go through it. Right now, the only implemented feature is forcing the traffic through the Sidecar, but there is no automatic action enforced to avoid data exfiltration. Exclusive communication access is granted only to the offloaded pods, while any communications originating from pods within the data consumer cluster are blocked. Offloaded pods are restricted from communicating with the Internet to prevent bypassing inter-cluster policies.
Traffic inspection	Inspect traffic from offloaded pods to "home" pods to prevent data exfiltration.	The Sidecar currently lacks traffic inspection capabilities. However, it does monitor all requests to and from the application container.
Service protocols and communications	To prevent the creation of resources that could potentially bypass imposed restrictions, offloaded pods must be restricted from accessing the API server of their original cluster.	Currently, it is possible to disable communication to the API server of the original cluster for all offloaded pods, but there is no fine-grained control.
Restricted Volume mounting	The offloaded pod must not mount external volumes, to prevent data exfiltration by bypassing controlled communication.	Liqo does not currently offer any built-in volume mounting restriction mechanisms.
No code exfiltration	To ensure the security of offloaded pods, they must be protected against image theft and unauthorized image pulling, while also maintaining isolation from the host cluster to prevent inspection, changes, or execution.	To mitigate this risk, offloaded pods should be deployed with the *pull always* flag enabled. Additionally, in scenarios where safeguarding against code exfiltration is necessary, it might be advisable to restrict the presence of only the aggregation layer of the algorithm in the remote cluster, while keeping the main algorithm protected. However, Liqo does not currently offer any built-in code protection mechanisms, but it can be solved with orthogonal solutions.
Restricted resource consumption	The cluster that accepts offloaded pods should be able to limit their resource consumption to avoid a negative impact on cluster operations.	A cache system, a custom data structure, stores the information of each Liqo peering connection and the defined limits imposed by the data producer. The Validating Webhook is the only component that has access to it, and it is the only one that can make the proper calculations. This system provides constraints that limit aggregate resource consumption at the peering level. It can limit the number of objects that can be created in a remote cluster, as well as the total amount of compute resources that may be consumed by workloads in that cluster.

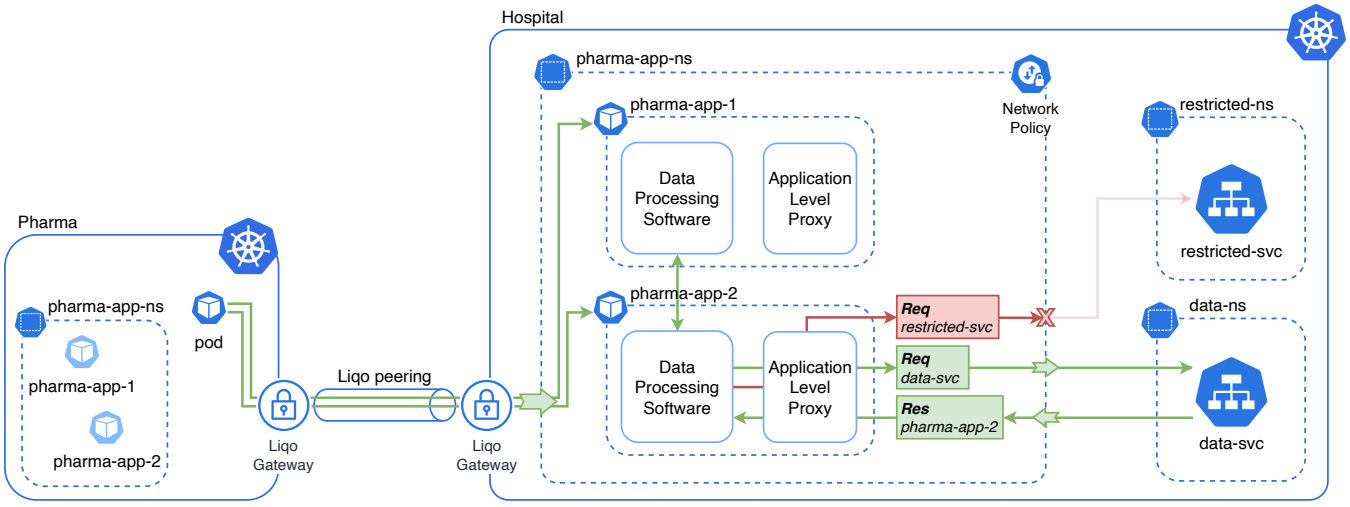


Figure 3: Infrastructure-level data space: deployment example and main communication flows.

implementation, the Sidecar functions as a traffic monitoring tool, devoid of any metrics, and is built upon the Envoy proxy [5].

6 EXPERIMENTAL EVALUATION

Experimental results have been collected with our custom software running on Liqo v0.8. This version did not include (yet) a proper implementation of the cross-cluster network policies, which had to be carried out by our proof-of-concept implementation.

The gathered results pertain to the following scenario. Within the Pharma cluster, a pod seeks to access the data supplied by the Hospital cluster. Given the aforementioned workflow has been accomplished, the Pharma cluster has a single pod containing the processing logic, which is offloaded to the Hospital cluster. When the Pharma cluster's pod intends to access data, it must reach out to the offloaded pod to request the desired data. The offloaded pod then communicates with the service exposed by the Hospital within the data space, to which it has been granted access. It collates data, performs an aggregation or analysis, and subsequently sends the processed information back to the requesting pod. All the data sent back to the Pharma pods is inspected and checked by the sidecar, which makes sure no sensitive data is transmitted (e.g., checking the data against a well-defined data structure). In a nutshell, the offloaded pod serves as intermediary, bridging the gap between the Pharma pod and the data provided by the Hospital cluster.

6.1 Data space creation time

The standard, or *vanilla*, execution of Liqo requires the establishment of a point-to-point connection between the clusters. Subsequently, a namespace is offloaded, and we will hereinafter refer to it as *offloaded namespace*. Within this offloaded namespace, all deployed resources can benefit from the offloading feature of Liqo. Contrarily, the resources deployed within other namespaces, which are not subject to offloading, are unable to benefit from it.

Our solution adheres to the same two-phase process: peering and namespace offloading. Accordingly, the reported measurements are applicable to both implementations, obviating the need for comparative analysis. The peer-to-peer connection requires a time duration of (3.8098 ± 0.7780) *s*, and the namespace offloading necessitates (0.3097 ± 0.0738) *s*. A cursory glance at these values reveals that the peering phase is the predominant consumer of time.

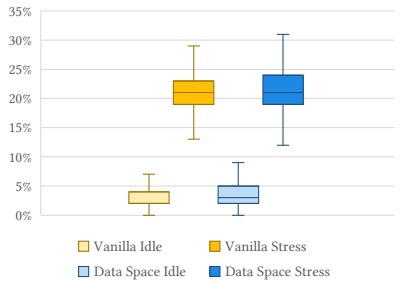
Table 2: Pod deployment time.

#Offloaded Pods	Vanilla (s)	Data Space (s)
1	$0,095 \pm 0,021$	$0,09 \pm 0,023$
5	$0,240 \pm 0,069$	$0,214 \pm 0,077$
10	$1,214 \pm 0,038$	$1,218 \pm 0,038$
100	$32,794 \pm 6,346$	$31,945 \pm 3,368$

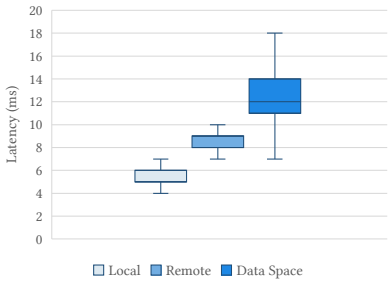
The third phase pertains to the deployment phase where we assessed the deployment of several pods in both the standard Liqo setup and our proposed solution. These findings are encapsulated in Table 2. Upon scrutiny, it becomes evident that the results for the two scenarios bear a remarkable similarity. This observation allows us to infer that the inclusion of Init and Sidecar containers in our proposed solution does not significantly influence the time required for deployment.

6.2 Resource consumption

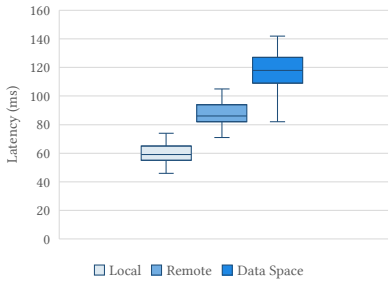
Resource consumption serves as a key metric for evaluating the computational demands placed upon a system during data transfer events. In the context of our study, this evaluation aims to identify any potential overhead on the Liqo Gateway resulting from our additional security rules. To this end, we conducted an analysis based on four distinct scenarios, as depicted in Figure 4a, which were drawn from both the vanilla implementation of Liqo and our proposal. Initially, data was collected during an idle period before a stress test was initiated. This test design was intended to facilitate a comparative examination of the four scenarios. To simulate data transfer between the two clusters, we used the tool iPerf [2].



(a) CPU consumption.



(b) Latency with 10 parallel connections.



(c) Latency with 100 parallel connections.

Figure 4: Experimental evaluation.

Upon examining the graph, we observed that our implementation, denoted as *Data Space* in Figure 4a, imposes only a negligible amount of overhead, whether in idle or stress conditions, when compared to the vanilla Liqo implementation. This minor increase in CPU consumption is a small price to pay for the advantages of our proposed solution, which allows data to be provisioned via data spaces. This solution avoids introducing any substantial overhead that might be deemed unacceptable if it significantly exceeded that of the vanilla Liqo implementation. We also recorded RAM consumption throughout our experiments. However, our findings indicated that our solution had no discernible impact on RAM usage when compared to the vanilla Liqo. Consequently, in the interest of brevity, we have omitted the corresponding graph.

6.3 Latency

Latency, within this context, is defined as the duration between the moment data is requested by a pod and the point at which the requested data becomes available. The premise of our experiment was that the desired data resides within a pod, leading us to explore pod-to-pod communication scenarios. Under this premise, we evaluated three distinct scenarios to ascertain the time needed for a pod to access data. (i) In the first scenario, the communicating pods were located within the same cluster (either Pharma or Hospital). Thus, these pods utilized local connectivity for data exchange. This scenario simulates communication between a data-seeking pod and a data-provider pod within the same cluster. We term this scenario as *Local*. (ii) In the second scenario, the communicating pods were situated across two different clusters (one in Pharma and the other in Hospital), necessitating the use of Liqo for connection. Consequently, data transfer had to pass through the inter-cluster tunnel. This scenario depicts communication between pods in separate clusters, offering a comparison point to a situation where Pharma needs to access a remote service located in the Hospital cluster and download all relevant data for computation within the Pharma cluster. We refer to this scenario as *Remote*. (iii) The third scenario mirrors the second but it introduces one or more offloaded Pharma pods within a data space in the Hospital cluster. In this setup, Pharma pods need to liaise with these offloaded pods, which in turn, communicate with the data-bearing pod in the Hospital. The offloaded pod(s) play a *proxy* role, providing the Hospital with control over data output. This scenario is referred to as *Data Space*.

We gathered latency benchmarks utilizing Apache Benchmark [18], entailing 10K requests with a varying number of parallel connections. For ease of understanding, we report two instances, one involving 10 parallel connections and the other 100, depicted in Figure 4b and Figure 4c, respectively. As illustrated in these figures, pod-to-pod latency increases when the pods belong to different clusters compared to those within the same cluster. However, the *Data Space* scenario warrants further examination. During the testing process, the offloaded Pharma pods functioned merely as proxies, managing all requests to Hospital data, thus inducing a noticeable latency impact. This proxy role gives Hospital control over data flow since the offloaded pods are situated within the data space. When deploying our solution in this manner, latency increases by up to 37% (median value) in comparison to the *Remote* scenario.

This finding directs us back to the initial premise of our work. If we leverage the data gravity capability inherent in our proposed solution, we can manage local and remote latencies more effectively. By assigning data aggregation or analysis tasks to the Pharma offloaded pod, the resultant data has a smaller size compared to the original. In this setup, communication between the offloaded pod and the data-owning Hospital pod remains local to the Hospital cluster, mimicking the *Local* scenario latency behavior depicted in Figure 4b and Figure 4c. Conversely, when the Pharma pod retrieves the aggregated data from the offloaded pod, the latency mirrors the *Remote* scenario. Although *Data Space* latency is higher than *Local* and *Remote* latencies, data aggregation mitigates this issue by reducing latency within the data aggregator pod and lowering the total volume of data transferred from Hospital to Pharma, hence the total cost of the deployment in case of clusters running in public cloud providers. While we cannot lessen the latency directly, we can effect a significant reduction in total data transfer time.

7 LIQO AND THE GAIA-X ARCHITECTURE

The data spaces architecture presented in Section 2 and generally adopted by Gaia-X defines application-level primitives, but it does not mention any infrastructure-level solution. In our opinion, application-oriented data spaces open up the following problems: (i) it enables secure data exchange, but it does not support any computing component (i.e., pods); hence, in case of a very large quantity of data, all data needs to be transferred from the producer to the consumer (no support for data gravity); (ii) IDSA connector

allows access to data from the data consumer, but the latter might prefer having its own API gateway with its aggregation rules; (iii) mutually reachable endpoints are required (e.g., publicly exposed on the Internet) to allow connectors to exchange data.

The architecture presented in this paper can be easily integrated with application-level data spaces to address the aforementioned problems. For instance, in infrastructure-level data spaces, connectors are just pods that can be executed on either the local or remote cluster, depending on the desired scenario. Hence, considering the architecture in Figure 1, Liqo can be used to enhance the pure data exchange capabilities of current data space solutions with a data-gravity approach, enabling the remote deployment of processing instances instead of transferring data. We begin with an overview of the existing workflow involving IDS Connector, followed by an exploration of integration with Liqo. We assume that each involved entity has both the IDS Connector and Liqo already installed.

IDS connectors can publish the description of their data endpoints at an IDS meta-data broker. This allows potential data consumers to look up available data sources and data in terms of content, structure quality, actuality, and other attributes. Consequently, data consumers can make informed decisions on selecting a data producer based on their specific requirements and needs.

The initial step entails the data consumer contacting the chosen data producer via the data connector to initiate a request for data exchange. To establish a secure communication channel, the data producer's connector and the data consumer's connector engage in authentication and authorization mechanisms. These mechanisms verify the identity and access rights of both parties, which may involve the exchange of authentication tokens, certificates, or other credentials. Once the data producer's connector has validated the data usage conditions and obtained any necessary consent, it grants the data consumer's connector access to the requested data. The data consumer's connector enforces any access control rules defined by the data producer's policies, such as filtering or reduction. Over the established communication channel, the data producer's connector securely transmits the requested data to the data consumer's connector, allowing the latter to receive and process the data according to its specific requirements. Within this workflow, Liqo introduces the potential for two new scenarios.

The first scenario leverages the nature of Liqo itself. Utilizing its direct connection between clusters, Liqo can facilitate peering between the involved entities after the authentication phase. In essence, instead of exchanging data between the connectors through public endpoints, Liqo enables the use of a private tunnel and service reflection. By doing so, the need for public endpoints to transmit data is eliminated, ensuring enhanced security without disrupting the current implementation of the IDS connector.

The second scenario leverages Liqo's ability to deploy pods in remote clusters for exchanging computations rather than data. This approach relieves the requirement to transfer large amounts of data by instead transmitting the processing units responsible for handling that data. Leveraging Liqo's data-gravity approach, bandwidth consumption is reduced, and computations are attracted to the data, thereby optimizing resource utilization and efficiency.

8 CONCLUSIONS

This paper delineates a novel approach to create infrastructure-level data spaces, harnessing the capabilities of the Liqo project. The proposed solution enables data consumers to establish effective connections with data producers, thereby facilitating controlled data retrieval. Additionally, it promotes the deployment of processing algorithms closer to the data, thereby fostering data gravity.

Our findings substantiate the advantages of this solution, showcasing a negligible overhead against the *vanilla* Liqo. The proposed solution can be employed to safeguard data exchange via data spaces, empowering data providers with the ability to regulate the exchange of data. This capability gives data providers the flexibility to terminate the connection as and when they see fit. Simultaneously, our solution offers the advantage of utilizing offloaded pods for data aggregation, enabling data consumers to tap into the potential of data spaces. This signifies a shift of computational operations closer to the data source, thereby diminishing latency between computation and data. Moreover, it facilitates a reduction in data transfer if the data is aggregated or analyzed *in situ* by offloaded pods before being transmitted to the data consumer. The contraction in data transfer consequently reduces the time required for data retrieval once the aggregation or analysis phase concludes. When contextualized within the framework of initiatives like IDSA and the Gaia-X project, our proposed solution can serve as an instrumental component of these broader ecosystems, acting as a key element to enable data gravity.

In conclusion, our solution provides a versatile, efficient, and secure approach for infrastructure-level data exchange. By enabling data gravity, it aligns with future-oriented trends of data localization and analytics, empowering the architecture proposed by Gaia-X.

ACKNOWLEDGMENTS

This work was partly supported by the GEANT Innovation Programme Year 2022, project "Borderless Data Spaces", and the European Union's Horizon Europe research and innovation programme under grant agreement No 101070473, project FLUIDOS (Flexible, scaLable, secUre, and decentralIseD Operating System). In addition, this research was conducted as part of Jacopo Marino Ph.D. programme, under the financing of the Piano Nazionale di Ripresa e Resilienza (PNRR) and the NextGenerationEU initiative. Finally, we would like to thank Luca Francescato, Marco Iorio, Claudio Pisa, Ferdinando Ricchiuto, and Francesco Torta for their support, comments, and contributions in the different stages of this work.

REFERENCES

[1] Catena-X. 2023. *Catena-X.* https://catena-x.net
[2] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh Prabhu. 2023. *iPerf.* https://iperf.fr
[3] Eclipse Foundation. 2023. *Eclipse Dataspace Connector.* https://projects.eclipse.org/projects/technology.edc
[4] Engineering. 2023. *TRUE Connector.* https://www.eng.it/en/case-studies/true-connector-per-facilitare-la-condivisione-di-dati-in-gaiax
[5] Envoy Project Authors. 2023. *Envoy.* https://www.envoyproxy.io
[6] European Commission. 2022. *The European Data Act.* https://digital-strategy.ec.europa.eu/en/library/data-act-proposal-regulation-harmonised-rules-fair-access-and-use-data
[7] European Commission. 2022. *The European Data Governance Act.* https://digital-strategy.ec.europa.eu/en/policies/data-governance-act
[8] Fraunhofer ISST. 2023. *Dataspace Connector.* https://www.isst.fraunhofer.de/en/business-units/data-business/technologies/Dataspace-Connector.html

[9] Gaia-X. 2023. *Gaia-X.* https://gaia-x.eu
[10] Patrik Hummel, Matthias Braun, Max Tretter, and Peter Dabrock. 2021. Data sovereignty: A review. *Big Data & Society* 8, 1 (2021), 2053951720982012.
[11] International Data Spaces Association. 2023. *International Data Spaces Website.* https://internationaldataspaces.org
[12] International Data Spaces e. V. 2023. *IDS Reference Architecture Model.* https://github.com/International-Data-Spaces-Association/IDS-RAM_4_0
[13] Marco Iorio, Fulvio Risso, Alex Palesandro, Leonardo Camiciotti, and Antonio Manzalini. 2022. Computing Without Borders: The Way Towards Liquid Computing. *IEEE Transactions on Cloud Computing* (2022), 1–18. https://doi.org/10.1109/TCC.2022.3229163
[14] Kristina Irion. 2012. Government Cloud Computing and National Data Sovereignty. *Policy & Internet* 4, 3-4 (2012), 40–71. https://doi.org/10.1002/poi3.10 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/poi3.10
[15] Liqo. 2023. *Liqo.* https://liqo.io
[16] Lars Nagel and Douwe Lycklama. 2021. Design Principles for Data Spaces - Position Paper. https://doi.org/10.5281/zenodo.5105744
[17] Proton AG. 2018. *General Data Protection Regulation (GDPR) Website.* https://gdpr.eu/what-is-gdpr/
[18] The Apache Software Foundation. 2023. *ab - Apache HTTP server benchmarking tool.* https://httpd.apache.org/docs/2.4/programs/ab.html
[19] Coral Walker and Hassan Alrehamy. 2015. Personal Data Lake with Data Gravity Pull. In *2015 IEEE Fifth International Conference on Big Data and Cloud Computing.* 160–167. https://doi.org/10.1109/BDCloud.2015.62