



All-in-one middleware for industrial human-robot-interaction

arise-middleware.eu

Document History

Ver.	Date	Description	Author	Partner
0.1	05/12/2024	ToC	Raúl Sánchez-Mateos	EPROS
0.2	09/12/2024	Fast DDS performance improvements, discovery configuration and new default ROS 2 mode	Carlos Ferreira	EPROS
0.3	09/12/2024	ROS 2 "Keys" feature	Mario Dominguez	EPROS
0.4	10/12/2024	Background, Fast DDS v3, and XTypes, and proofreading	Raúl Sánchez-Mateos	EPROS
0.5	11/12/2024	DDS Enabler	Juan López	EPROS
0.6	13/12/2024	Revision	Francisco Melenez	FF
1.0	13/12/2024	Vulcanexus benchmark	Raúl Sánchez-Mateos	EPROS

List of author(s), contributors(s) and reviewer(s)

Name	Partner (Acronym)	Contribution/Comment
Raúl Sánchez-Mateos	EPROS	Project Manager
Carlos Ferreira	EPROS	Software Engineer
Mario Dominguez	EPROS	Software Engineer
Juan Lopez	EPROS	Software Engineer

Disclaimer

The sole responsibility for the content of this publication lies with the authors. It does not necessarily reflect the opinion of the European Union. The European Commission is not responsible for any use that may be made of the information contained therein.

Publication Details

Grant Agreement Number 101135784

Acronym ARISE

Full Title	Agile, human-centric, and Real-time enabled SourceE technologies advancing industrial HRI in Europe
Topic	HORIZON-CL4-2023-DIGITAL-EMERGING-01-02
Funding Scheme	HORIZON Innovation Actions
Start Date	01/01/2024
Duration	42 months
Project URL	www.arise-middleware.eu
Project Coordinator	CARTIF
Deliverable	D2.2 - ARISE Vulcanexus Releases 1
Work Package	WP2 - Open Middleware and technology enablers for Industrial HRI
Delivery Month (DoA)	M12
Version	1.0
Actual Delivery Date	31.12.2024
Type	Other
Dissemination Level	Public
Lead Beneficiary	EPROS
Keywords	DDS, ROS, ROS 2, eProxima, Fast DDS, Middleware, Vulcanexus

D2.2 - ARISE Vulcanexus Releases 1

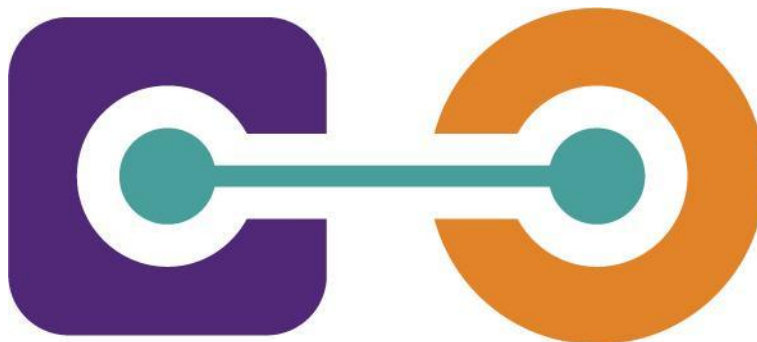


Table of Contents

1 Introduction	7
1.1 About the project	7
1.2 About this document	7
1.3 Intended audience	7
1.4 Reading recommendations	8
2 Background	9
2.1 eProsima Fast DDS	9
2.2 ROS 2	10
2.3 Vulcanexus	10
2.3.1 Vulcanexus Metapackages	11
2.3.2 Release Procedure	13
3 Vulcanexus Jazzy (v4.0.0)	14
3.1 Fast DDS v3	14
3.1.1 Performance improvements	15
3.2 X-Types support in ROS 2	16
3.3 DDS Keys in ROS 2	17
3.3.1 Benefits of keyed topics	19
3.3.2 Repository contributions for the feature	20
3.3.2.1 rosidl	20
3.3.2.2 rosidl_fastrtps_cpp	20
3.3.2.3 rmw_fastrtps	20
3.4 ROS 2 Discovery Mechanisms	20
3.5 Tutorials and Use Cases	23
3.6 Benchmarking of Vulcanexus over "plain" ROS 2	24
4 Future release	26
4.1 DDS Enabler	26
4.2 ROS 2 nodes new default configuration	26

List of Figures

Figure 1: Topics in ROS 2

Figure 2: Keyed topic

Figure 3: Topic instances example

Figure 4: Traffic flow in Discovery Server v2

Figure 5: Traffic flow in Discovery Server v3

Figure 6: Launch Schema for Discovery Server Auto Mode

Acronyms & Abbreviations

ACR.	Description	ACR	Description
DDS	Data Distribution Service		
HRI	Human-Robot Interactio		
LTS	Long-Term Support		
RCL	Ros Client Library		
OMG	Object Management Group		
CLI	Command Line Interface		
SHM	Shared Memory		
TCP	Transmission Control Protocol		
UDP	User Datagram Protocol		

1 Introduction

1.1 About the project

ARISE is an innovative action aimed at simplifying, reducing costs, and broadening the adoption of industrial Human-Robot Interaction (HRI) deployments across Europe. This initiative aligns with the principles of Industry 5.0, emphasizing the creation of resilient, sustainable, and human-centric work environments. In this envisioned future, companies recognize that investing in industrial HRI is not merely an expense but a strategic approach to achieving both immediate and long-term goals. By prioritizing human-centric methodologies over traditional technology-driven approaches, ARISE positions technology as a tool to empower people, rather than people serving or being replaced by technology. Within this framework, industrial HRI emerges as a transformative asset, fostering seamless collaboration between humans and robots, enabling them to tackle complex challenges efficiently and work collaboratively across diverse workplace scenarios and shift patterns.

1.2 About this document

This deliverable outlines the features, tools, and enhancements introduced in the Vulcanexus Jazzy (v4.0.0) release. The primary objective of this document is to detail the advancements in middleware functionality, such as Fast DDS v3 integration, the introduction of DDS keys, and enhanced discovery mechanisms, as well as to provide guidelines for the effective utilization of these new features.

This document serves as a technical reference for developers and stakeholders, accompanied by supporting materials such as tutorials, use cases, and installation packages, to facilitate the adoption and implementation of Vulcanexus Jazzy.

1.3 Intended audience

The intended audience for this deliverable includes:

- Robotics software developers and engineers seeking scalable middleware solutions.
- Researchers and academics in robotics and industrial IoT.
- Industry stakeholders involved in deploying distributed robotics applications.

1.4 Reading recommendations

This document is organized into several chapters. Chapter 1 introduces the project and the document's structure. Chapter 2 provides background information on core components such as Fast DDS, ROS 2, and Vulcanexus, along with their significance in distributed systems. Chapter 3 details the features of the Vulcanexus Jazzy release, including middleware improvements and new functionalities. Chapter 4 outlines the planned enhancements for future releases.

For a deeper understanding of the middleware's technical foundations, readers are encouraged to consult the official [Fast DDS documentation](#) and the [Vulcanexus Jazzy release notes](#). These resources complement the insights provided in this document and offer practical guidance for utilizing Vulcanexus effectively.

2 Background

2.1 eProsima Fast DDS

[Fast DDS](#), developed by eProsima, is a leading implementation of the Data Distribution Service (DDS) standard, an industry-recognized library for real-time, distributed, and high-performance communication. As an open-source solution, Fast DDS has become a critical enabler for applications demanding reliable and efficient data exchange, particularly in robotics, industrial IoT, and other latency-sensitive domains. Its role in the Robot Operating System 2 (ROS 2) as the default communication middleware highlights its relevance in cutting-edge robotics development.

Fast DDS is recognized for its robust set of features, making it a versatile and high-performing middleware:

- **High Performance and Low Latency.** One of the defining attributes of Fast DDS is its optimized performance. It achieves remarkably low latency and high throughput, even when handling large payloads, a critical requirement for applications such as autonomous robotics and real-time data analytics.
- **Zero-Copy Communication.** The integration of zero-copy communication sets Fast DDS apart by minimizing unnecessary data copying between application layers. This optimization significantly reduces communication overhead and boosts overall efficiency, particularly in scenarios involving high data rates or large message sizes.
- **Quality of Service (QoS) Flexibility.** Fast DDS provides a comprehensive array of configurable QoS policies, allowing developers to tailor communication behaviour to the specific needs of their applications. Whether prioritizing reliability, latency, or resource efficiency, these QoS settings ensure that Fast DDS can meet diverse operational requirements.
- **Scalability and Robustness.** Designed for scalability, Fast DDS supports distributed systems ranging from small networks of devices to large-scale, interconnected infrastructures. It ensures robust communication across nodes, even under challenging network conditions, making it suitable for both local and wide-area deployments.
- **Security Features.** Security is integral to Fast DDS, which incorporates mechanisms such as authentication, encryption, and access control. These features ensure that data integrity and confidentiality are maintained, addressing critical concerns in industrial and IoT applications.
- **Integration with ROS 2.** As the default middleware in ROS 2, Fast DDS facilitates seamless communication between robotic components. Its compatibility and ease of integration with ROS 2 make it a cornerstone of modern robotics systems, supporting tasks such as sensor data fusion, actuator control, and real-time decision-making.

Fast DDS's capabilities align perfectly with the needs of robotics and industrial IoT. In robotics, its integration with ROS 2 enables efficient communication within complex systems comprising multiple nodes and heterogeneous components. Autonomous robots, for instance, rely on Fast DDS for real-time exchange of sensor data and command messages, ensuring timely responses to dynamic environments.

In industrial IoT applications, Fast DDS serves as a communication backbone, enabling reliable and real-time data exchange between devices, sensors, and centralized systems. It supports critical use cases such as predictive maintenance, process automation, and digital twins by ensuring low latency and fault-tolerant communication across distributed networks.

2.2 ROS 2

The Robot Operating System 2 (ROS 2) is an open-source framework designed to support the development of robotic systems and applications. Building on the foundation of its predecessor, ROS 1, it introduces significant enhancements in scalability, flexibility, and real-time capabilities to address the needs of modern, distributed robotic systems. ROS 2 provides a comprehensive set of tools, libraries, and conventions that simplify the complex process of designing and deploying robotics software. It enables seamless communication between robotic components through its modular architecture, making it adaptable for applications ranging from small-scale robots to large industrial automation systems.

A key feature of ROS 2 is its middleware-agnostic design, achieved through the adoption of the Data Distribution Service (DDS) standard. This allows developers to leverage various DDS implementations, including those tailored for real-time and high-performance communication. Additionally, ROS 2 incorporates improved support for multi-platform deployments, robust security mechanisms, and enhanced Quality of Service (QoS) settings, making it suitable for mission-critical applications. With its commitment to modularity, interoperability, and scalability, ROS 2 has become a cornerstone for robotics research, development, and industrial adoption, fostering innovation across diverse fields.

2.3 Vulcanexus

Vulcanexus (<https://vulcanexus.org/>) is an all-in-one toolset designed to extend and enhance the capabilities of the ROS 2 ecosystem. Built around Fast DDS, which serves as its fixed middleware implementation, Vulcanexus provides a comprehensive suite of features and tools that go beyond the standard ROS 2 distribution. This makes it an ideal choice for developers seeking to build robust and efficient robotics applications while maintaining compatibility with the underlying principles and architecture of ROS 2. By offering features such as advanced communication mechanisms, performance monitoring, and simulation tools, Vulcanexus streamlines development workflows and improves system performance.

Vulcanexus significantly enhances the ROS 2 environment by incorporating the latest version of Fast DDS, unlocking advanced features that are not available in the standard ROS 2 distribution. Features such as topic keys improve data distribution and scalability, while the dynamic language binding API, compliant with the OMG XTypes v1.3 specification, enhances flexibility and integration. Additionally, capabilities like type propagation and the gather-send feature optimize performance, particularly when managing large data exchanges.

Beyond middleware improvements, Vulcanexus provides a variety of tools that enhance the developer experience. For instance, the ROS 2 Monitor offers a graphical interface for monitoring communications, while ROS 2 Spy provides a command-line tool for introspecting ROS 2 environments in a human-readable format. Other utilities, such as the ROS 2 Record & Replay application, facilitate efficient message logging and playback, and the ROS 2 Router simplifies the connection of distributed environments. Collectively, these tools improve introspection, debugging, and overall system management.

It also aims to improve the development experience for robotics applications. Among its key objectives is the enhancement of ROS 2 documentation, achieved through the addition of detailed tutorials, practical examples, and diverse use cases. This commitment to improving accessibility and usability ensures that developers can more effectively leverage ROS 2 potential while promoting the adoption of open-source robotics solutions.

2.3.1 Vulcanexus Metapackages

Vulcanexus is organized into meta-packages, with each meta-package tailored to address a specific use case. A meta-package in the context of ROS 2 and Vulcanexus is essentially a package that groups together multiple related packages to simplify their management and distribution. It acts as a collection point, ensuring that all the included packages are installed and work seamlessly as a unit. Metapackages are particularly useful in environments like ROS 2, where developers often rely on a diverse set of interdependent libraries and tools targeting and common use case.

The structure of a meta-package typically includes:

1. A package manifest (e.g., package.xml) describing the dependencies and contents.
2. References to all the individual packages it encompasses.
3. A streamlined way to install, update, or remove a set of tools and libraries.

Vulcanexus employs meta-packages to organise its rich suite of tools and features into cohesive units tailored to different use cases. Examples include:

- Vulcanexus Core: ships core libraries for robotics applications.
- Vulcanexus Tools: focuses on helping developers with the introspection and debugging of their systems.

- Vulcanexus Cloud: targets the deployment of DDS entities in the cloud and distributed environments.
- Vulcanexus Micro: oriented to the development and deployment of embedded ROS 2 applications.
- Vulcanexus Simulation: provides the tools for robot modelling and simulation.
- Vulcanexus Base: installs the Tools, Micro and Cloud components jointly.
- Vulcanexus Desktop: installs all of the above packages besides [ROS 2 Desktop package distribution](#). ROS 2 Desktop provides additional visualization tools, examples, demos, and tutorials. This is the most complete Vulcanexus installation and it is intended for developers that want a better understanding of the ROS 2 ecosystem.

By categorizing functionalities in this way, Vulcanexus improves modularity and usability, enabling developers to easily integrate the specific components they need.

The ARISE project aims to deliver a new Vulcanexus meta-package designed to facilitate the installation and deployment of the results developed within Work Package 2 (WP2). Creating a meta-package to consolidate all results from WP2, mainly Tasks T2.1 and T2.2, offers numerous advantages that enhance the utility and impact of the ARISE project. By centralizing the tools, libraries, and deliverables from WP2 into a single package, the meta-package simplifies accessibility for users and stakeholders. Instead of managing multiple components separately, users can install and utilize the entire suite of WP2 developments with ease. This approach not only saves time but also reduces the complexity associated with deploying the tools across different systems and environments.

Furthermore, the meta-package facilitates collaboration among ARISE participants and external stakeholders. With all WP2 outputs conveniently bundled, collaborators can quickly adopt and test the tools, enabling broader usage and feedback. This shared access fosters a collaborative environment that promotes continuous improvement and innovation within the scope of the project.

Additionally, the new Vulcanexus meta-package aligns with ARISE's commitment to open-source principles. By providing an accessible, well-documented, and cohesive package, the project encourages the reuse and adaptation of WP2 results within the broader robotics and human-robot interaction (HRI) communities. This openness not only broadens the impact of the project but also positions ARISE as a key contributor to advancing open-source technologies in the field.

In conclusion, Vulcanexus builds upon and significantly enhances the ROS 2 ecosystem by offering a richer set of tools, improved middleware performance, and support for advanced use cases such as cloud integration and simulation. It provides developers with an all-in-one solution that streamlines the creation and management of robotics applications, making it a superior choice compared to the vanilla ROS 2 distribution. For projects that require advanced communication, distributed setups, or resource-constrained device support, Vulcanexus offers unparalleled advantages, ensuring a more efficient and effective development process.

2.3.2 Release Procedure

Vulcanexus follows a structured release cycle designed to ensure continuous improvement, compatibility with technological advancements, and alignment with user needs. This cycle includes annual major updates, alongside periodic minor and patch releases, to maintain a robust and up-to-date ecosystem.

Major releases of Vulcanexus are synchronized with the annual major releases of ROS 2. For instance, the most recent ROS 2 Jazzy release was accompanied by a corresponding Vulcanexus Jazzy release. A key distinction between the standard ROS 2 release and the Vulcanexus release lies in the version of Fast DDS included. Specifically, while ROS 2 Jazzy incorporates Fast DDS v2.14, Vulcanexus Jazzy ships with Fast DDS v3. This upgraded middleware version introduces enhanced features and bug fixes, improving the middleware layer, which is a critical component of the ROS 2 distribution.

In addition to major updates, Vulcanexus regularly publishes minor releases that integrate the latest advancements in middleware and associated eProsima tools. These updates ensure that the most stable and feature-rich version of Fast DDS is always available within the ROS 2 ecosystem.

Long-Term Support (LTS) versions of Vulcanexus are released biennially, with intermediate shorter-term versions offering additional flexibility for developers in planning and executing their projects. This structured and proactive approach to releases makes Vulcanexus's commitment to providing users with state-of-the-art features and tools, ensuring a continuously evolving and reliable platform.

3 Vulcanexus Jazzy (v4.0.0)

Vulcanexus Jazzy (v4.0.0) is the first release of Vulcanexus under the ARISE project. This release is designed to implement the primary features outlined in Task T2.1 of Work Package 2 (WP2), addressing critical enhancements to ROS 2. These features include:

1. The improvement and integration of dynamic types functionality within ROS 2 to establish this capability as a distinctive advantage of Vulcanexus over standard ROS 2 distributions.
2. The exposure of the "keys" feature inherent to DDS within ROS 2, addresses scalability challenges by mitigating the proliferation of topics. While this functionality is a fundamental aspect of DDS, it was historically omitted from ROS 2 due to the absence of an equivalent concept in ROS 1, leading to notable scalability issues.
3. Enhancements to ROS 2's discovery mechanism, aimed at optimizing performance and reliability in environments characterized by lossy networks.

These advancements collectively aim to strengthen Vulcanexus's role as an improved and scalable middleware solution within the ROS 2 ecosystem. The current status of each feature will be explained in the following sections.

The materials included in the Vulcanexus Jazzy release are as follows:

- Publicly available Docker images hosted on DockerHub. See <https://hub.docker.com/r/eprosima/vulcanexus/tags?name=jazzy>.
- Debian packages for streamlined installation and integration. See <http://repo.vulcanexus.org/debian/dists/noble/>.
- Snap packages for enhanced software distribution and deployment. See <https://snapcraft.io/publisher/eprosima>.
- Comprehensive documentation and tutorials to support user adoption and implementation. See <https://docs.vulcanexus.org/en/jazzy/>.

The release notes for Vulcanexus Jazzy (v4.0.0), including a comprehensive list of the included packages, are available at the following link:

<https://docs.vulcanexus.org/en/jazzy/rst/notes/jazzy/notes.html#notes-jazzy-latest>

3.1 Fast DDS v3

The integration of Fast DDS v3 into Vulcanexus Jazzy represents a significant advancement for ROS 2 applications, offering enhanced performance, scalability, and feature support. Fast DDS, developed by eProsima, serves as the default middleware for both Vulcanexus and ROS 2, facilitating efficient communication in distributed robotic systems. However, Vulcanexus

distinguishes itself by incorporating the latest Fast DDS v3, whereas the standard ROS 2 distribution includes an earlier version.

Incorporating Fast DDS v3 into Vulcanexus Jazzy enables ROS 2 developers to leverage Fast DDS's latest improvements, resulting in improved system performance and scalability.

For a detailed overview of the features and improvements in Fast DDS v3, please refer to the official release notes: [Fast DDS v3 Release Notes](#).

3.1.1 Performance improvements

While Fast DDS v3 primarily introduces new functionalities, it also incorporates key performance enhancements.

To improve efficiency and reduce latency in lossy networks, the RTPS message creation process has been optimized through the implementation of the "**Gather Send**" feature. This approach eliminates redundant data copying between the RTPS and transport layers, significantly enhancing throughput, particularly for high-data-rate applications like video streaming.

RTPS messages are now constructed using a list of instances from a new class, `NetworkBuffers`, which primarily consists of a pointer and a size. This list serves as the new data transmission object between the transport and RTPS layers. The adoption of `NetworkBuffers` required updates to several components, including sender resource lambda functions, transport interfaces, the statistics module, and RTPS message creation logic. By leveraging this approach, runtime memory copies (*memcpy*) are minimized, leading to improved performance.

Additionally, the `SendBuffersAllocationAttributes` structure has been extended to include a new attribute that defines the allocation configuration for `NetworkBuffers`, providing further flexibility and control over buffer management.

The implementation of this feature was developed in the following pull requests:

- Gather Send (<https://github.com/eProsima/Fast-DDS/pull/4537>):

This PR contains the main implementation of the feature. It refactors the creation and sending of RTPS Messages by utilizing a list of `NetworkBuffers` (pointers + sizes) as the data transmission object. It impacts both the Transports and the RTPS Layer, involving updates to the sender resources lambda functions, transport interfaces, statistics module, and the logic implemented in RTPS message creation.

- IPayloadPool Refactor (<https://github.com/eProsima/Fast-DDS/pull/4892>):

This PR refactors internal classes that are used in Fast DDS to prepare the Gather Send feature.

3.2 X-Types support in ROS 2

As previously mentioned, any enhancements made to Fast DDS directly impact Vulcanexus, as Fast DDS serves as the default middleware for both Vulcanexus and ROS 2. However, Vulcanexus distinguishes itself by incorporating Fast DDS v3, a version not currently available in standard ROS 2 distributions. This distinction provides Vulcanexus with significant advantages through improved features and functionality.

One of the key advancements introduced in Fast DDS v3 is the extensive refactoring of XTypes (Extensible and Dynamic Topic Types). This refactor aligns with the implementation of the OMG XTypes v1.3 specification, resolving legacy issues that previously hindered its adoption in production systems. The improved XTypes support extends its applicability across a broader range of scenarios, including ROS 2 applications, by introducing compatibility with data types that were not previously supported. These include maps, nested structs, unions, and enums, significantly enhancing the middleware's flexibility and robustness for handling complex data structures.

In the current version of Vulcanexus, ROS 2 integrates support for the OMG XTypes specification, which introduces several important concepts:

1. **A Type System with Extensibility:** The XTypes specification defines a flexible type system that allows data types to evolve over time while maintaining backward compatibility.
2. **Type Representation:** It supports both Interface Definition Language (IDL) and TypeObject representations, ensuring comprehensive and interoperable type descriptions.
3. **Data Representation Over the Wire:** XTypes standardizes how data is serialized for transmission, ensuring consistent and reliable communication across distributed systems.
4. **Language Binding:** XTypes defines both plain and dynamic language bindings. eProsima's Fast DDS-Gen facilitates this process by generating plain language bindings from IDL-type representations. However, the dynamic language binding—requiring a specialized API—is a unique feature of the Fast DDS library that cannot be directly exposed to ROS 2 users.
5. **Remote Type Discovery:** One of the most impactful features of XTypes in ROS 2 is its built-in mechanism for the automatic discovery of remote data types. This capability allows applications to dynamically subscribe to and receive data from ROS 2 systems without prior knowledge of the specific data types being used.

The Remote Type Discovery feature is particularly significant as it enables interoperability between ROS 2 and external systems. For example, eProxima leverages this functionality in its DDS Enabler, a bridge that facilitates seamless data exchange between the FIWARE Context Broker and ROS 2 networks. By utilizing this feature, external applications can ingest from and feed data into ROS 2 ecosystems without requiring manual configuration or type definitions.

This comprehensive integration of XTypes in Vulcanexus provides developers with a powerful and flexible framework for building scalable, interoperable, and efficient distributed robotic systems.

3.3 DDS Keys in ROS 2

One of the core concepts involved in the Vulcanexus communication scheme is *topics*, which serve as the bridge that enables data exchange among nodes within a particular system.

Topics are named communication channels that allow nodes (software modules) in a distributed system (such as a robotic application) to exchange messages with each other. These messages can contain various types of data, such as sensor readings, control commands, or status updates, enabling different parts of the robot to collaborate and share information.

The following image depicts a simple scenario where two sensors publish data on different topics, and two remote nodes subscribe to it. The first Logging node only subscribes to the Sensor 2 output topic, while the Controller node subscribes to both topics:

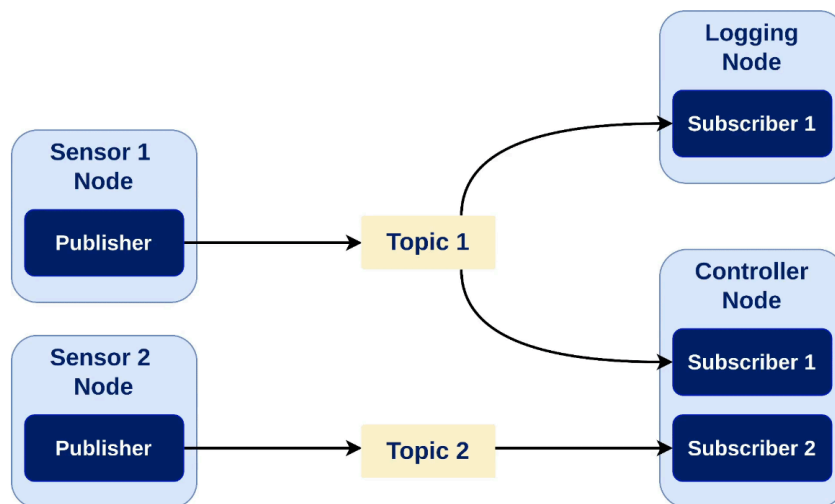


Figure 1: Topics in ROS 2

When any new data is available in the sensors, it is transmitted through the corresponding topic and all the subscribers attached to that topic receive the message.

In this situation, each of the sensors has a dedicated topic for transmitting its data. This suggests excessive resource usage because the middleware has to create and manage additional entities (topics, publishers, and subscriptions) and can potentially lead to a discovery overhead as the number of sensors scales up.

While standalone topics allow nodes to exchange messages freely, keyed topics add logic of organization to this communication process.

In a keyed topic, every message is associated with a topic instance and each topic instance is identified by a unique key. This key allows nodes to update different states of the same kind. In this sense, keys can be thought of as the "primary key" of a database. While a topic's message is used to represent the state of a real-world object, each unique key corresponds to the specific topic instance to which the topic's message belongs.

Following the example above, the next image illustrates this idea, where the sensors publish data with a unique key (sensor ID) corresponding to the object being observed:

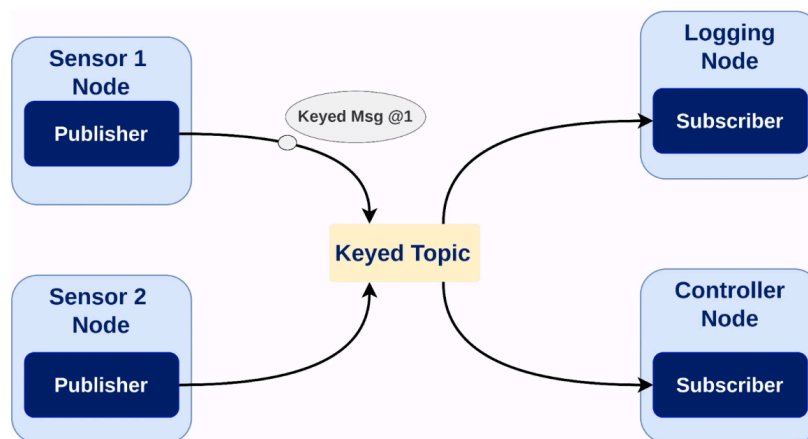


Figure 2: Keyed topic

In this case, only one topic and subscriber per node is required to receive the data of interest.

Conceptually, topic instances are a way of multiplexing the transmission of updates of several objects of the same logical kind over the same resource, i.e. the topic. Imagine a scenario where multiple robots are exploring an uncharted territory and are sending their positions to a central control node. Instead of having multiple topics, one per robot, topic keys allow to have a single topic where each robot sends its status with a unique key (the robot id) that identifies the robot. That is the interest of topic instances.

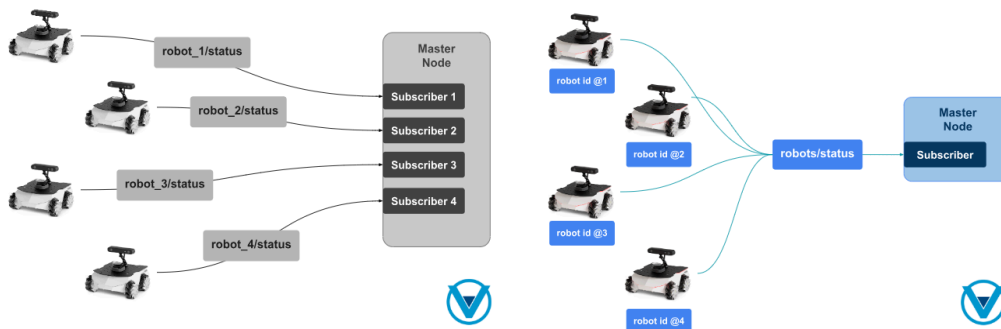


Figure 3: Topic instances example

3.3.1 Benefits of keyed topics

The use of topic keys reports several benefits, including:

- Efficient middleware infrastructure and data distribution: the use of keyed topics reduces the number of entities (subscriptions, publishers, and topics) needed in the data flow of the application which entails a more efficient usage of resources and bandwidth.
- A better history management: Fast DDS maintains a cache of recent updates, typically the last 10 updates for each instance. This caching mechanism, configured separately for publishers and subscriptions, prevents instances with frequent changes from overriding the latest value of another instance that changes less frequently. Moreover, this per-instance cache can be managed by the source application or a persistence service for late-joining subscribers, ensuring they initialize with the current value or recent changes of each object. Without keyed topics, the middleware cannot intelligently cache data on a per-instance basis, resulting in late joiners needing to process a larger history before accessing the current value of a specific instance.
- Easy integration with databases: individual records in a table are uniquely identified by the values of the fields marked as the "primary key" for that table. If those same fields are also marked as the key fields for the corresponding Topic, then the integration is seamless and can work hand-in-hand with the database table storage.
- Improved message filtering: by combining topic keys with Content Filter Topic, nodes have the ability to subscribe selectively to messages by utilizing their keys, allowing for precise filtering and minimizing superfluous message handling. Please consult the Topic Instances Subscription Filtering Tutorial for practical guidance.

In summary, keyed topics enhance the flexibility, efficiency, and organization of message exchange. By leveraging keyed topics, developers can design more robust and scalable robotic applications capable of handling diverse data sources and complex environments.

3.3.2 Repository contributions for the feature

The core updates for supporting the topic keys feature comprise the following packages:

- `rosidl`: new functionalities for the parser and dynamic TypeSupport.
- `rosidl_typesupport_fastrtps`: serialization mechanisms in the static pipeline generation.
- `rmw_fastrtps`: adds the integration of the feature within the ros middleware layer.

Together, these modifications provide improved performance and greater flexibility for developers, enhancing the overall scalability of ROS-based applications.

3.3.2.1 rosidl

A message data type can be defined as *keyed* using annotations in the proper *.idl* message file. Changes in the `rosidl` meta-package introduce proper processing in the parser and add the information of whether a particular member is part of the key to the introspection TypeSupport:

- Adds utility function `has_any_member_with_annotation` to `rosidl_parser`.
- `rosidl_typesupport_introspection`:
 - Adds the member `is_key_` to the member descriptor.
 - Adds the member `has_any_key_member_` to the structure descriptor.

Pull Request (ROS 2): <https://github.com/ros2/rosidl/pull/796>

3.3.2.2 rosidl_fastrtps_cpp

Changes in this package include the introduction of new callbacks for performing operations such as serialization, deserialization or obtaining sizes of the message keys within the static message generation pipeline.

Pull Request (ROS 2): https://github.com/ros2/rosidl_typesupport_fastrtps/pull/116

3.3.2.3 rmw_fastrtps

The ROS 2 framework stack adds two more layers on top of the middleware. These layers are Ros Client Library (RCL) and Ros Middleware Layer (RMW). To support topic keys, proper changes were made to the `rmw_fastrtps` repositories that:

- Specializes the necessary methods for keys in the Fast DDS TypeSupport, acting as a proxy of those methods with the key callbacks defined in the message data type.
- Configures the Fast DDS middleware to use proper Quality of Service when using keys.

Pull Request (eProsima): https://github.com/eProsima/rmw_fastrtps/pull/33

3.4 ROS 2 Discovery Mechanisms

Discovery Server is an alternative discovery mechanism based on a client-server topology, in which a server DomainParticipant operates as the central point of communication. It collects and processes the meta-traffic sent by the client DomainParticipants, and then distributes the appropriate information among the rest of the clients.

Discovery Server also implements a filter feature that allows for further reduce the number of discovery messages sent by using the topic of the different nodes to decide if two nodes must be connected, or they could be left unmatched. Unlike conventional discovery methods that rely on multicast, the Discovery Server eliminates the need for multicast. This approach significantly reduces discovery-related network traffic and offers an efficient solution for environments with stringent network requirements.

In previous versions of Fast DDS Discovery Server, connecting to any server required specifying its Server GUID. This approach resulted in tedious configurations and limited scalability. To streamline the process, Fast DDS introduced the "Server ID" parameter, which mapped each Server ID to a fixed GUID. This enabled clients to connect to servers by simply specifying the Server ID, eliminating the need to manually input the entire GUID. While this feature enhanced user experience, it still required *prior* knowledge of the network, thereby limiting scalability.

To address these challenges, Fast DDS v3, included in Vulcanexus Jazzy, introduced significant improvements focused on user experience and reliability in lossy networks. The updated Discovery Server eliminates the requirement for GUIDs or Server IDs, allowing clients to connect to servers using only an IP address and port. This enhancement simplifies client-server connections and facilitates the deployment of multiple servers without concerns about GUID uniqueness, thereby improving the scalability and manageability of Discovery Server networks.

The new Discovery Server also employs a mesh topology to interconnect servers. Newly discovered servers receive information about other servers in the mesh, enabling them to establish their own connections automatically. This replaces the manual configuration required in earlier versions. The mesh topology offers two key advantages:

1. **Improved communication speed:** Each server in the mesh directly broadcasts announcements to all other servers.
2. **Simplified network expansion:** Adding a new server only requires connecting it to one existing network member, after which it automatically discovers other servers. Furthermore, the remote server list can be dynamically updated at runtime without restrictions, allowing new connections to be added without interrupting existing ones.

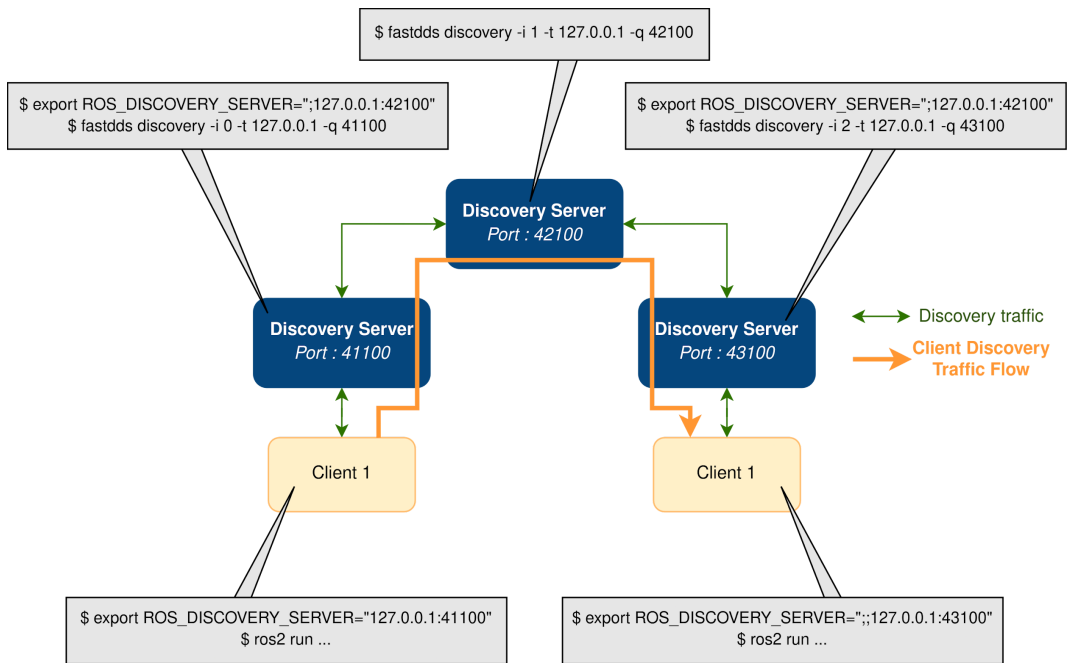


Figure 4: Traffic flow in Discovery Server v2

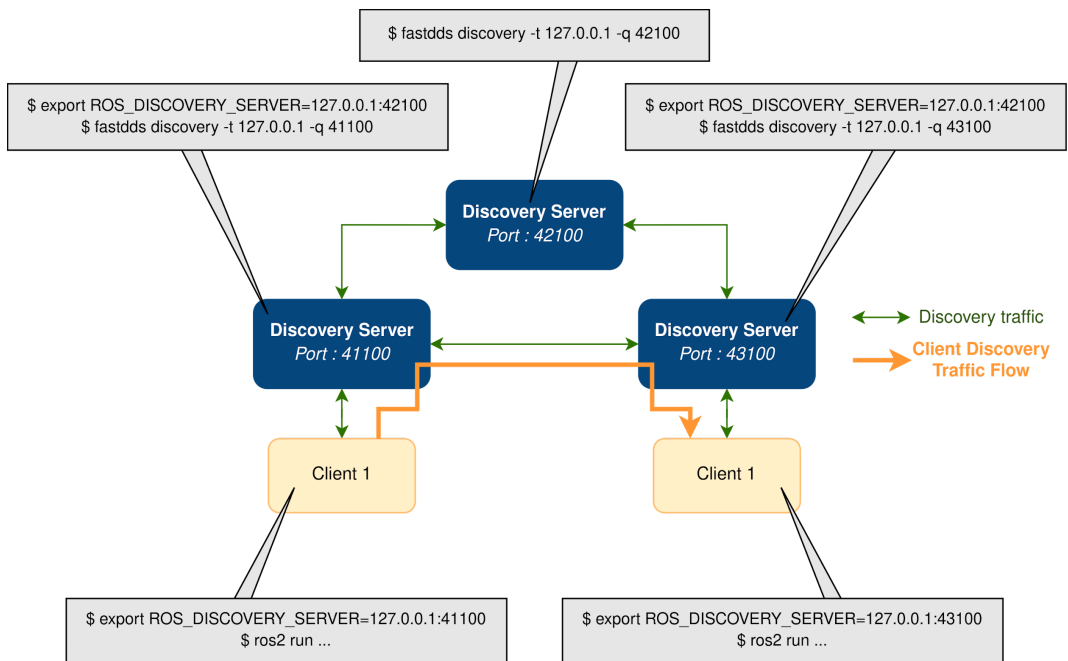


Figure 5: Traffic flow in Discovery Server v3

Additionally, the new version is fully compliant with the TCP protocol, which is more reliable in lossy environments, such as WiFi networks, due to its connection-oriented architecture.

Clients can combine Shared Memory transport with TCP to ensure optimal performance and reliability between ROS2 Nodes. It is also fully compatible with X-types.

The implementation of this feature was developed in the following pull requests:

- GUID-Less Discovery Server (<https://github.com/eProsima/Fast-DDS/pull/4716>):

This PR introduces significant changes to the Discovery Server in Fast-DDS to operate without a fixed GUID. It allows servers and clients to connect using only locators, implements a mesh topology for server connections, and updates the server connection list at runtime without restrictions. The PR also deprecates certain parameters and modifies the server and client discovery behaviour for improved connectivity.

- Support Type-Lookup with DS (<https://github.com/eProsima/Fast-DDS/pull/4768>):

This PR modifies the behavior of the TypeLookup Service to ensure compatibility with the Discovery Server. Participants issue the TypeLookupRequest to the participant that sent them the EDP data, rather than to the participant containing the endpoint. This adjustment allows servers to respond to their clients, thereby keeping discovery traffic limited to servers.

- Refactor DS example (<https://github.com/eProsima/Fast-DDS/pull/4913>):

This PR refactors the DiscoveryServerExample with a new format and uses the new GUID less feature.

- Fix DS TCP port logic (<https://github.com/eProsima/Fast-DDS/pull/4941>):

This PR solves small issues related to logical ports in Discovery Servers.

- Backup DS with X-Types (<https://github.com/eProsima/Fast-DDS/pull/4740>):

This PR fixes a bug in the Backup Server initialization.

- Fast DDS Python (<https://github.com/eProsima/Fast-DDS-python/pull/147>):

This PR makes the Fast DDS Python repository compatible with the changes of the new Discovery Server.

3.5 Tutorials and Use Cases

In the latest version of Vulcanexus, eProsima has introduced a comprehensive set of tutorials and use cases aimed at addressing the most common challenges faced by ROS 2 developers when building distributed robotics architectures using DDS. These resources are designed to provide practical guidance and solutions for frequent issues, including:

1. **Vulcanexus WiFi and Large Data Tutorials.** This series of tutorials focuses on resolving typical problems encountered when deploying ROS 2 over WiFi networks. These challenges include latency, packet loss, and bandwidth limitations, which can adversely affect the performance and reliability of applications. The tutorials provide strategies to mitigate these issues effectively.
 - a. [1.1.1. How to solve wireless network issues in ROS 2](#)
 - b. [1.1.2. How to handle large data video streaming in ROS 2](#)
 - c. [1.1.3. How to compress images for video streaming in ROS 2](#)

2. **Scalability and Large-Scale Networks Tutorials.** This series is tailored to assist developers in managing the complexities of using ROS 2 in large-scale networks. Factors such as latency, packet loss, and network congestion are addressed, with the tutorials offering insights into maintaining the functionality and performance of distributed systems in such environments.
 - a. [1.2.1. How to Use a Discovery Server to Optimize ROS 2 Node Discovery](#)
 - b. [1.2.2. How to Use DDS Router for Scalable ROS 2 Network Communication](#)

These additions aim to enhance the development experience and provide robust support for creating reliable and scalable robotics systems with ROS 2.

3.6 Benchmarking of Vulcanexus over “plain” ROS 2

This section provides an analysis of the superior quality of Vulcanexus in comparison to the standard ROS 2 framework. Based on the results and insights presented in the preceding sections, this analysis highlights Vulcanexus's advanced capabilities and advantages.

First, Vulcanexus framework, based on Fast DDS v3, offers several key advantages over the standard ROS 2 implementation:

- **Enhanced Middleware Integration:** Vulcanexus integrates the latest version of Fast DDS, which includes features such as dynamic types (XTypes), keyed topics, and optimized discovery mechanisms. These features are not fully realized in the plain ROS 2 distribution.
- **Advanced Development Tools:** Vulcanexus includes unique tools such as the ROS 2 Monitor, ROS 2 Spy, and ROS 2 Router for introspection, debugging, and efficient network communication. These tools enhance the developer's ability to manage and scale distributed systems effectively.
- **Modular Architecture:** The meta-package structure of Vulcanexus, including specialized packages like Vulcanexus Core, Tools, Cloud, and Micro, allows developers to customize their installations for specific use cases.

Moreover, performance benchmarking focuses on scalability and discovery in challenging environments, including lossy networks. With support for keyed topics and optimized discovery mechanisms, Vulcanexus minimizes resource overhead and enhances system

scalability. In contrast, plain ROS 2 suffers from discovery and topic proliferation issues as the system size increases.

Vulcanexus' use of the Fast DDS Discovery Server, particularly in its mesh topology, significantly reduces discovery-related traffic and ensures reliable communication in environments with high packet loss. Benchmarking results show up to a ~90% reduction in discovery traffic compared to plain ROS 2 (see [Discovery Server article](#)).

Then, the "Gather Send" feature in Fast DDS v3 enhances data throughput by reducing redundant memory operations, ensuring consistent performance in high-data-rate applications.

These results can be considered as the initial outcomes of Task T2.2, which is focused on measuring the performance superiority of Vulcanexus over the "plain" ROS 2 distribution. Future deliverables will emphasize testing Vulcanexus and plain ROS 2 under real-world scenarios to substantiate the analyses presented here. The findings from these comparative studies will form an integral part of subsequent Vulcanexus releases, reinforcing the ARISE project's commitment to advancing industrial human-robot interaction. By disseminating benchmarking results and use-case studies, the project aims to foster the adoption of Vulcanexus as the standard middleware solution within the ROS 2 ecosystem.

4 Future release

This section provides an overview of the key features, tools, and enhancements that will be included in the forthcoming release of Vulcanexus Jazzy (v4.1.0).

4.1 DDS Enabler

eProsima's [FIWARE DDS Enabler](#) serves as a single point of entry for DDS communication, managing all necessary DDS participants and efficiently transmitting the DDS data published into a DDS environment to the FIWARE Context Broker, as well as allowing to inject data from the Context Broker into a DDS network.

DDS Enabler supports extensive configuration options through a YAML file. This allows users to easily fine-tune various DDS settings to meet their specific requirements, including the management of topic visibility (by means of filtering expressions) for granular control over data distribution.

DDS Enabler can be configured so that DDS topics, data types and entities are automatically discovered without the need to specify the types associated to the data being exchanged. This capability is facilitated by the [XTypes](#) functionality supported in eProsima Fast DDS, a C++ implementation of the DDS (Data Distribution Service) Specification developed by the Object Management Group (OMG).

The underlying core of DDS Enabler is [eProsima DDS Pipe](#), a foundational component utilized across various eProsima products to facilitate seamless communication between different applications within a DDS (Data Distribution Service) environment. It efficiently handles the creation and discovery of DDS participants, ensuring smooth and reliable communication with the DDS ecosystem. The module ensures that payloads are efficiently transmitted, minimizing latency and maximizing throughput.

In order to facilitate the translation of DDS data into a more exchangeable and manipulation-friendly format, several [serialization utilities](#) were implemented. These include the DDS data to JSON converter, as well as a DDS type serializer into readable Interface Definition Language (IDL) strings. Ongoing is the development of a JSON to DDS data deserializer that will allow to injection of data from the Context Broker into a DDS network.

4.2 ROS 2 nodes new default configuration

eProsima's Discovery Server mechanism effectively removes the need for multicast discovery traffic and is a flexible solution for different network topologies, proposing a centralized discovery approach that requires users to launch additional Discovery Server processes. Despite this fact being seen as a drawback, the ROS community (especially the ones migrating from ROS 1) is already accustomed to having a centralized discovery entity (e.g. ROS Master). Additionally, ROS 2 already has a background process, especially for running CLI commands, the ROS 2 daemon.

With the idea of improving the ROS 2 user experience, a new default discovery mechanism based on Discovery Server called **Discovery Server Auto Mode** is being developed within Vulcanexus Jazzy. The new mode uses the [GUID-Less feature](#) and several improvements to the Fast DDS CLI and the `ROS_DISCOVERY_SERVER` environment variable to simplify the use of the Discovery Server and provide easier and faster ways to handle network deployments.

The deployment of the new feature will be done using the already existing `ROS_DISCOVERY_SERVER` and `ROS_STATIC_PEERS` environment variables. A new value for the `ROS_DISCOVERY_SERVER` environment variable will be added (AUTO), which will enable the new Discovery Server Auto Mode. The `ROS_STATIC_PEERS` environment variable will be interpreted as the remote server list for the Discovery Server, allowing the user to define remote connections during initialization.

The design comprises two main streamlines:

1. Adding new functionalities inside Fast DDS to deal with the automatic launch of the server and the new values for the `ROS_DISCOVERY_SERVER` and `ROS_STATIC_PEERS` environment variables.
2. The extension of the already existing Fast DDS Discovery CLI Tool to manage discovery servers' lifetime, configuration and connections.

Whenever a ROS 2 node is launched with the environment variable `ROS_DISCOVERY_SERVER` set to AUTO, Fast DDS will automatically launch the Discovery Tool in the background.

Every ROS 2 node (client) will only point to the local Discovery Server in the same DDS Domain. The Fast DDS Discovery tool will ensure that exactly one Discovery Server is running in the same DDS Domain, by assigning fixed ports to each domain. Communication across DDS domains and external hosts is done in a Server-to-Server fashion. So, if the `ROS_STATIC_PEERS` is empty, there will be no communication between different DDS Domains. However, if the `ROS_STATIC_PEERS` is set, a Discovery Server in a given domain will be able to discover another Discovery Server in a different domain. The same reasoning applies whenever users desire to connect to external machines in LAN or WAN, filling the `ROS_STATIC_PEERS` will be required.

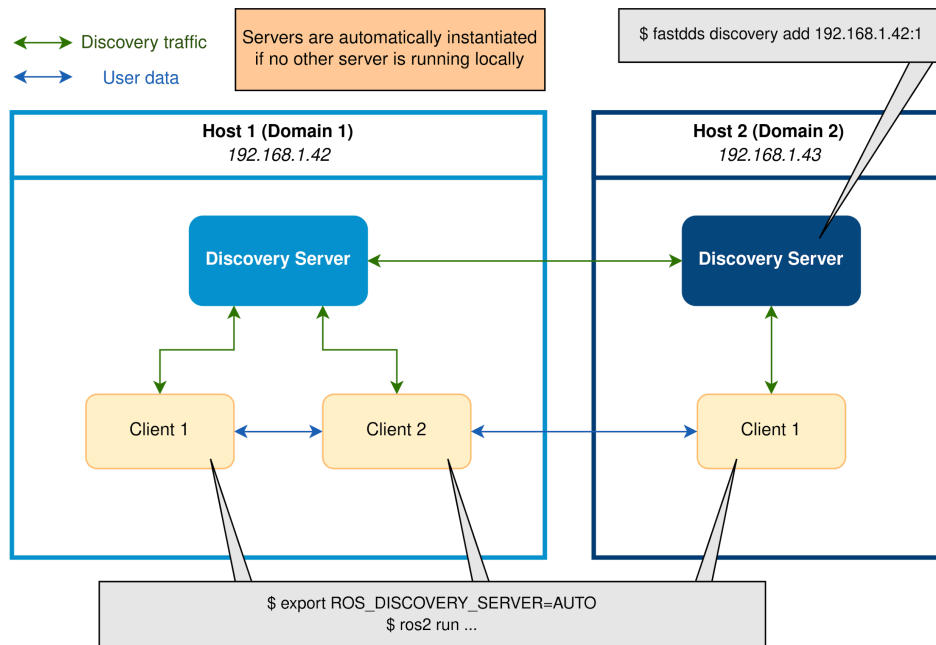


Figure 6: Launch Schema for **Discovery Server Auto Mode**

Additional changes will be made in the Fast DDS Discovery CLI Tool to provide new features and functionalities. The following list describes the new keywords that will be added and that will be available using 'fastdds discovery <keyword>':

- **auto**: It will check if an active server exists in the domain specified. If there is no active server, it will create one. If there exists an active server, it will simply output an informative log. The server instantiated will use SHM and TCP transports and it will be located in the port calculated using the well-known ports logic assignment.
- **stop**: It will stop a running server in the specified domain. If there is no server running it will simply output an informative log. An additional 'stop all' command will be available to kill all running Discovery Server instances.
- **add**: It will add new remote servers to the running server in the specified domain. It shall accept a remote server list as an additional argument in the CLI. The format of the remote server list should follow the guidelines of the `ROS_DISCOVERY_SERVER` environment variable to indicate servers (IP:Port). Additionally, it will be possible to name Servers directly from their domain, using an "IP:Domain" pair.
- **set**: It will completely replace the remote servers list of the corresponding server with a new list. This keyword needs to be used if at least one active remote connection needs to be shut down. It will kill the server and start it again with the new configuration. The same port will be used.
- **list**: It will inspect all active local servers and return an output log. The log will be in a list format and it will only show the domain, address and port of each server.

All servers automatically instantiated or using the CLI Tool with the 'auto' or 'start' keyword will use both Shared Memory and TCP transports.