EKUMEN

ROS 2™ Migration:

Paths, Pitfalls, and Why It's Urgent

ROS Noetic's support ended in May 2025, making migration to ROS 2 urgent. This expert-led guide explains why you need to migrate now, the best strategies to do it, common pitfalls to avoid, and practical steps to prepare your system. Get clear insights to ensure a smooth, efficient, and future-proof transition to ROS 2.

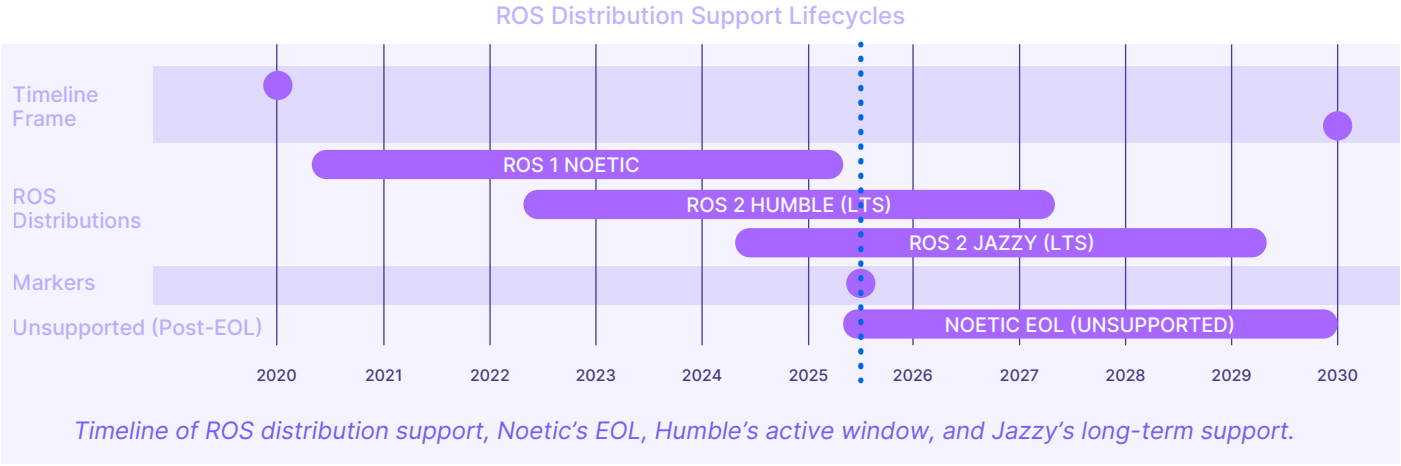
ROS to ROS 2 Migration

Authors:

Alexis Pojomovsky | Michel Hidalgo | Ivan Paunovic

Why migrate now?

ROS Noetic reached end of life in May 2025, ending official updates and security patches. Although it is technically possible to continue using ROS, doing so means your team would need to take full ownership of maintaining, building, and shipping your robotics stack independently. You can choose to stay on ROS for the foreseeable future, but this comes with the hidden cost of having to maintain every part of your stack alone, while losing the community backing and collective momentum that come with an actively supported ecosystem. For most teams, especially startups that need to focus on improving their product rather than maintaining infrastructure, staying on ROS is rarely a practical long-term choice.



Timeline of ROS distribution support, Noetic's EOL, Humble's active window, and Jazzy's long-term support.

In contrast, ROS 2 offers a stable, scalable, and secure framework with multi-robot support and a modern architecture better suited for production systems. Migrating is not just maintenance; it is a chance to modernize your stack, reduce technical debt, and align with current engineering practices for robotics.

Migration strategy summary

Migrating to ROS 2 requires a strategy aligned with your system's complexity, risk tolerance, and operational needs. Common approaches include:

ALL-AT-ONCE MIGRATION:
Rewrite and deploy the system on ROS 2 in one move.

PHASED MIGRATION:
Gradually migrate while maintaining partial ROS operation.

Each approach has trade-offs that require careful consideration. To de-risk either strategy, the complementary practice of parallel validation should be used to test the migrated system against a known baseline.

Choosing your migration strategy

All-at-Once Migration

This approach involves fully rewriting your system in ROS 2 before deployment, switching over in a single step. It allows:

- A clean architectural refresh, removing legacy workarounds and technical debt.
- Adoption of ROS 2 paradigms, including lifecycle nodes, deterministic execution, and modern launch practices.
- Alignment with supported ROS 2 distributions and tools from the start.

However, it also means:

- Long development periods without a working system.
- Higher integration and validation effort concentrated at the end.
- Significant upfront risk.

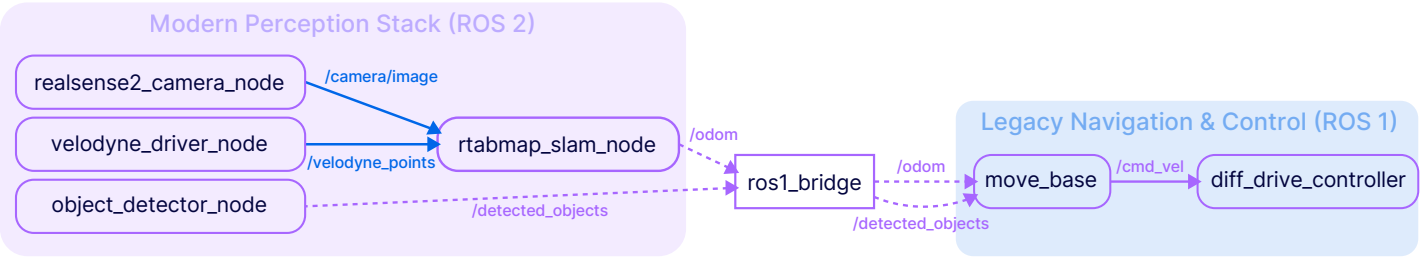
For most mid to large robotics projects, this approach is rarely practical due to the complexity of hardware integrations, cross-dependencies across subsystems, and operational uptime requirements. It is generally more suitable for smaller projects, early-stage products, or highly isolated systems where freezing ROS development during migration is feasible.

Phased Migration

Phased migration allows you to migrate incrementally while keeping your system operational, using the `ros1_bridge` for interoperability. This approach supports node-by-node, topic-by-topic, or container-based migration, gradually replacing parts of your system while maintaining continuous operation.

Containerizing ROS, ROS 2, or both during migration is also an option, as it can help isolate dependencies and reduce environment conflicts. However, it brings its own set of challenges related to networking, hardware access, and orchestration. For clarity, this section focuses on host-based installations. Still, most of the concepts discussed here can also be applied to a fully containerized strategy.

To apply phased migration effectively, it is important to carefully analyze the data flowing through your system's topics. The ros1_bridge can introduce latency and bottlenecks, especially when bridging high-bandwidth data streams such as camera feeds or point clouds. Ideally, topics carrying heavy data should be produced and consumed by nodes running on the same ROS version to avoid unnecessary overhead during migration.



Keep raw sensor data in a single version of ROS; bridge only lightweight topics.

At the same time, phased migration comes with trade-offs that teams should plan for:

ROS Noetic runs on Ubuntu 20.04, while current ROS 2 LTS distributions like Humble are officially binary-distributed under Ubuntu 22.04.

Building ROS 2 Humble on Ubuntu 20.04 from source is possible, and it's actually an officially supported target. It's also Tier 3, which means it doesn't get as much attention as other officially supported targets. This said, it could be a decent starting point given its LTS nature.

Using an older ROS 2 distribution like Galactic (already EoL as of July 2025) could also be a valid intermediate step during migration, provided it is treated as a temporary measure with a clear plan to transition to a supported ROS 2 distribution once the migration stabilizes.

Gazebo versions considerations

Migrating from ROS to ROS 2 often requires moving from Gazebo Classic to the newer Gazebo simulator, aligning your simulation stack with your modernization efforts. While there is official guidance on which Gazebo versions pair with each ROS 2 distribution, aligning these can become tricky if you target intermediate, source-built ROS versions during migration. Although this is typically less critical than your ROS version decisions, simulator alignment can still introduce friction if not planned for early.

To navigate this, consider the following:

- Most recent Gazebo versions, like Harmonic, align with current ROS 2 distributions and newer OS versions, but may be difficult to adopt if your system is anchored on Ubuntu 20.04 during migration.

- Using Gazebo Garden or Fortress can be a practical way to test and validate your ROS 2 components under Ubuntu 20.04 while your system is still in transition. However, Gazebo has seen significant fixes, new features, and performance improvements in its most recent versions. Targeting older versions may limit your access to these advancements and can introduce friction if your expectations are not aligned with the capabilities they have to offer.

- Treat the use of Garden or Fortress as a temporary measure, with a clear plan to transition to Gazebo Harmonic once your system is fully migrated to ROS 2 and operating on a newer OS environment.

These intermediate steps should be seen as pragmatic rungs on the migration ladder rather than final destinations. They allow you to maintain momentum while incrementally modernizing your stack, provided there is a clear plan to converge on fully supported distributions and tools once your ROS 2 migration is stable.

Teams adopting phased migration need to balance the operational benefits of incremental migration with the engineering effort and technical debt of maintaining intermediate states.

Parallel Validation

Parallel validation is a complementary approach that should be integrated with either an all-at-once or phased migration to de-risk the process. It involves establishing a robust set of end-to-end tests before porting to create a clear baseline of your system's expected behavior. This baseline helps validate that the new ROS 2 system achieves the same functionality.

A practical approach is to record real-world scenarios using ROS rosbags, then convert these bags to ROS 2 format. By replaying this data under ROS 2, you can exercise your system under the same conditions used for validation in ROS, enabling accurate comparisons of behaviors, outputs, and performance.

Potential advantages:

Creates a clear behavioral baseline before migration begins.	Consistent, repeatable testing using real-world data.	Reduces migration risk by detecting discrepancies early in the process.	Encourages the development of integration and regression tests as part of the migration strategy.

Things to keep in mind:

- This approach does not replace live testing with hardware and sensors, which is still necessary to validate your system end-to-end.

  Reliably implementing these tests is a difficult task. They can be prone to flakiness due to timing sensitivities or message transport inconsistencies, and they can break when system interfaces are modified.
- Changes in message definitions require you to update your tests and regenerate or convert existing rosbags, which adds maintenance overhead to the migration process.

Parallel validation is particularly valuable for high-reliability systems and teams seeking to de-risk their migration while continuing development and operations under ROS.

Preparing your codebase for migration

Regardless of which migration strategy you choose, the state of your current ROS codebase will significantly influence your migration's complexity and risk.

In many robotics systems, ROS-specific interfaces and core business logic are tightly coupled inside nodes, making it difficult to isolate what needs to change during migration. A practical, often overlooked approach is to refactor parts of your codebase before migration to separate ROS-specific concerns (nodes, message handling, service calls) from system business logic and algorithms.

ROS is much more than a communication layer, and in many systems, logic is often coupled to other ROS components beyond messaging. For example, `tf/tf2` is deeply tied to how coordinate frame transformations are handled within algorithms, and `actionlib` may drive goal-oriented workflows tightly integrated with system behavior. Additionally, it is common to see ROS message types used directly as general-purpose data structures within business logic, further entangling system code with ROS-specific dependencies. These dependencies do not always translate cleanly into ROS-agnostic structures, and forcing abstraction in these areas can introduce unnecessary complexity without clear migration benefits.

In practice, a straightforward pattern can help prepare your codebase for migration. Often, a ROS node that instantiates and uses a ROS-agnostic class is sufficient, keeping your system's core logic clean while handling all ROS communications, transforms, and other ROS-specific concerns within the node itself. The more your business logic remains ROS-agnostic, the less you will need to modify during migration, effectively reducing the surface of change, which is directly tied to the risk introduced by the migration process. This structure allows your ROS node to explicitly orchestrate communication and ROS integrations while keeping your business logic independent and reusable across different environments, including testing frameworks and the eventual ROS 2 system.

Adopting this preparatory structure can:

- Keep core logic fully unit-testable and stable throughout migration.
- Reduce the surface area of migration to primarily the ROS interface layers.
- Simplify porting efforts by clearly isolating what changes between ROS and ROS 2.

Investing in this preparation can de-risk your migration, regardless of whether you pursue an all-at-once or phased migration. It is especially valuable for large, long-lived robotics systems where stability and testability are critical.

Parallel Validation

ASPECT	ROS (NOETIC)	ROS 2 (HUMBLE/JAZZY)	IMPACT
Communication	TCPROS/UDPROS	DDS (default) or Zenoh	Requires RMW selection and QoS tuning
Discovery	Centralized (`roscore`)	Decentralized	Removes single point of failure
Launch	XML	Python (.launch.py), XML, YAML	Python enables dynamic, conditional launches with higher verbosity
Security	None	(optional) DDS Security	Encryption and authentication available
Parameter management	Global	Node-local, declared	Requires adapting parameter handling
Build system	`catkin`	`colcon` with `ament`	Workspace-based builds, advanced dependency management
Simulation	Gazebo Classic	Gazebo (AKA Gazebo Ignition), Isaac Sim, O3DE	Plan simulator updates as part of migration

About DDS and alternative RMWs

ROS 2 uses DDS by default for scalable communication, but it is not limited to DDS-based RMWs. Zenoh has recently emerged as an alternative middleware option, offering low-overhead, efficient communication, especially in wide-area or lossy networks. However, Zenoh is the new kid on the block within the ROS ecosystem and is officially supported starting from ROS Iron and newer distributions.

If your migration is taking place on ROS Humble or older, it might be advisable to start with a well-established DDS implementation such as CycloneDDS or FastDDS. While Zenoh is promising, adopting it prematurely on unsupported distributions may expose you to issues that have since been resolved in later ROS 2 and OS releases. At this stage, our goal is simply to raise awareness of Zenoh's existence for future evaluations, rather than to recommend it as the default choice during your migration.

Testing and CI for safe migration

A structured testing approach reduces migration risk and increases confidence in your system:

Unit tests for core logic using GTest (C++) and Pytest (Python).	Integration tests with `launch_pytest` or `launch_testing` to validate node interactions.	System tests using simulation and `rosbag2`-converted replays for ROS 2 validation with existing ROS data.	CI pipelines using GitHub Actions and `industrial_ci` for linting, building, and automated testing.

Pitfalls to avoid

Missing ROS 2 package equivalents during system audits, leading to delays later in the migration process.	Misconfigured QoS or middleware parameters causing inconsistent runtime behavior.	Underestimating the differences in ROS 2 parameter management, which can affect runtime assumptions.	Relying indefinitely on the `ros1_bridge` without a clear plan for decommissioning.

Migration checklist week 1

- [ ] Announce your migration plans internally to align stakeholders.
- [ ] Schedule ROS 2 training for your team.
- [ ] Set up a ROS 2 test environment
- [ ] Audit your ROS stack for dependencies and package availability.
- [ ] Test the ros1_bridge on a small subsystem.
- [ ] Select a low-risk node as a pilot migration candidate.
- [ ] Draft your migration plan with clear phases and goals.

Track progress with KPIs such as migration coverage, test pass rates, and pre/post-migration performance benchmarks to maintain momentum.

References

- ROS vs ROS2: Key differences, benefits, and why the future belongs to ROS2 - RoboticsBiz
- The ROS vs ROS 2 Transition - Southwest Research Institute
- Migrating Two Large Robotics ROS Codebases to ROS2 - InfoQ
- Migrating a large ROS codebase to ROS 2 - ROSCon
- Experiences with ROS 2 on our robots and what we learned on the way
- Discussion on ROS to ROS2 transition plan
- How to migrate a ROS project from ROS to ROS2
- launch_pytest: a framework for launch integration testing

contact@ekumenlabs.com

ekumenlabs.com