U.S. ARMY COMBAT CAPABILITIES DEVELOPMENT COMMAND – GROUND VEHICLE SYSTEMS CENTER

Evaluation of ROS2 Eloquent

Matthew Schickler

matthew.schickler@flir.com

AGENDA

OVERVIEW (30 minutes)

- Introduction
- Background
- User Survey
- Summary Findings
- Migration
- Training
- Conclusions
- Recommendations

DETAILED ROS2 FEATURE SURVEY (45 minutes)

Q&A (10 minutes)

INTRODUCTION

OBJECTIVE: Evaluate ROS2 Eloquent readiness for ROS-M and make recommendations for future action.

STRATEGY: Execute a targeted gap analysis:

- Compare core ROS2 functionality to what is offered in ROS1
- Perform a feature survey of ROS2 packages maintained by the ROS community:
  - In ROS1, almost 2500 separate packages exist so limit survey to packages of particular interest to ROS-M
  - As a coarse indicator of relative maturity, identify which packages are available as Debians from packages.ros.org versus which must be built from source
- Conduct a user survey among NAMC members to understand the current levels of ROS usage, plans for ROS2, and issues preventing greater ROS2 adoption
- Identify existing resources for ROS2 training and migration

BACKGROUND

- ROS1 has been under active development for over 12 years:
  - Extremely popular framework for robot software development
  - Primarily research, growing desire to use in production systems
- ROS2 is a ground-up redesign announced in 2014 with the goal of adding security, reliability, and real-time
- ROS2 uses industry-standard Data Distribution Service (DDS) middleware
- ROS2 API is substantially improved over ROS1, but changes are significant
- Migration from ROS1 to ROS2 is needed
- Five ROS2 distros so far, each adding back capability originally in ROS1
- Latest distro is Eloquent, Foxy coming May 2020

USER SURVEY

SUMMARY

- **8 questions asking if members are currently using ROS1 or ROS2, what features are most important to them, and what is required to increase adoption of ROS2**
- **17 respondents**

KEY TAKEAWAYS

- **71% of respondents are currently using ROS1**
  - 48% to a significant (50%) or greater extent; 6% are using it exclusively
- **41% of respondents are using ROS2**
  - 18% to a significant (50%) extent; none are using it exclusively
- **53% of respondents are quite to highly likely to start using ROS2 within the next year**
- **30% of respondents are quite to highly likely to start porting ROS1 code to ROS2 within the next year**
- **The lack of overall maturity of ROS 2 and key ROS1 features missing from ROS2 are the primary factors hindering adoption**
  - Most Important: a comprehensive set of device drivers, a motion planning framework (MoveIt!), and XML based launch files
- **63% of respondents indicated significant to high interest in online, self-paced ROS2 training materials**

SUMMARY FINDINGS

- As of ROS2 Eloquent, core ROS2 infrastructure is approaching feature parity with ROS1 but not there yet
- ROS2 Foxy might be point at which parity is achieved
- Completeness and maturity of ROS2 drivers and algorithms lagging behind core functionality
- Cartographer (3D SLAM) no longer supported by Google
- Still waiting for many ROS packages to be officially released (e.g. MoveIt2! Is in Beta)
- Training materials are still sparse but some forms like tutorials are growing
- More experience with specific ROS packages necessary to determine suitability for ROS-M projects

MIGRATION - PORTING FROM ROS1 TO ROS2

- Some major considerations:
  - CMakeLists.txt and package.xml changes
  - Re-design nodes to use Node base class
  - Re-design nodes for asynchronous operation
  - New API for nodes, timers, topics, services, actions
  - Convert Nodelets to Components
  - Convert dynamic reconfigure to parameter callbacks
  - Convert global parameters to local parameters

- More detailed conversion guide at:

  https://index.ros.org/doc/ros2/Contributing/Migration-Guide

- Goal of MARS project is to port RTK (large code base) from ROS1 to ROS2, should result in more detailed porting guidance for the community

MIGRATION - ROS1 BRIDGE

- Translates between ROS1 message protocol and DDS
- Drawbacks:
  - Bridge is a single point of failure
  - Extra CPU load due to message translation
  - Increased message passing latency
- In 2016 study, ROS2 bridge latency was found to only be about 500 microseconds

TRAINING

- ROS1 available training includes books, tutorials, references, and course material
- ROS2 lags behind in all of these areas but the number of ROS2 tutorials and demos is increasing
- No ROS2 books but some very basic online courses exist:

  https://www.udemy.com/course/ros2-how-to

  https://www.theconstructsim.com/robotigniteacademy_learnros/ros-courses-library/ros2-basics-course

- Robotis Turtlebot 3 e-Manual worth taking a look at:

  http://emanual.robotis.com/docs/en/platform/turtlebot3/ros2_bringup/

CONCLUSIONS - FEATURE GAPS

- General mechanism for setting level of individual loggers at runtime
- Bridging ROS1 actions to ROS2 actions (experimental implementation exists)
- Remote launch
- Missing control algorithms (e.g. differential drive, Ackermann)
- Missing or incomplete drivers (e.g. IMU, GPS)
- Missing or incomplete RQt plugins (e.g. console, logger levels, bag, diagnostics, TF tree)

CONCLUSIONS - QUALITY

- Most of the ROS2 software is relatively new compared to ROS1
- Significant amount of testing and bug fixing needed by ROS community to bring ROS2 to same level of quality as ROS1
- Using DDS helps but the middleware is only a small subset of overall software
- ROS2 is an open source project and quality will come mostly through bug reporting and fixing related to real-world use
- Bottom line: The more we use it, the better it will get

CONCLUSIONS - COMPLIANCE

- ROS2 is significantly improved over ROS1 in the areas of security and reliability due its use of DDS as its middleware
- However, DDS is NOT a silver bullet:
  - Different implementations may vary in terms of performance and compliance with the specification
  - Must be properly configured in order to achieve security benefits
- Real-time determinism is an important feature to meet reliability and safety requirements:
  - Real-time Operating System Support
  - ROS2 code must be real-time safe (currently not the case)
  - Need guidance for developers on how to make their code safe
  - Current status of Real-time Working Group (as of Feb 2020):
    https://discourse.ros.org/t/ros-2-real-time-working-group-online-meeting-10-feb-5-2020-meeting-minutes/12809

RECOMMENDATIONS

- Help port packages or capabilities identified as feature gaps
- Execute an updated middleware performance study
- Execute an updated DDS security analysis
- Create guidance related to configuring security in ROS2
- Participate in ROS2 real-time working group
- Execute targeted projects to migrate existing ROS1 code bases to ROS2 and report/fix bugs
- Until a viable alternative is identified for 3D SLAM, help support maintenance of Cartographer SLAM for ROS2
- Identify and support projects that use drivers and algorithm packages supplied by the ROS community with goal of evaluating the suitability of those packages for ROS-M projects

ROS2 FEATURE SURVEY

- **Build System**
- **Core**
  - Nodes
  - Communication
  - Middleware
  - Components
  - Launch
  - Parameters
  - Plugins
  - Logging
  - Transforms
  - Bonding
- **Algorithms**
  - Diagnostics
  - Controllers
  - State Estimation
  - SLAM
  - Navigation
  - Perception
  - Manipulation
- **Drivers**
  - CAN
  - GPS
  - IMU
  - LiDAR
  - Cameras
- **Tools**
  - Command Line
  - RQt
  - Rviz
  - Gazebo

BUILD SYSTEM

- CMake is still the fundamental build system for individual ROS2 packages
- In ROS1, the catkin build tool was used to build sets of inter-dependent packages:
  - Tools: catkin_make, catkin_make_isolated, catkin_tools
    CMakeLists.txt: use find_package(catkin …) and the catkin_package() command to define a ROS package
  - Package manifest file (package.xml)
- In ROS2, catkin has been replaced:
  - Tool: “colcon build”
    CMakeLists.txt: use find_package() for each individual ROS package and ament_package() to define a ROS package
  - New package manifest file format
  - colcon build output is aligned more closely with the output of a standard CMake build (e.g. there is no longer a “devel” directory)

ROS2 FEATURE SURVEY

- Build System
- **Core**
  - Nodes
  - Communication
  - Middleware
  - Components
  - Launch
  - Parameters
  - Plugins
  - Logging
  - Transforms
  - Bonding
- Algorithms
  - Diagnostics
  - Controllers
  - State Estimation
  - SLAM
  - Navigation
  - Perception
  - Manipulation
- Drivers
  - CAN
  - GPS
  - IMU
  - LiDAR
  - Cameras
- Tools
  - Command Line
  - RQt
  - Rviz
  - Gazebo

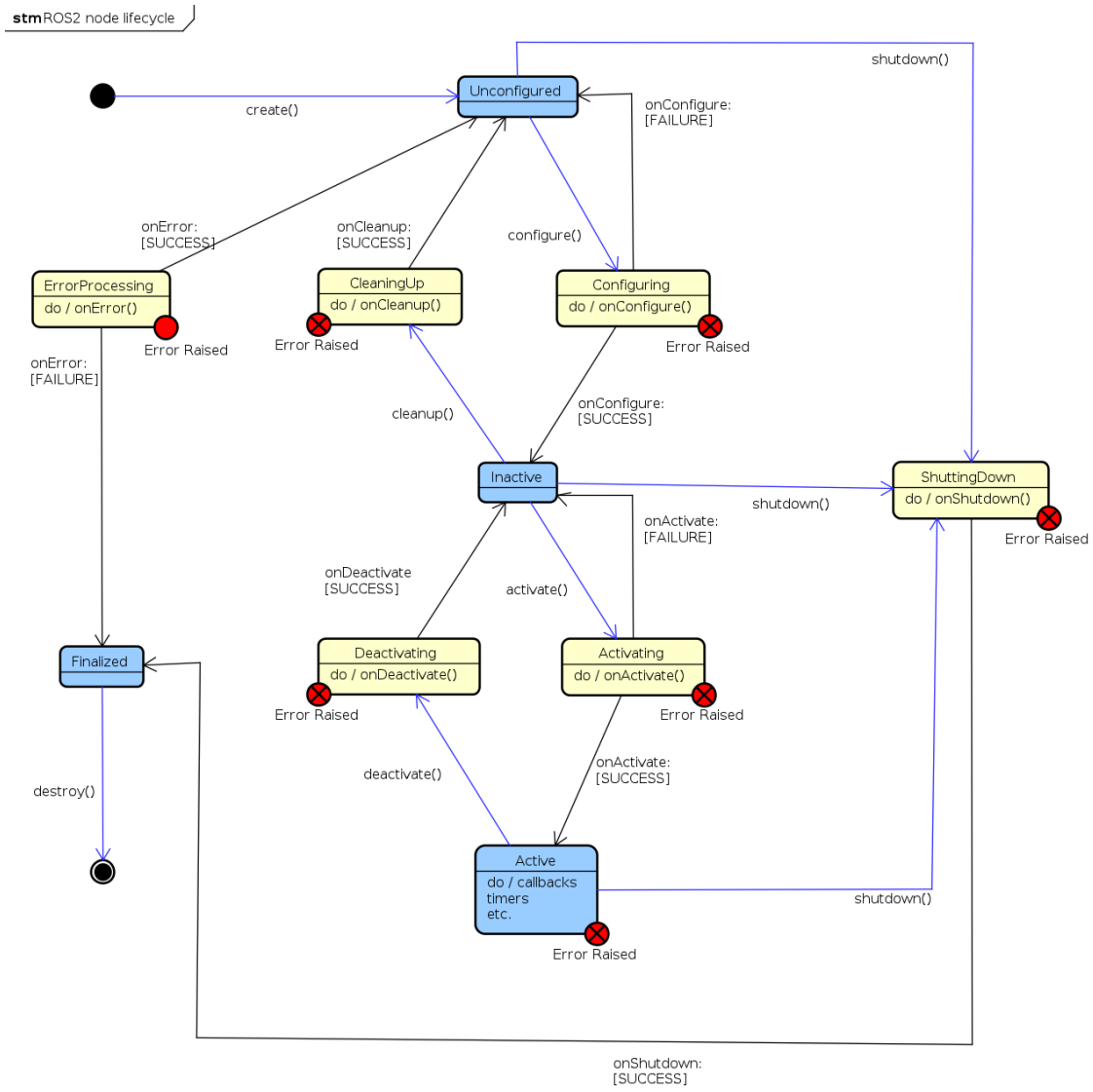
NODES

- Implemented by deriving from the Node base class
- Single process can now easily contain multiple nodes (this required rewriting parts of your code as a Nodelet in ROS1)
- Node activities orchestrated through executors (can be single-threaded or multi-threaded)
- New API encourages an asynchronous, event-driven design that uses callbacks to handle timers, subscribers, and service callbacks
- Asynchronous operation, elimination of blocking operations important for overall responsiveness and real-time safe designs

NODES - LIFECYCLE MANAGEMENT

- New capability not supported by ROS1
- More control over node initialization/shutdown sequencing
- Implemented by inheriting from LifecycleNode base class
- A LifecycleNode has a well-defined set of states and transitions. The four primary states are:
  - **Unconfigured**: Initial state
  - **Inactive**: Configured but not currently performing any processing
  - **Active**: Main state, performing processing
  - **Finalized**: Shut down but available for debugging and introspection
- When a transition is requested, callbacks are made to give the node a chance to take appropriate actions
- The lifecycle model is fully integrated with the command line tools and the launch system

NODES - LIFECYCLE STATE DIAGRAM



COMMUNICATION

- Supports topics, services, and actions as in ROS1 but with a different API
- Actions are now part of the core API instead of an add-on
- Uses the same message definition format as ROS1 with some very minor differences related to types that define time and durations

MIDDLEWARE

- Based on Data Distribution Service (DDS)
- DDS-Security Profile adds capabilities not in ROS1:
  - Privacy: Strong symmetric key encryption (e.g. AES)
  - Authentication: Private keys and X.509 certificates
  - Authorization: Read/write permissions per-node, per-topic
  - Authorization config files cryptographically protected
- Supports ROS1 Quality of Service (QoS) configurations:
  - Best effort or reliable transport
  - Queue depths
  - Latching
  - Liveliness (required “Bond” add-on package in ROS1)
- Adds new QoS configurations:
  - Deadlines: sends notification if update not sent/received on time
  - Lifespan: prevents delivery of stale data

MIDDLEWARE (CONTINUED)

- DDS capabilities not yet available in ROS2:
  - Message Prioritization: service higher priority messages before lower priority messages, drop low priority before high
  - Time Sensitivity: use the age of messages to determine the processing order to guarantee deadlines are achieved
- Performance analysis was done in 2016 and showed that DDS latency was about the same as in ROS1.
- A security analysis was done in 2018 that concluded that the default ROS2 middleware “did not conform to the security specification by OMG [Object Management Group]”.
- Active development on the middleware has continued since that time, so an updated analysis is required to re-assess the situation

MIDDLEWARE

SUPPORTED DDS IMPLEMENTATIONS

Product	License	Notes
eProsima Fast RTPS	Apache2	ROS2 default middleware
ADLINK Opensplice	Apache2	Support will be discontinued in favor of Cyclone
RTI Connext	Commercial	Must be licensed
Eclipse Cyclone	EPL 2.0	Currently missing security and QoS features

COMPONENTS

- Similar to ROS1 Nodelets
- Nodelets required partial re-write of your existing code
- A ROS2 Node can be turned into a Component with only a few extra lines of code in your package
- Components can be dynamically loaded into a manager process (similar to nodelet manager)

LAUNCH

- New programmatic launch system based on Python provides increased flexibility over ROS1 XML launch
- Supports event handling and node lifecycles, which allows:
  - Initialization sequencing decisions like “don’t launch node B until node A is active (i.e. fully configured and running)”
  - Increased flexibility for handling faults (ROS1 only allows “respawn process” or “shutdown entire launch system”)
- Does not yet support remote launch
- Static XML launch files also supported as of Eloquent
  - Similar to ROS1 XML but with some changes
  - Migration is needed
  - Respawn attribute not supported, enhancement request exists

PARAMETERS

- In ROS1 any node can read/write any parameter (global)
- In ROS2 each parameter is owned by exactly one node (local)
- Parameter owner can enforce constraints such as a valid value range or read-only access
- Dynamic reconfigure through simple callback
- Lack of global parameter server could be a problem for migration in cases where multiple nodes shared a single parameter
- While it is possible to implement a single global parameter server node that allows general, unrestricted parameter access, it is not recommended since this violates system security principles
- Shared parameters should be grouped into cohesive sets and managed by nodes created specifically for that purpose

LOGGING

- Similar to ROS1 but has generalized the concept of a logger object
- Requires an additional logger object argument to logger macro
- Added features:
  - Log based on the Boolean result of a general function
  - Skip first log message
- Missing features:
  - Throttle with an initial delay
  - Filter based on formatted log message contents
  - Generalized mechanism for changing individual log levels at runtime
- Would benefit greatly from a tool like rqt_logger_level

OTHER/MISCELLANEOUS

- URDF still used for robot models
- Transforms still calculated using TF2
- Plugin support is available and already used heavily by navigation stack, Rviz, and MoveIt!
- The ROS1 bond package (for monitoring liveliness of the connection between two nodes) has been ported

ROS2 FEATURE SURVEY

- Build System
- Core
  - Nodes
  - Communication
  - Middleware
  - Components
  - Launch
  - Parameters
  - Plugins
  - Logging
  - Transforms
  - Bonding
- **Algorithms**
  - Diagnostics
  - Controllers
  - State Estimation
  - SLAM
  - Navigation
  - Perception
  - Manipulation
- Drivers
  - CAN
  - GPS
  - IMU
  - LiDAR
  - Cameras
- Tools
  - Command Line
  - RQt
  - Rviz
  - Gazebo

DIAGNOSTICS

Package	Description	Eloquent	Repo
diagnostics_updater	Tools for easily updating the diagnostics topic	Debian	1

[1] https://github.com/ros/diagnostics/tree/eloquent/diagnostic_updater

- Currently not supported in ROS2:
  - diagnostic_aggregator: a node that uses analyzer plugins to process and categorize diagnostics data
  - rqt_robot_monitor: a plug-in that allows viewing of diagnostic status through the RQt GUI

CONTROLLERS

Package	Description	Eloquent	Repo
ros_control	Controller manager, robot hardware interface	Source	1
ros_controller	Useful controllers such as joint trajectory, etc.	Source	2

[1] https://github.com/ros-controls/ros2 control/tree/dashing update
[2] https://github.com/ros-controls/ros2 controllers/tree/dashing update

- ROS1 controller packages have NOT yet been ported to ROS2
- As of January 2020, there was interest by Amazon and PAL Robotics to form a working group with the goal of doing a conversion
- Controllers for differential and Ackermann vehicle control would be useful additions to ROS2 for ROS-M

STATE ESTIMATION & SLAM

Package	Description	Eloquent	Repo
robot_localization	Non-linear state estimation through sensor fusion	Source	1
slam_toolbox	Lifelong mapping and localization (2D)	Source	2
cartographer	Google Cartographer SLAM (3D)	Source	3

[1] https://github.com/cra-ros-pkg/robot localization/tree/dashing-devel
[2] https://github.com/SteveMacenski/slam toolbox/tree/eloquent-devel
[3] https://github.com/ros2/cartographer

- SLAM Toolbox selected by ROS2 TSC as default SLAM
- Cartographer more appropriate for ROS-M because it supports 3D SLAM
- A port of LaMa SLAM has also been discussed

NAVIGATION

Package	Description	Eloquent	Repo
navigation2	Global and local planner for navigation	Source	1
teb_local_planner	Timed Elastic Band Local Planner	Source	2

[1] https://github.com/ros-planning/navigation2/tree/eloquent-devel
[2] https://github.com/rst-tu-dortmund/teb local planner/tree/eloquent-devel

- navigation2 is the ROS2 replacement for ROS1 move_base:
  - Task coordination using behavior trees
  - Plugin architecture for planners
  - A* default global planner
  - DWA default local planner
  - TEB local planner port in progress

PERCEPTION

Package	Description	Eloquent	Repo
image_pipeline	Camera calibration, distortion removal, stereo, depth	Source	1
perception_pcl	Data structures and algorithms for working with point clouds	Debian	2
vision_opencv	Data structures and algorithms f or computer vision	Debian	3
tensorflow	ROS nodes for using tensorflow machine learning	Python in repo	4
object_analytics	Object tracking and 3D localization	Source	5

[1] https://github.com/ros-perception/image pipeline/tree/ros2
[2] https://github.com/ros-perception/perception pcl/tree/dashing-devel
[3] https://github.com/ros-perception/vision opencv/tree/ros2
[4] https://github.com/alsora/ros2-tensorflow
[5] https://github.com/intel/ros2 object analytics

MANIPULATION

Package	Description	Eloquent	Repo
moveit2	Joint motion planning	Source	1
grasp	Grasp detection and planning	Source	2

[1] https://github.com/ros-planning/moveit2
[2] https://github.com/intel/ros2_grasp_library

- MoveIt! 2 is currently in Beta

ROS2 FEATURE SURVEY

- Build System
- Core
  - Nodes
  - Communication
  - Middleware
  - Components
  - Launch
  - Parameters
  - Plugins
  - Logging
  - Transforms
  - Bonding
- Algorithms
  - Diagnostics
  - Controllers
  - State Estimation
  - SLAM
  - Navigation
  - Perception
  - Manipulation
- **Drivers**
  - CAN
  - GPS
  - IMU
  - LiDAR
  - Cameras
- Tools
  - Command Line
  - RQt
  - Rviz
  - Gazebo

CAN

Package	Description	Eloquent	Repo
ros_canopen	Speak to devices using the CANopen protocol	Source	1

[1] https://github.com/ros-industrial/ros canopen/tree/dashing

- DBW support for Lincoln MKZ and (Fiat Chrysler) FCA platforms is available in ROS1 but has not been ported to ROS2

GPS

Package	Description	Eloquent	Repo
novatel_gps_driver	Support for NovAtel GPS / GNSS receivers	Source	1
gps_tools	Convert raw GPS data into ROS odometry	Source	2

[1] https://github.com/swri-robotics/novatel_gps_driver/tree/dashing-devel
[2] https://github.com/swri-robotics/gps_umd/tree/dashing-devel/gps_tools

- Drivers for GPS devices from Microstrain (GX4/GX5) and ublox have not yet been ported to ROS2.

IMU

Device	Eloquent	Repo
Microstrain 3DM-GX2	Source	1
PhidgetSpatial 3/3/3	Debian	2

[1] https://github.com/ros-drivers/microstrain 3dmgx2 imu/tree/dashing-devel
[2] https://github.com/ros-drivers/phidgets drivers/tree/dashing

- ROS1 includes drivers for the following devices that have not been ported to ROS2:
  - MicroStrain GX4/GX5
  - Bosch BNO055
  - DSP-3000
- ROS1 also includes IMU support packages for filtering that are not yet ported to ROS2
- Since most devices support some level of on-board sensor fusion, the filtering packages may not be necessary for many applications

LIDAR

Device	Eloquent	Repo
Velodyne	Source	1
Hokuyo	Source	2
SICK	Source	3
Ouster	Debian	4

[1] https://github.com/ros-drivers/velodyne/tree/dashing-devel
[2] https://https://github.com/bponsler/urg_node/tree/ros2-devel
[3] https://github.com/SICKAG/sick_scan2
[4] https://github.com/SteveMacenski/ros2_ouster_drivers

CAMERAS

Package	Description	Eloquent	Repo
image_common	Calibration management and image transport	Debian	1

Device	Eloquent	Repo
Intel RealSense	Source	2
StereoLabs Zed	Source	3

[1] https://github.com/ros-perception/image common/tree/dashing
[2] https://github.com/intel/ros2_intel_realsense
[3] https://github.com/stereolabs/zed-ros2-wrapper

- Drivers for popular GigE Vision monocular cameras (e.g. AVT Prosilica, FLIR Blackfly, Basler Ace) have not yet been ported to ROS2.

ROS2 FEATURE SURVEY

- Build System
- Core
  - Nodes
  - Communication
  - Middleware
  - Components
  - Launch
  - Parameters
  - Plugins
  - Logging
  - Transforms
  - Bonding
- Algorithms
  - Diagnostics
  - Controllers
  - State Estimation
  - SLAM
  - Navigation
  - Perception
  - Manipulation
- Drivers
  - CAN
  - GPS
  - IMU
  - LiDAR
  - Cameras
- **Tools**
  - Command Line
  - RQt
  - Rviz
  - Gazebo

COMMAND LINE

- ROS1 commands such as roslaunch, rosrun, etc. have been consolidated under a single “ros2” command with these subcommand functions:
  - Run Launch Files
  - Run Executables
  - List and Describe Nodes
  - List, Describe, Set, and Get Parameters
  - Publish, Subscribe, and Describe Topics
  - Execute and Describe Services
  - Play and Record Bags

- ROS2 CLI “Cheat Sheet”:

  https://github.com/ubuntu-robotics/ros2_cheats_sheet

RQT

- RQt is a plugin-based GUI framework for ROS tools
- About 2/3 of the ROS1 RQt Plugins have been ported to ROS2
- The following RQt Plugins are not yet supported in ROS2:

Plugin	Description
Package Graph	Displays graphs of ROS package dependencies
Bag	Record, play, and inspect bag files
Web	Provides a panel to access web URLs
Diagnostics Viewer	Display raw diagnostics
Logger Levels	Dynamically set verbosity on a per logger basis
Runtime Monitor	Display current, categorized diagnostic status
Navigation Viewer	Displays maps and plans
Pose View	Visualize the orientation described by a pose topic
TF Tree	Visualize the transform tree of a robot

RVIZ

- RViz is a GUI tool for visualizing data published by ROS nodes
- Almost all default display plugins have been ported to ROS2
- Only DepthCloud and Effort are missing, do not seem to be widely used.
- Note that point clouds are visualized in RViz using the PointCloud plugin

GAZEBO

- Gazebo is the standard physics simulation environment and visualizer for ROS.
- Work was completed over the summer of 2019 to port the bulk of Gazebo to ROS2
- Ignition Gazebo is an updated and restructured version of the classic Gazebo software:
  - Features improved performance and rendering capabilities
  - Uses its own middleware layer, called Ignition Transport, but a ROS2-to-Ignition bridge is available

Q&A