I-VICS R4 Test Document

Introduction

<Details about Additional tests required for this Blue Print in addition to the Akraino Validation Feature Project>

This article motivates developers to adopt integration testing by explaining how to write, run, and evaluate the results of integration tests.

Akarino Test Group Information

<The Testing Ecosystem>

1. colcon is used to build and run test
2. pytest is used to eventually execute the test, generate jUnit format test result and evaluate the result
3. unit testing describes testing big picture

Overall Test Architecture

Integration tests determine if independently developed software modules work correctly when the modules are connected to each other. In ROS 2, the software modules are called nodes.

Integration tests help to find the following types of errors:

- Incompatible interaction between nodes, such as non-matching topics, different message types, or incompatible QoS settings
- Reveal edge cases that were not touched with unit tests, such as a critical timing issue, network communication delay, disk I/O failure, and many other problems that can occur in production environments
- Using tools like `stress` and `udpreplay`, performance of nodes is tested with real data or while the system is under high CPU/memory load, where situations such as `malloc` failures can be detected

Test Bed

Test Framework

This section provides examples for how to use the `integration_tests` framework. The architecture of `integration_tests` framework is shown in the diagram below.

blocked URL
integration_test architecture

Traffic Generator

Test API description

<Akraino common tests>

The Test inputs

Test Procedure

Expected output

Test Results

<Blueprint extension tests>

The Test inputs

Test Procedure

Expected output

Test Results

<Feature Project Tests>

Integration test with a single executable

The simplest scenario is a single node. Create a package named `my_cool_pkg` in the `~/workspace` directory; it's recommended to use the package creation tool.

`my_cool_pkg` has an executable that prints `Hello World` to `stdout`. Follow the steps below to add an integration test:

1. Create a file `~/workspace/src/my_cool_pkg/test/expected_outputs/my_cool_pkg_exe.regex` with the content `Hello\sWorld`
   - The string in the file is the regular expression to test against the `stdout` of the executable
2. Under the `BUILD_TESTING` code block, add a call to `integration_tests` to add the test
   set(MY_COOL_PKG_EXE "my_cool_pkg_exe")
   add_executable(${MY_COOL_PKG_EXE} ${MY_COOL_PKG_EXE_SRC} ${MY_COOL_PKG_EXE_HEADERS})
   ...
   find_package(integration_tests REQUIRED)
   integration_tests(
   EXPECTED_OUTPUT_DIR "${CMAKE_SOURCE_DIR}/test/expected_outputs/"
   COMMANDS
   "${MY_COOL_PKG_EXE}"
)
   ...
3. Build `~/workspace/`, or just the `my_cool_pkg` package, using `colcon`:
   $ ade enter
   ade$ cd ~/workspace/
   ade$ colcon build --merge-install --packages-select my_cool_pkg
4. Run the integration test
   $ ade enter
   ade$ cd ~/workspace/
   ade$ colcon test --merge-install --packages-select my_cool_pkg --ctest-args -R integration
   ...
   Starting >>> my_cool_pkg
   Finished <<< my_cool_pkg [4.79s]

   Summary: 1 package finished [6.30s]

NoteUse --ctest-args -R integration to run integration tests only.

`colcon test` parses the package tree, looks for the correct build directory, and runs the test script. `colcon test` generates a jUnit format test result for the integration test.

By default `colcon test` gives a brief test report. More detailed information exists in `~/workspace/log/latest_test/my_cool_pkg`, which is the directory that holds the directories `ctest`, `stdout`, and `stderr` output. Note that these directory only contains output of `ctest`, not the output of tested executables.

1. `command.log` contains all the test commands, including their working directory, executables, arguments
2. `stderr.log` contains the standard error of `ctest`
3. `stdout.log` contains the standard output of `ctest`

The `stdout` of the tested executable is stored in the file `~/workspace/build/my_cool_pkg/test_results/my_cool_pkg/my_cool_pkg_exe_integration_test.xunit.xml` using jUnit format:

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite errors="0" failures="0" name="pytest" skips="0" tests="1" time="3.121">
<testcase classname="my_cool_pkg.build.my_cool_pkg.my_cool_pkg_exe_integration_test_Debug"
file="~/workspace/build/my_cool_pkg/my_cool_pkg_exe_integration_test_Debug.py"
line="36" name="test_executable" time="3.051783800125122">
<system-out>(test_executable_0) pid 21242:
['~/workspace/build/my_cool_pkg/my_cool_pkg_exe'] (all >; console, InMemoryHandler: test_executable_0)
[test_executable_0] Hello World
[test_executable_0] signal_handler(2)
(test_executable_0) rc 27
() tear down
</system-out>
</testcase>
</testsuite>
```

`test_executable_i` corresponds to the `(i+1)th` executable. In this case, only one executable is tested so `i` starts from `0`. Note that `test_executable_0` prints `Hello World` to `stdout`, which is captured by the `launcher`. The output matches the regex `Hello\sWorld` specified in the expected output file. The `launcher` then broadcasts a `SIGINT` to all the test executables and marks the test as successful. Otherwise, the integration test fails.

NoteSIGINT is broadcast only if the output of the last executable matches its regex.

For detailed information about how `integration_tests` operates, see the Q&A section below.

Integration test with multiple executables

In the `my_cool_pkg` example, only one executable is added to the integration test. Typically, the goal is to test the interaction between several executables. Suppose `my_cool_pkg` has two executables, a `talker` and a `listener` which communicate with each other with a ROS2 topic.

The `launcher` starts the `talker` and `listener` at the same time. The `talker` starts incrementing the index and sending it to the `listener`. The `listener` receives the index and prints it to `stdout`. The passing criteria for the test is is if `listener` receives the indices `10`, `15`, and `20`.

Here are the steps to add multiple-executable integration tests:

1. Create two files
    a. `~/workspace/src/my_cool_pkg/test/expected_outputs/talker_exe.regex` with content `.*`
    b. `~/workspace/src/my_cool_pkg/test/expected_outputs/listener_exe.regex` with content
    10
    15
    20
2. Under the `BUILD_TESTING` code block, call `integration_tests` to add the test
    ...
    find_package(integration_tests REQUIRED)
    integration_tests(
    EXPECTED_OUTPUT_DIR "${CMAKE_SOURCE_DIR}/test/expected_outputs/"
    COMMANDS
    "talker_exe --topic TOPIC:::listener_exe --topic TOPIC"
)
    ...

    ```
    1. The character set `:::` is used as delimiter of different executables
    2. `integration_tests` parses the executables, arguments, and composes a valid test python
    script
    3. More information about the python script can be found in the
    [Q&A](@ref how-to-write-integration-tests-how-does-integration-tests-work) section
    ```

3. Build `~/workspace/`, or just the `my_cool_pkg` package, using `colcon`:
    $ ade enter
    ade$ cd ~/workspace/
    ade$ colcon build --merge-install --packages-select my_cool_pkg
4. Run the integration test
    $ ade enter
    ade$ cd ~/workspace/
    ade$ colcon test --merge-install --packages-select my_cool_pkg --ctest-args -R integration
    Starting >>> my_cool_pkg
    Finished <<< my_cool_pkg [20.8s]

    Summary: 1 package finished [22.3s]

When the environment is properly configured, after 20 seconds, the integration test shall pass. Similar to the single node example, the `launcher` starts the `talker` and `listener` at the same time. The `launcher` periodically checks the `stdout` of each executable.

The regex of `talker` is `.*`, which always matches when the first output of `talker` is captured by launcher. The regex of `listener` is `10`, `15`, and `20`. After all entries in this regex are matched, a SIGINT is sent to all commands and the test is marked as successful.

The locations of output files are the same with single executable example. Output of `ctest` is is `~/workspace/log/latest_test/my_cool_pkg/`. Output of tested executables is stored in `~/workspace/build/my_cool_pkg/test_results/my_cool_pkg/` in jUnit format.

NoteBy the time SIGINT is sent, all the regex have to be successfully matched in the output. Otherwise the test is marked as failed. For example, if the regex for talker is 30, the test will fail.

Use executables from another package

Sometimes an integration test needs to use executables from another package. Suppose `my_cool_pkg` needs to test with the `talker` and `listener` defi ned in `demo_nodes_cpp`. These two executables must be exported by `demo_nodes_cpp` and then imported by `my_cool_pkg`.

When declaring the test, a namespace must be added before `talker` and `listener` to indicate that executables are from another package.

Use the following steps to add an integration test:

1. Add `<buildtool_depend>ament_cmake</buildtool_depend>` to `~/workspace/src/demo_nodes_cpp/package.xml`
2. In `~/workspace/src/demo_nodes_cpp/CMakeLists.txt`, export the executable target before calling `ament_package()`
   install(TARGETS talker EXPORT talker
   DESTINATION lib/${PROJECT_NAME})
   install(TARGETS listener EXPORT listener
   DESTINATION lib/${PROJECT_NAME})
   find_package(ament_cmake REQUIRED)
   ament_export_interfaces(talker listener)
3. Create two regex files
   a. `~/workspace/src/my_cool_pkg/test/expected_outputs/demo_nodes_cpp__talker.regex` with content `.*`
   b. `~/workspace/src/my_cool_pkg/test/expected_outputs/demo_nodes_cpp__listener.regex` with content `20`
4. In `~/workspace/src/my_cool_pkg/package.xml`, add the dependency to `demo_nodes_cpp`
   <test_depend>demo_nodes_cpp</test_depend>
5. Under the `BUILD_TESTING` code block in `~/workspace/src/my_cool_pkg/CMakeLists.txt`, call `integration_tests` to add the test
   ...
   find_package(integration_tests REQUIRED)
   find_package(demo_nodes_cpp REQUIRED) # this line imports targets(talker) defined in namespace demo_nodes_cpp
   integration_tests(
   EXPECTED_OUTPUT_DIR "${CMAKE_SOURCE_DIR}/test/expected_outputs/"
   COMMANDS
   "demo_nodes_cpp::talker:::demo_nodes_cpp::listener" # format of external executable is namespace::executable [--arguments]
)
   ...
6. Build `~/workspace/`, or just the `my_cool_pkg` package, using `colcon`:
   $ ade enter
   ade$ cd ~/workspace/
   ade$ colcon build --merge-install --packages-select my_cool_pkg
7. Run the integration test
   $ ade enter
   ade$ cd ~/workspace/
   ade$ colcon test --merge-install --packages-select my_cool_pkg --ctest-args -R integration

When `ament_export_interfaces(talker listener)` is called in `demo_nodes_cpp`, ament generates a `demo_nodes_cppConfig.cmake` file which is used by `find_package`. The namespace in this file is `demo_nodes_cpp`. Therefore, to use executable in `demo_nodes_cpp`, a namespace and `::` has to be added.

The format of an external executable is `namespace::executable --arguments`. The `integration_tests` function sets the regex file name as `namespace__executable.regex`. One exception is that no namespace is needed for executable defined in the package that adds this integration test.

Add multiple integration tests in one package

If `my_cool_pkg` has multiple integration tests added with the same executable but different parameters, `SUFFIX` has to be used when calling `integration_tests`.

Suppose `my_cool_pkg` has an executable `say_hello` which prints `Hello {argv[1]}` to the screen. Here are the steps to add multiple integration tests:

1. Create two regex files
   a. `~/workspace/src/my_cool_pkg/test/expected_outputs/say_hello_Alice.regex` with content `Hello\sAlice`
   b. `~/workspace/src/my_cool_pkg/test/expected_outputs/say_hello_Bob.regex` with content `Hello\sBob`
2. Call `integration_tests` to add integration test
   integration_tests(
   EXPECTED_OUTPUT_DIR "${CMAKE_SOURCE_DIR}/test/expected_outputs/"
   COMMANDS "say_hello Alice"
   SUFFIX "_Alice"
)
   integration_tests(
   EXPECTED_OUTPUT_DIR "${CMAKE_SOURCE_DIR}/test/expected_outputs/"

COMMANDS "say_hello Bob"
SUFFIX "_Bob"
)

3. Build `~/workspace/`, or just the `my_cool_pkg` package, using `colcon`:
   $ ade enter
   ade$ cd ~/workspace/
   ade$ colcon build --merge-install --packages-select my_cool_pkg
4. Run the integration test
   $ ade enter
   ade$ cd ~/workspace/
   ade$ colcon test --merge-install --packages-select my_cool_pkg --ctest-args -R integration

By specifying `SUFFIX`, `integration_tests` adds the correct suffix to the regex file path.

Test Dashboards

Single pane view of how the test score looks like for the Blue print.

Total Tests	Test Executed	Pass	Fail	In Progress

Additional Testing

Bottlenecks/Errata