iRODS Development and Testing Environments (v8)

Alan King
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
alanking@renci.org

Terrell Russell
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
unc@terrellrussell.com

ABSTRACT

iRODS Build and Test continues to evolve. Testing a distributed system is hard and this paper describes the eighth generation of our efforts to do it well. This paper includes containers, Python, and no Groovy.

Keywords

iRODS, testing, development, framework, python

INTRODUCTION

The iRODS Build and Test infrastructure has now been around for more than a decade and continues to evolve as additional features and coverage are required. The iRODS Team has worked to easily and flexibly deploy different iRODS topologies and efficiently exercise the numerous scenarios that a robust iRODS installation can service.

In addition to writing and maintaining a growing lists of tests, seeing those tests pass builds confidence in the changes developers make to software, asserting the correctness of the changes and of the entire system.

Providing an easy and consistent framework in which to run these tests, and see their results, builds confidence for the entire community.

This eighth version (v8) is the best we've done so far.

HISTORY

The following listing is an overview of the history of the various Build-and-Test Systems for iRODS. Each shows a flow of technologies used to provide confidence in iRODS at the time. Over the years, this series of flows also became the gatekeeper for whether iRODS was ready for a new release.

v1 - July 2011: Python → Node.js → RabbitMQ → Celery → Eucalyptus [1]

v2 - October 2012: Python → Node.js → ssh → OpenStack

v3 - January 2013: Hudson → Python → OpenStack

v4 - October 2013: Hudson → Python → vSphere long-running VMs [2]

v5 - Spring 2015: Jenkins → Python → Ansible → zone_bundles → vSphere dynamic VMs [3]

v6 - Spring 2017: Jenkins → Python → vSphere dynamic VMs → build/test hooks [4]

iRODS UGM 2022 July 5-8, 2022, Leuven, Belgium

v7 - Summer 2019: Docker → Jenkins → Python → Docker → build/test hooks [5]

v8 - Summer 2022: Python → Docker → build/test hooks

LIMITATIONS OF THE PAST

The most recent change to the system is the removal of Jenkins. Jenkins [6], the successor to Hudson, is a java-based automation and continuous integration server that was very helpful in being the place where 'jobs' were saved and managed and run from 2013 to 2022. However, there were a few limitations of that approach that we hoped to overcome with v8.

First, since the move to a Docker-based, every-developer-runs-their-own-system approach with v7, the requirement that every developer must now also be an administrator of their own Jenkins became a bit heavyweight and onerous.

Second, the existing structure of Jenkins is pretty inflexible with its notion of jobs and servers and a history of each job over time. We found that we wanted more granular insight across types of inputs to those jobs, rather than just by job name itself.

Third, the combinatoric explosion of variables that we would like to test had become too difficult to maintain in a simple list of Jenkins jobs. If we realized we wanted to test an additional variable, our list of manually defined and curated jobs (in Groovy!) could multiply by the number of enumerated values of the new variable. This was unsustainable and we found ourselves considering writing Python wrappers to generate these Groovy jobs.

Additionally, the test results were captured in the Jenkins namespace and were hard to extract in a flexible manner. Any packages created needed to be extracted and kept in a parallel namespace taking up room in our mental model and additional disk space on individual developers' machines.

Lastly, also related to disk space, since every Jenkins job was building things from scratch, we were dealing with a Docker image explosion - there was one tag per test run in the system - leading to thousands and thousands of nearly identical images. There were a number of disk full events on the development team which were always surprisingly tricky and annoying to recover from.

Stepping back, we realized our needs both as developers and project maintainers were not being met satisfactorily and we needed something better.

iRODS BUILD-AND-TEST SYSTEM (V8)

After a few whiteboard sessions, we determined that we really wanted to separate the building of packages (solving our Docker image explosion problem) from the running of tests (solving our too-many-Jenkins-jobs problem). We needed a consistent flow from source code to built packages to test results (see Figure 1).

We also were looking to provide a more consistent learning and development environment for interns and new hires. This led us to a very straightforward replacement of the initial generic flow with two stages, development and testing. Each stage is now represented by its own standalone git repository with clean inputs and outputs as seen in Figure 2.

DEVELOPMENT ENVIRONMENT

The new iRODS development environment git repository [7] is designed to provide the machinery to easily build the iRODS server, its various plugins, and the required external dependencies (externals) for all supported operating systems as well as a selection of debugging tools. The abstraction layer to provide this on a single machine is through the use of container technology, currently handled by Docker.

The containers build from code that is local to the host machine and produce local packages on the host machine. All of the build tools and processes are run within the containers and do not otherwise affect the host environment

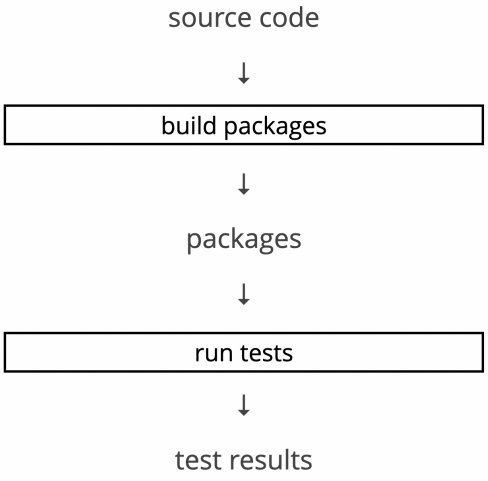

Figure 1. Generic Build-and-Test Workflow

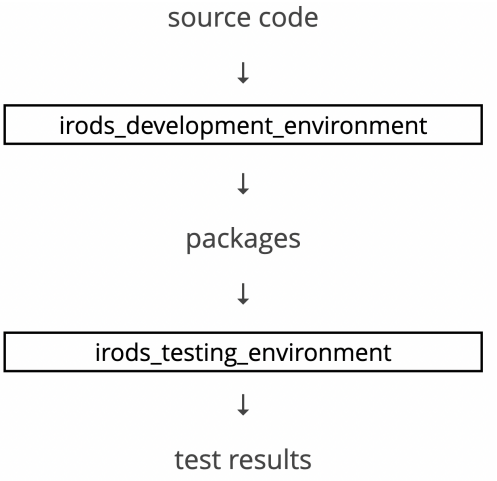

Figure 2. Workflow with Repositories

or filesystem.

The advantages to this approach are numerous when compared to VMs or multiple build machines. First, there is less network traffic to and from the code repositories (in our case, largely GitHub). Second, local source files allow the developer to use their preferred coding environment and tooling which increases speed and confidence. Third, the container-technology's build cache allows for faster iteration between build and test cycles. Lastly, having a consistent process means that the development efforts look very similar to the release process - the same machinery that builds our packages for development and testing is now used to build the packages that are released to everyone else.

An example usage of the iRODS development environment is as follows:

```
$ docker run --rm \
           -v ${irods_sourcedir}:/irods_source:ro \
           -v ${irods_builddir}:/irods_build \
           -v ${icommands_sourcedir}:/icommands_source:ro \
           -v ${icommands_builddir}:/icommands_build \
           -v ${irods_packagedir}:/irods_packages \
           -v ${externals_packagedir}:/irods_externals_packages:ro \
           irods-core-builder:${PLATFORM}-${VERSION}
```

The above example runs an `irods-core-builder` for a particular `${PLATFORM}` and `${VERSION}`. The three read-only (`ro`) volume mounts specify the location of the local source code for the iRODS server (`${irods_sourcedir}`), the iRODS iCommands (`${icommands_sourcedir}`), and the external dependency packages already built and gathered for this platform and version (`${externals_packagedir}`). The other three volume mounts specify the locations of two build directories for the build artifacts (`${irods_builddir}` and `${icommands_builddir}`) and the location where the newly built packages will be deposited (`${irods_packagedir}`).

TESTING ENVIRONMENT

The new iRODS testing environment git repository [8] is designed to provide the machinery to test various iRODS configurations. The python scripts currently leverage the functionality of Docker Compose to easily stand up one or more iRODS zones, configure them, federate them, and then run tests and gather the results. Again, through the use of container technology this happens on a single host machine.

These local scripts execute the issued commands in long-running containers. They install and configure local (newly built) or released packages and generate local test results. The scripts have options to skip the testing which allows a developer to quickly have access to a running zone from their latest built packages for manual inspection and testing.

The first advantage of this approach is precision control for running various tests in parallel since multiple independent containers can run independent zones to avoid any interaction or dependencies. A second related advantage is that this approach provides a convenient way to reproduce reported issues because of the consistent, reproducible configurations. This also provides a consistent process for both bench (manual) and automated testing.

The current list of scripts available in the testing repository:

- stand_it_up.py - stand up a zone with multiple servers
- federate.py - stand up and federate multiple zones
- run_core_tests.py - run iRODS server tests
- run_unit_tests.py - run unit tests for iRODS libraries
- run_topology_tests.py - run tests on a multi-server zone
- run_federation_tests.py - run tests in federated zones
- run_plugin_tests.py - run tests for iRODS plugins

An example usage of running the core tests is as follows:

```
$ python run_core_tests.py \
  --project-directory projects/ubuntu-20.04/ubuntu-20.04-postgres-10.12 \
  --irods-package-directory ~/hdd/builds/irods_packages/4-3-stable/ubuntu-20.04 \
  --concurrent-test-executor-count 4
```

The above example runs the core test suite with configuration details defined in the `--project-directory` located in the relative path of `projects/ubuntu-20.04/ubuntu-20.04-postgres-10.12` with the binary packages found in the `--irods-package-directory` of `~/hdd/builds/irods_packages/4-3-stable/ubuntu-20.04`. The `--concurrent-test-executor-count` of `4` instructs the script to stand up four concurrent identical zones, distribute the tests across those four zones, and run the tests in parallel. As the tests complete, the log files and the test results are copied back to the host machine. Once all tests are complete, the script stops the four zones and removes the running containers.

The results of the fourth zone can be seen here:

```
-----
results for [ubuntu-2004-postgres-1012_irods-catalog-provider_4]
        passed tests:
                [[  30.0808]s]  [test_collection_mtime]
                [[ 837.2263]s]  [test_iadmin]
                [[  59.8457]s]  [test_ichmod]
                [[  13.3225]s]  [test_ifsck]
                [[  58.9188]s]  [test_ils]
                [[   8.9275]s]  [test_imeta_help]
```

```
                [[  34.9601]s]  [test_imv]
<snip>
                [[  28.2190]s]  [test_quotas]
                [[1081.7692]s]  [test_resource_types.Test_Resource_CompoundWithUnivmss]
                [[ 808.2622]s]  [test_resource_types.Test_Resource_Passthru]
                [[2091.9287]s]  [test_resource_types.Test_Resource_Replication]
                [[ 857.6486]s]  [test_resource_types.Test_Resource_Unixfilesystem]
                [[ 917.5663]s]  [test_rulebase]
                [[  78.1721]s]  [test_symlink_operations]
        skipped tests:
        failed tests:
        return code:[0]
        time elapsed: [7.345e+03]seconds ([   2]hours [    2.424]minutes)
-----


All tests passed! :)
time elapsed: [10955.3559]seconds ([   3]hours [ 2.5893]minutes)
==== end of test run results ====
```

The logs for this test run can then be found in the reported location:

```
2022-07-04 20:57:00,726 INFO - collecting logs
[/tmp/ubuntu-2004-postgres-10123x3tjb2r/ubuntu-2004-postgres-1012_be703715-7901-4a34-affa-10ebea651ff4]
```

FUTURE WORK

The next few steps for the iRODS Build and Test infrastructure are incremental. The container-based approach will probably last for a while and progress will come from a few different areas, including automation, client testing, and environmental reproduction and orchestration.

Since moving away from Jenkins, the automation of testing every commit has fallen away. Working to reproduce the visibility of continuous integration is a near-term goal. Some progress has already been made, but it is not clear whether building this from scratch will be worth the effort.

Adding various iRODS clients to the testing infrastructure is an ongoing effort as well. Most clients do not already have their own test suites. Command line tools will be easy enough to write tests for, but GUIs will require additional work.

The original design goal for the iRODS Zone Report was to provide a serialization format for a zone's topology that could be generated from an existing deployment as well as be handed to a tool for automatic deployment for testing and issue reproduction. The format itself has proven useful but needs some modernization work for the 4.3 release series.

SUMMARY

The eighth generation of the iRODS Build and Test infrastructure provides a cleaner slate for the iRODS Consortium to build confidence and visibility around the core iRODS server and its plugins. It has already increased iteration speed for the development team and can guarantee any released binaries come from the same machinery that shows all the tests are passing.

REFERENCES

[1] Russell, T., Coposky, J., Brieger, L., Stealey, M.: Initial Enterprise iRODS Release. 2012 iRODS User Group Meeting (2012). https://irods.org/uploads/2012/03/Russell-RENCI-EiRODS.pdf

[2] Russell T.: iRODS 4.0 - Build and Test. 2014 iRODS User Group Meeting (2014). https://irods.org/uploads/2014/06/Terrell-iRODS-4.0-BuildAndTest.pdf

[3] Russell T., Keller, B.: iRODS Cloud Infrastructure and Testing Framework. 2015 iRODS User Group Meeting (2015). https://irods.org/uploads/2015/06/RussellKeller-TestingFramework.pdf

[4] Russell, T., Gill, J.: iRODS Build and Test (part of the iRODS Technology Update). 2018 iRODS User Group Meeting (2018). https://irods.org/uploads/2018/Russell-iRODS-Technology_Update-slides.pdf

[5] Russell, T., Gill, J.: iRODS Build and Test (part of the iRODS Technology Update). 2019 iRODS User Group Meeting (2019). https://irods.org/uploads/2019/Russell-iRODS-UGM2019_Technology_Update-slides.pdf

[6] Jenkins. https://www.jenkins.io

[7] iRODS Development Environment. https://github.com/irods/irods_development_environment

[8] iRODS Testing Environment. https://github.com/irods/irods_testing_environment