z/TPF Automated Test Framework

Jamie Farmer
z/TPF Development

Problem

- The cost of testing z/TPF application code can be expensive.
  - Test resources are required to manually run and validate test cases
  - Manual testing takes time, increasing time to market for new application function
  - Perception of z/TPF and the mainframe suffers making it harder to get new function on the platform

- The quality of code can suffer
  - In some cases tests are lost over time as skills and priorities change
  - Testing is done with throw away code to get the application tested more quickly

- **There is no z/TPF testing framework in place that allows for repeatable, automated testing**

z/TPF Automated Test Framework

- Allows you to create new (or convert old), self-validating programmatic test cases similar to other testing frameworks, like Google Test.
  - Set of TPF unique APIs to create z/TPF automated tests.

- Provides the ability to organize tests by a namespace
  - For example, airco.res.overbook

- Framework automatically detects new test cases when they are loaded to the z/TPF system
  - Allows for tests to be included with the modules being tested.

- Ability to query / run testcases defined in the z/TPF automated test framework
  - Using the new ZDEVO operator command
  - Using a Java application (ie. jUnit) on a remote platform

Creating Automated Test Cases

qovb.c

```
TPF_TESTSUITE ("airline.overbook", "TOV*, TOB1");

TPF_TESTCASE (overbook_firstClass, "overbooking in first class") {

     // Test case logic

}

TPF_TESTCASE (overbook_economy, "Test overbooking in economy")  {

     // Test case logic

}

.
.   Additional overbook tests
.
```

TPF_TESTSUITE

- defines the namespace of tests in this shared object
- Code coverage mask – reserved for future enhancements

One or more individual test cases that will be contained in the namespace defined

Automatic Detection of Test Cases

qovb.c -> QOVB.so

```
TPF_TESTSUITE ("airline.overbook", "TOV*, TOB1");


TPF_TESTCASE (overbook_firstClass, "overbooking in first
class") {

     // Test case logic

}


TPF_TESTCASE (overbook_economy, "Test overbooking in
economy")  {

     // Test case logic

}
```

OLD/TLD load driver program

z/TPF

Automated Test Table

airline.overbook
  overbook_firstClass
  overbook_economy

Upon loading, the testsuite/testcases in the shared object are runnable using command or from a remote platform.

z/TPF Framework Architecture

z/TPF

Invoked from a remote platform through a REST request.

Test Framework Process

For each test case being run
- Create new process and invoke
- Wait for test case completion

Operator issues ZDEVO command to run one or more tests

New Process

Testcase Process

TPF_TESTCASE(apptest,...) {

- **Perform Test Case Setup**
  - ECB fields
  - Database Setup
  - Data levels
- **Calls to application code**
- **Verify results**
  - Return values
  - ECB fields
  - Read database
  - Etc.
- **Perform any clean-up**
- Return

}

Application must Return

Application Code

Invoking z/TPF Application Code

```
TPF_TESTCASE (overbook_firstClass, "Test overbooking in first class") {
     struct overbook_input overbook_parms;

     …  Setup environment for call

     TPF_TC_INFO ("Calling overbook routine");

     TPF_TC_TIMEOUT(3);

     int rc = process_overbook (&overbook_parms);

     if (rc == RETURN_ERROR)
         TPF_TC_ERROR ("overbook failure-%d", overbook_parms.errCode);
         return;

     …  Validate results
}
```

TPF_TC_INFO to log the flow through the test case

Ability to change timeouts at runtime
Default – 10 seconds

Call to application function

TPF_TC_ERROR to log error messages

- **During the execution of a testcase if a TPF_TC_ERROR at any time during the test case logic results in a test case failure**

Test Case Message Logging

Successful Test Case

```
zdevo run airline.overbook overbook_firstClass
CSMP0097I 13.55.13 CPU-B SS-BSS  SSU-HPN  IS-01
DEVO0004I 13.55.13 PROCESSING FOR THE SELECTED TEST
                   CASES IS STARTED.
-- overbooking in first class --
TEST CASE STARTED
  Calling overbook routine
    … Additional information / error messages
TEST CASE COMPLETED IN 10ms - PASSED - overbook_firstClass
CSMP0097I 13.55.13 CPU-B SS-BSS  SSU-HPN  IS-01
DEVO0005I 13.55.13 RESULTS FOR TEST 1 -
overbook_firstClass
CSMP0097I 13.55.13 CPU-B SS-BSS  SSU-HPN  IS-01
DEVO0018I 13.55.13 1 TEST WERE COMPLETED.
                   1 PASSED, 0 FAILED, 0 SKIPPED+
END OF DISPLAY
```

Failed Test Case

```
zdevo run airline.overbook overbook_firstClass
CSMP0097I 14.30.18 CPU-B SS-BSS  SSU-HPN  IS-01
DEVO0004I 14.30.18 PROCESSING FOR THE SELECTED TEST
                   CASES IS STARTED.
-- overbooking in first class --
TEST CASE STARTED
   Calling overbook routine
     … Additional information / error messages
   ERROR IN QOVB,qovb.c:1813 overbook failure-19
TEST CASE COMPLETED IN 14ms - FAILED - overbook_firstClass

CSMP0097I 14.30.18 CPU-B SS-BSS  SSU-HPN  IS-01
DEVO0007E 14.30.18 AN ERROR OCCURRED WHILE RUNNING TEST CASE
overbook_firstClass.
PGM: QOVB SOURCE: qovb.c LINE: 1813 - overbook failure-19
CSMP0097I 14.30.18 CPU-B SS-BSS  SSU-HPN  IS-01
DEVO0005I 14.30.18 RESULTS FOR TEST 1 - overbook_firstClass
CSMP0097I 14.30.18 CPU-B SS-BSS  SSU-HPN  IS-01
DEVO0020E 14.30.18 1 TEST WERE COMPLETED.
                   0 PASSED, 1 FAILED, 0 SKIPPED
END OF DISPLAY
```

Setting Up Testcase and Validating Results

How do you setup the environment for a test case? How do you validate results?

```
TPF_TESTCASE (overbook_firstClass, "Test overbooking in first class") {
     struct overbook_input overbook_parms;

     …  Setup environment for call

     TPF_TC_INFO ("Calling overbook routine");
     int rc = process_overbook (&overbook_parms);
     if (rc == RETURN_ERROR)
         TPF_TC_ERROR ("overbook failure-%d", overbook_parms.errCode);
         return;


     …  Validate results
        and restore environment

}
```

- Setting up input structures
- Setting up ECB fields
- Setting up data level
- Setting up database

- Validating output
- Validating ECB fields
- Validating data levels
- Validating database
- Validate for memory leaks
- Validate performance (I/O, etc)

May need to restore things to ensure subsequent tests are consistent and repeatable.

Multi-ECB Test Cases

Use TPF_TC_NEW_ECB/TPF_TC_DONE_ECB to include created ECBs in the test case

```
TPF_TESTSUITE ("airline.overbook", "TOV*, TOB1");


TPF_TESTCASE (overbook_multiECB, "overbook multi-ECB") {
     for (int i = 0; i < 10; i++) {
          swisc_create(…)

          TPF_TC_NEW_ECB();     ← Indicates another ECB is
                                  participating in the test
     }
}
```

```
void main() {
   struct overbook_input overbook_parms;

   TPF_TC_INFO ("Calling overbook routine");
   int rc = process_overbook (&overbook_parms);
   if (rc == RETURN_ERROR)
      TPF_TC_ERROR ("overbook failure");
      return;
   TPF_TC_ECB_DONE();
}
```

ECB 1

ECBs created via SWISC

```
void main() {
   struct overbook_input overbook_parms;

   TPF_TC_INFO ("Calling overbook routine");
   int rc = process_overbook (&overbook_parms);
   if (rc == RETURN_ERROR)
      TPF_TC_ERROR ("overbook failure");
      return;
   TPF_TC_ECB_DONE();
}
```

ECB *n*

- Testcase ECB waits for all ECBs to issue TPF_TC_ECB_DONE before completion
  - If any ECB issues TPF_TC_ERROR the testcase fails

Passing Parameters

Can use TPF_TC_SET_PROPERTY/TPF_TC_GET_PROPERTY to pass parameters between ECBs and functions

```
TPF_TESTSUITE ("airline.overbook", "TOV*");


TPF_TESTCASE (overbook_multiECB, "overbooking multiECB")
{
   int option = 1;
   TPF_TC_SET_PARAMETER("option",
       (void*) option, sizeof(option));

   swisc_create(…)

   TPF_TC_NEW_ECB();
}
```

New ECB

```
void main() {
   int length = 0;
   int option =
     *(int *) TPF_TC_GET_PROPERTY("option",length);

   switch (option) {
      …

   }

}
```

- **Parameters saved in shared memory**
- **Parameters can be set or obtained from any ECB participating in test case.**
- **Can be used for…**
  - **Test case specific processing in common routines.**
  - **Serialization of multi-ECB testing**

Invoking Test Cases Using Command

- The ZDEVO command allows you to query and run automated tests that have been loaded to the z/TPF system.
  - Command has two primary parameters – namespace(s) and test case(s)
    - ZDEVO RUN airline.overbook overbook_firstClass

- Support for wildcards in namespace and test cases providing a better user experience
  - ZDEVO RUN airline* overbook* ←All namespaces starting with airline and all testcases starting with overbook
  - ZDEVO RUN airline.overbook over*economy ←All tests starting with “over” and ending in “economy”

- Testcase output is displayed on console and optionally written to file
  - All TPF_TC_INFO and TPF_TC_ERROR messages
    - “Quiet” option to only display testcase output on error

ZDEVO Examples

Querying Tests

```
zdevo info airline.overbook *
CSMP0097I 08.34.24 CPU-B SS-BSS  SSU-HPN  IS-01
DEVO0010I 08.34.24 TEST CASE INFORMATION DISPLAY
PGM   NAME                 DESCRIPTION
----- ------------------- ----------------
**********************************************
QOVB  airline.overbook
**********************************************
QOVB overbook_firstClass  overbooking in first class
QOVB overbook_economy     overbooking in economy
**********************************************
2 TEST CASES TOTAL
```

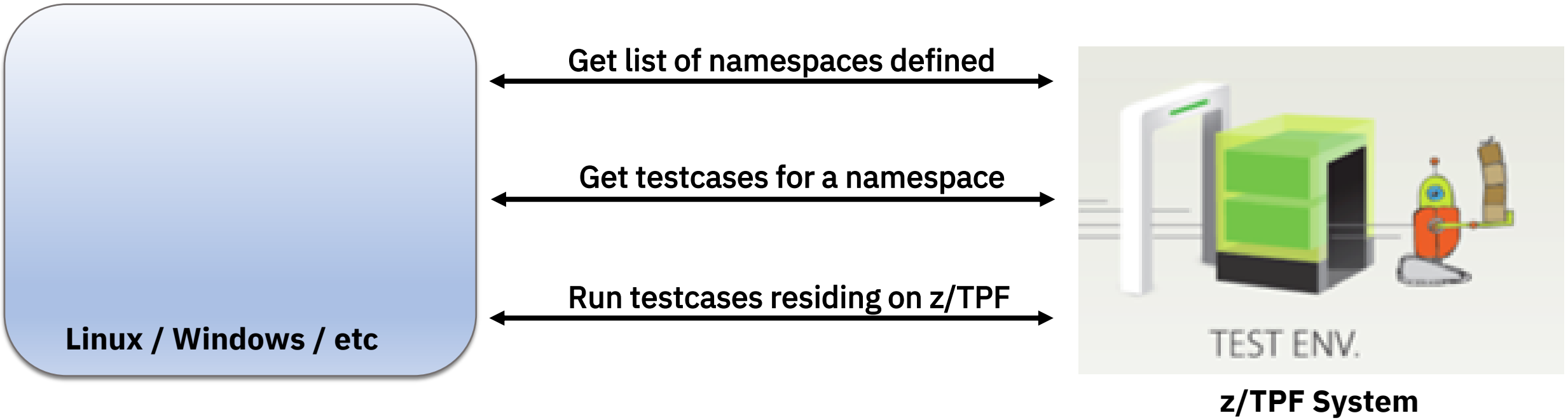
Running Tests

```
zdevo run airline.overbook overbook_economy quiet
CSMP0097I 08.32.22 CPU-B SS-BSS  SSU-HPN  IS-01
DEVO0004I 08.32.22 PROCESSING FOR THE SELECTED
TEST CASES IS STARTED.
CSMP0097I 08.32.41 CPU-B SS-BSS  SSU-HPN  IS-01
DEVO0018I 08.32.41 1 TESTS WERE COMPLETED.
                   1 PASSED, 0 FAILED, 0 SKIPPED+
```

```
zdevo run airline.overbook * quiet
CSMP0097I 08.32.22 CPU-B SS-BSS  SSU-HPN  IS-01
DEVO0004I 08.32.22 PROCESSING FOR THE SELECTED
TEST CASES IS STARTED.
CSMP0097I 08.32.41 CPU-B SS-BSS  SSU-HPN  IS-01
DEVO0018I 08.32.41 2 TESTS WERE COMPLETED.
                   2 PASSED, 0 FAILED, 0 SKIPPED+
```

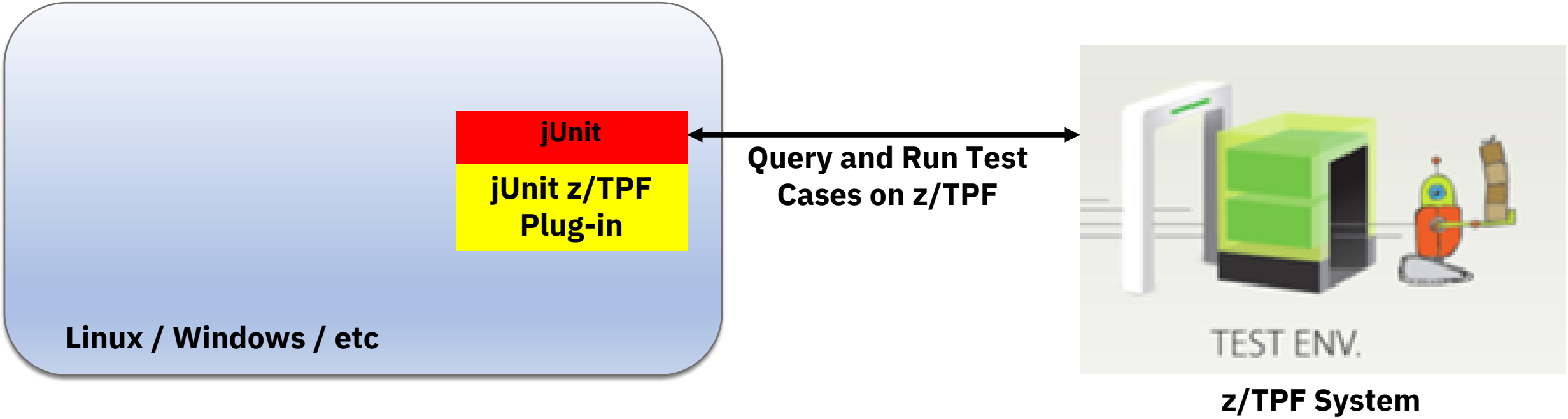
Invoking Test Cases Remotely

REST services have been created to query and run tests from a remote platform



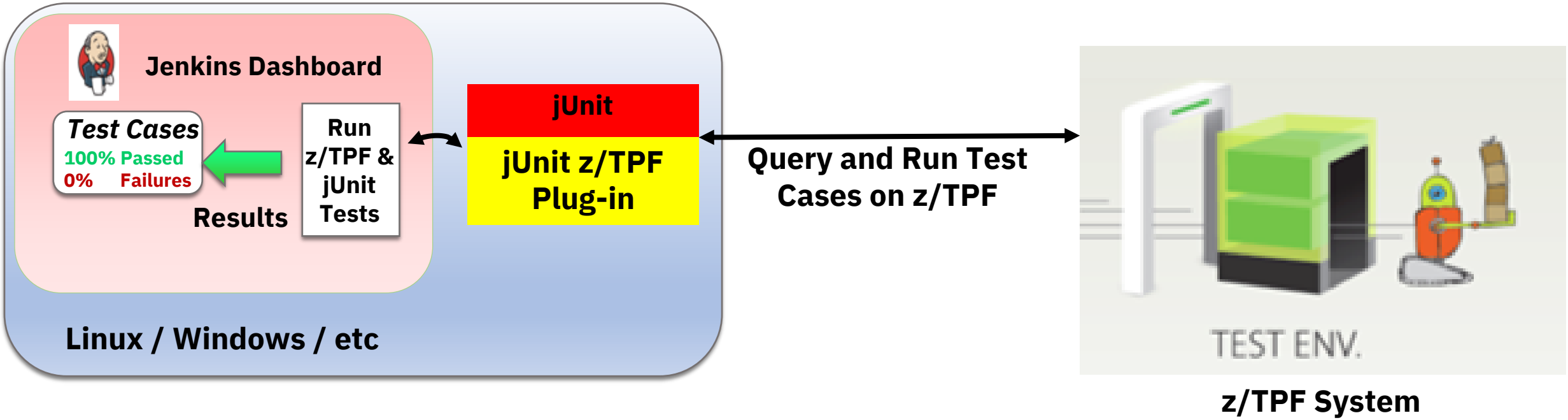
The jUnit Plugin for z/TPF

Provides the ability to run z/TPF automated tests from a Java application (ie. jUnit) on a remote platform

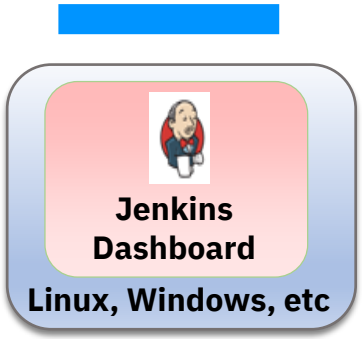


Integrating Into Automated Testing Platforms

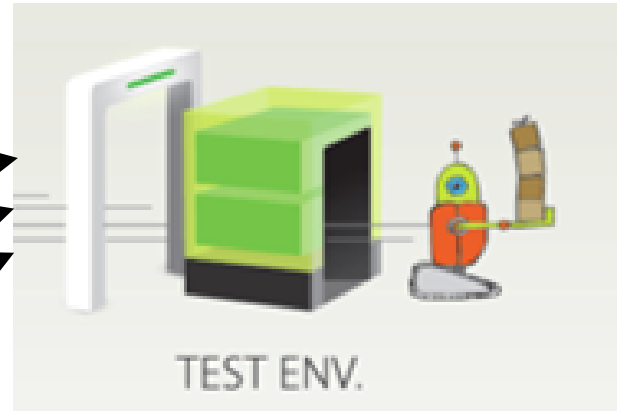
Can integrate into Open Tooling packages like Jenkins to facilitate automated testing

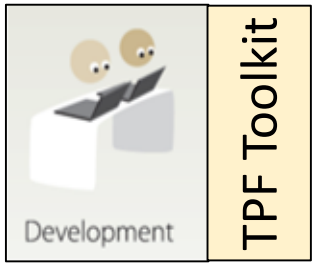


Expanding Test Coverage

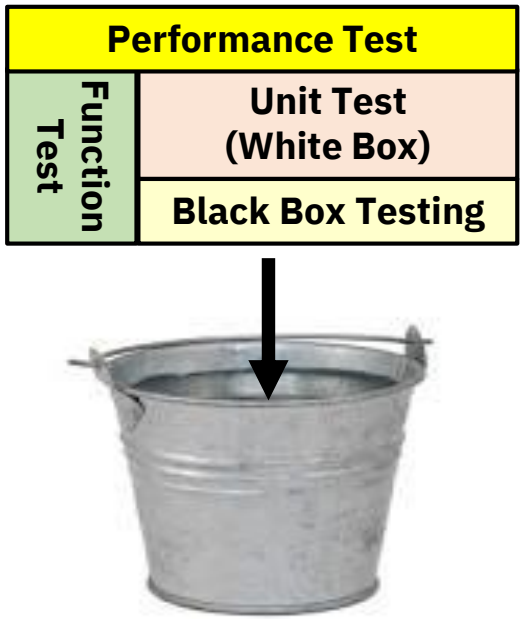


Query and Run Test Cases on z/TPF





- Tests can be invoked on z/TPF from anywhere!
- Tests residing on z/TPF can be combined with external block box tests
  - Testing end-to-end application messages



Automated Regression Test

Value Statement

- The z/TPF automated testing framework allows you to create automated tests on the z/TPF system
  - The test framework can be integrated into automated test platforms.

- A z/TPF automated testing environment…
  - Leads to more efficient and effective testing
    - Test organizations can focus on more complex testing
  - Shifts testing left to find problems earlier
    - Reduces costs when problems are found
  - Results in a faster time to market
  - Improved code quality.

Recap

- The z/TPF Automated Test Framework requires the following
  - **PJ45217, 1Q 2018** -- Infrastructure APAR
  - **PJ43782, 3Q 2018** -- Initial support (invocations from ZDEVO)
  - **PJ45488, 4Q 2018** -- Remote invocation support
    - Includes delivery and support of the z/TPF jUnit plugin.

Thank You!

Questions or Comments?

Trademarks

IBM, the IBM logo, ibm.com and Rational are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Notes

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.