CHALLENGES EXTENDING TEST AUTOMATION FOR VIRTUALIZATION

Sharookh Daruwalla
Intel Corporation

CONTEXT

- Automation essential to manage today’s validation workloads
- Focus mainly for Traditional OS validation (Windows, Linux)
- Extending for Virtualization OSes is very different
- In this presentation, the focus is on:
  - Challenges when extending Traditional automation frameworks to support Virtualization OSes
  - Mechanisms useful to overcome them
AGENDA

➢ Background

➢ Virtualization Automation Challenges

➢ Virtualization Automation Design

➢ Key Points & Experience
**Test Automation Frameworks**

Test Automation Frameworks include 3 components:

- **Test Schedulers** trigger test execution.
- **Test Execution Engines** run test scripts on SUT.
- **System Under Test (SUT) = System + OS**

**Traditional vs. Virtualization**

- **Traditional OS** — direct and dedicated control of system hardware.
- **Virtualization** — system hardware is shared.
- **System to OS** ratio change from 1:1 to 1:n.
- Automation Frameworks stuck on 1:1.
AGENDA

- Background
- Virtualization Automation Challenges
- Virtualization Automation Design
- Key Points & Experience

VIRTUALIZATION AUTOMATION: CHALLENGES

- Consider a simple test scenario
- SUT – Virtualization

Challenge 1: How to manage dynamic resources??!!
VIRTUALIZATION AUTOMATION: CHALLENGES

- Test Execution Engines support Windows or Linux
- But what OS are running on the VMs??

**Challenge 2:**
Need to support multiple OSes!

VIRTUALIZATION AUTOMATION: CHALLENGES

- Automation uses “Setup-Test-Teardown” per test
- Setup-Teardown times dependent on number of VMs
- Virtualization $\approx 2 \times$ Traditional OS

**Challenge 3:**
How to manage long setup-teardown times?
VIRTUALIZATION AUTOMATION: CHALLENGES

Challenge 1:
How to manage dynamic resources??!!

Challenge 2:
Need to support multiple OSes!

Challenge 3:
How to manage long setup-teardown times?

AGENDA

➢ Background

➢ Virtualization Automation Challenges

➢ Virtualization Automation Design

➢ Key Points & Experience
**DESIGN: DYNAMIC RESOURCES**

Dynamic Resource Management:
- **Record systems in a test execution**
- **Pointer** to control various systems
- System Table holds key information
- **Pointer** helps control test execution correctly
  - Operations executed on system pointed by **Token**

<table>
<thead>
<tr>
<th>Host</th>
<th>OS, Auth, IPAdd, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM1</td>
<td>OS, Auth, IPAdd, ...</td>
</tr>
<tr>
<td>VM2</td>
<td>OS, Auth, IPAdd, ...</td>
</tr>
<tr>
<td>VMn</td>
<td>OS, Auth, IPAdd, ...</td>
</tr>
</tbody>
</table>

**DESIGN: MULTI-OS SUPPORT**

- Libraries categorized into 2 types
- **Common Libraries:**
  - Features common to all OSes – traditional and virtualization
  - E.g. `SendSystemCommand()`
- **Virtualization Libraries:**
  - Virtualization specific features only
  - E.g. `NewVM()`
DESIGN: TEST CONTINUATION MECHANISM

- Virtualization Setups include:
  - Host Setup
  - Virtual Machine setup

- Teardown is just the reverse of setup
- Setup-teardown times dependent on number of VMs
- Setup-teardown can dominate total test time

DESIGN: TEST CONTINUATION MECHANISM

- Group tests with similar setups
- Full setup for FIRST test
- Full teardown for LAST test
- Minimal setup-cleanup within tests
AGENDA

- Background
- Virtualization Automation Challenges
- Virtualization Automation Design
- Key Points & Experience

ITE-BERTA TEST AUTOMATION FRAMEWORK

- Built using in-house tools:
  - Integrated Test Environment (ITE)
  - Berta - test-scheduler
- Framework extended to support VMware ESXi
  - Other hypervisor (HyperV, KVM) support planned
- Libraries extended to support Linux
- Libraries support Dynamic Resource Management
- Test Continuation in-progress
PERSONAL EXPERIENCE

➢ Three Firsts – Validation, Automation and Virtualization
➢ Re-using existing framework to create common is possible
➢ Scripts satisfy dual purpose: Full and Selective Automation
➢ Script-in-script execution slow; debugging very painful

THANK YOU
**BACK-UP SLIDES**

**ITE-BERTA TEST AUTOMATION FRAMEWORK**

- **Built using two in-house tools:**
  - Integrated Test Environment (ITE)
  - Berta

- **ITE:**
  - Test Execution Engine
  - Windows OS Validation

- **Berta:**
  - Test Scheduler
  - Maintains test database (SUT, tests, etc.)
  - Provides metrics, send notification, etc.
**CHALLENGES: COMPLEX SETUP-TEARDOWN**

<table>
<thead>
<tr>
<th></th>
<th>Traditional OS (mins)</th>
<th>Virtualization OS (mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Establish connection between SUT and Target</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Add required LUNs</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Setup VMs (DataStores, VMs, OS, LUNs)</td>
<td>-</td>
<td>50</td>
</tr>
<tr>
<td><strong>Setup Time:</strong></td>
<td>29</td>
<td>85</td>
</tr>
<tr>
<td><strong>Run Test: (SAN IO for various packet sizes)</strong></td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Remove VM Setup (delete DataStores, VMs, etc.)</td>
<td>-</td>
<td>20</td>
</tr>
<tr>
<td>Release LUNs, SUT and Target connections</td>
<td>24</td>
<td>20</td>
</tr>
<tr>
<td><strong>Teardown Time:</strong></td>
<td>24</td>
<td>40</td>
</tr>
<tr>
<td><strong>Total Test Time:</strong></td>
<td>83</td>
<td>155</td>
</tr>
</tbody>
</table>

**DESIGN: DYNAMIC RESOURCE MANAGEMENT**

- Implement in two stages:
  - Recording new resources
  - Hand control to correct system (token)

- Extend supports for 3 types of systems:
  - **SUT** — Initially host
  - **PEER** — Traffic Partners
  - **VM** — Test Virtual Machine

- Toggle control to correct system:
  - Token (**controlSystem**) hands control
DESIGN: TEST CONTINUATION MECHANISM

- Need to reduce setup-teardown times for virtualization testing!
  - Takes 2x Time versus Traditional OS
- Test Continuation helps reduce test times for test groups
- The Idea involves:
  - Classify/Group tests with similar setups
  - Full setup for FIRST test
  - Minimal clean-up within group
  - Full teardown for LAST test

OTHER SLIDES
CONTEXT

- Automation been mainly focused for Traditional OSes (Windows, Linux)
- Extending automation for Virtualization OSes/Hypervisors is very different
- In this presentation, we present:
  - Challenges with extending automated validation for virtualization OSes
  - Mechanisms used to overcome them
- We present our internal ITE-Berta Automation Framework used to:
  - Extend a Windows-only automation framework to support Virtualization Hypervisors (VMware)

STORAGE NETWORKING

- Involves Storage Area Networks (SAN)
- SANs are dedicated networks to access consolidated data storage
  - Initiators are servers initiating RD/WR data requests
  - Targets are data storage devices that service these requests
  - LUNs are data storage hard-drive (logical unit numbers)
- Protocols include Fiber Channel (FC), iSCSI, FC over Ethernet (FCoE)
ITE-BERTA Test Automation Framework

- No-Touch Automation Solution
  - Monitors for new component and project builds
  - Triggers BAT testing on new builds
  - Triggers Regression testing on BAT-certified builds
  - Notifies key stakeholders

Example Test-Case Scenario

- Comparing Setup, Runtime, Teardown for Traditional vs. Virtualization

<table>
<thead>
<tr>
<th>Traditional OS</th>
<th>Virtualization OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup SUT and Target (FCF-FSW setup)</td>
<td></td>
</tr>
<tr>
<td>Add 1 LUN of required size</td>
<td>Add 5 LUNs of required size</td>
</tr>
<tr>
<td>n/a</td>
<td>Create a DataStore on the LUNs</td>
</tr>
<tr>
<td>n/a</td>
<td>Create 5 VMs with Windows/Linux OSes</td>
</tr>
<tr>
<td>n/a</td>
<td>Attach a LUN to each VM</td>
</tr>
<tr>
<td>Run traffic between SUT and Target</td>
<td>Run traffic between VMs and Target</td>
</tr>
<tr>
<td>Clean-up Setup</td>
<td></td>
</tr>
</tbody>
</table>
EXAMPLE TEST-CASE SCENARIO

Traditional

Virtualization

AUTOMATION INFRASTRUCTURE SETUP
VIRTUALIZATION AUTOMATION: CHALLENGES

- Virtualization OSes or Hosts allow VMs to work independently
- Virtualization OS validation involves both Host and VM testing
- The main challenges:
  
  STATIC RESOURCE MANAGEMENT
  
  SINGLE OS SUPPORT
  
  COMPLEX AND TIME-CONSUMING SETUP AND TEARDOWN

EXAMPLE TEST SCENARIO

Virtualization

LAN/SAN Traffic

SUT

VM

FCF

FSW

Target
**CHALLENGES: STATIC RESOURCE MANAGEMENT**

- Test execution engines use STATIC resource management
  - Resources selected before tests
  - Resources cannot change within tests
  - No communication-control with new resources

- Problems for Virtualization
  - VMs are virtual test resources (SUTs)
  - VMs created within tests
  - Need to communicate with VMs within tests

---

**Example:**

- Before VMs, SUT → ESXi HOST
- Later, VMs become the SUT (Host becomes manager)
- Run IO to-from the VMs
- Test execution engine cannot manage a system that it is not aware of!
CHALLENGES: STATIC RESOURCE MANAGEMENT

- Example Test Scenario:
  - Before VMs, SUT → ESXi HOST
  - Later, VMs become the SUT (Host becomes manager)
  - Test requires running IO between the VMs and Target
  - Test execution engine cannot manage a system that it is not aware of!

CHALLENGES: SINGLE OS SUPPORT

- Test Execution Engines typically support one OS
  - Most support either Windows or Linux
  - However, VMs created can run either Windows or Linux OS
  - Need to communicate-control with all VMs to complete test (start IO)

- Need to extend support for multiple OSes
  - Need to add support for Windows, Linux, ESXi
  - Complicated feature – Compiler support and/or Library support?
**CHALLENGES: COMPLEX SETUP-TEARDOWN**

- Setup and Teardown occurs for each test in automation
- Setup/Teardown is complex and time-consuming for virtualization
  - Setup/Remove Host, VMs, etc.
- Directly dependent on number of VMs

In current form, Virtualization Automation would require 2x hardware to run similar number of tests in the same time as Traditional Automation.
CHALLENGES: COMPLEX SETUP-TEARDOWN

- Traditional vs. Virtualization Times (on-average):
  - Setup takes ~3x longer
  - Teardown takes ~2x longer
- Total test time ~2x longer for virtualization

In current form, Virtualization Automation would require 2x hardware to run similar number of tests in the same time as Traditional Automation

VIRTUALIZATION AUTOMATION: DESIGN

<table>
<thead>
<tr>
<th>PowerCLI</th>
<th>C# Wraper</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>PowerCLI for infrastructure</td>
<td>Existing C#, VB for infrastructure</td>
<td>Existing C#, VB for infrastructure</td>
</tr>
<tr>
<td>PowerCLI for all Host Ops</td>
<td>PowerCLI in C# for Host Ops</td>
<td>PowerCLI in VB for Host Ops</td>
</tr>
<tr>
<td>PowerCLI for all tests</td>
<td>VB for tests</td>
<td>VB for tests</td>
</tr>
<tr>
<td>Native ESX support</td>
<td>Leverage existing framework</td>
<td>Leverage existing framework</td>
</tr>
<tr>
<td>Lower Maintenance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specific to ESX Virtualization</td>
<td>Double wrapping of ESX cmdlets</td>
<td>PowerCLI execution slower</td>
</tr>
<tr>
<td>Re-invents a new framework</td>
<td>Higher Maintenance</td>
<td>PowerCLI debugging painful</td>
</tr>
<tr>
<td></td>
<td></td>
<td>High Maintenance</td>
</tr>
</tbody>
</table>
**VIRTUALIZATION AUTOMATION: DESIGN**

![Diagram of Test Scheduler, Test Execution Engine, System under Test, Virtualization OS, Libraries, Infrastructure (VB Script, C#), and Host (PowerCLI)]

**SUMMARY**

- Extending automation for virtualization has challenges:
  - Static resource management
  - Single OS support
  - Long setup and teardown times

- We presented mechanisms that support:
  - Automation for both traditional and virtualization OSes
  - Dynamic resource management
  - Multiple traditional and virtualization OSes
  - Test Continuation Mechanism (to reduce setup-teardown times)