



**How To
Create a Continuous Testing
Environment with the TestComplete
REST Api and Virtual Machines**



Introduction

Stability and productivity are everything with automated testing. If you're going to be productive with TestComplete you must have a way to test and run your automated tests outside of your development machine. As your tests grow you cannot afford to waste time tying up your development machine as you run your automated tests. It's crucial that you have separate environments where you can execute tests independently.

We're going to walk you through a simple-ish setup where you develop on one machine and run on a separate virtual machine using TestExecute. Execute a script, your tests run on a VM, results available on your development machine.

What You'll Learn

When you follow this "how to" document you'll learn:

- i. How to configure a VM and install TestExecute
- ii. How TestComplete/TestExecute can work in 'shared' mode (sharing access to a projects files)
- iii. How to use the TestComplete API to automatically run your tests
- iv. How to capture and save test results in an Html format that you can access on a shared drive

Ultimately, you'll learn how to increase your productivity and increase the stability of your automated tests.

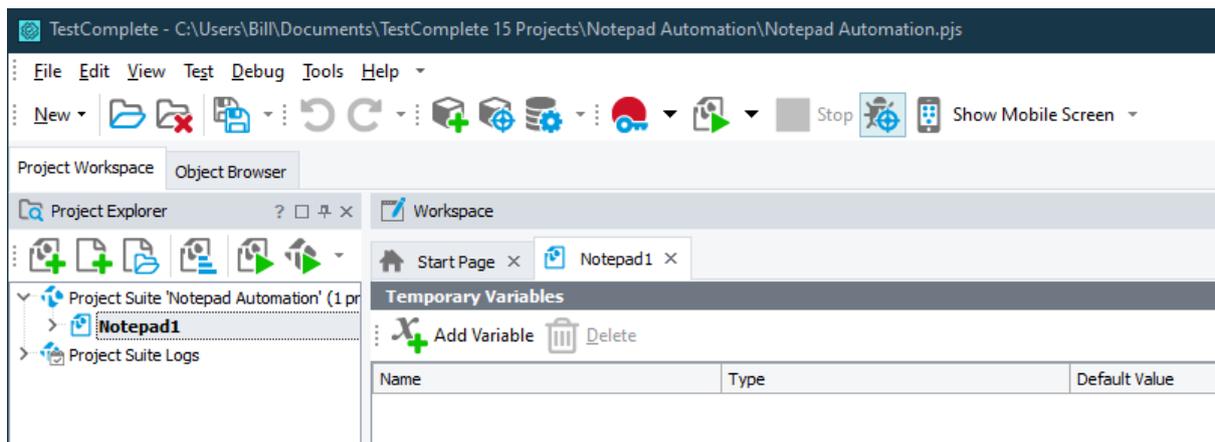
Stage 1 – Create Your TestComplete Project

We need to setup a development environment where you can develop tests. We'll refer to this as your main desktop/laptop machine. On this main machine we'll develop a simple automation script. We'll create a script that just opens Notepad, creates some content and then closes notepad.

1. Open TestComplete on your main machine, create a new project suite and create a new project. You can name them...

Project Suite	=> Notepad Automation
Project	=> Notepad1

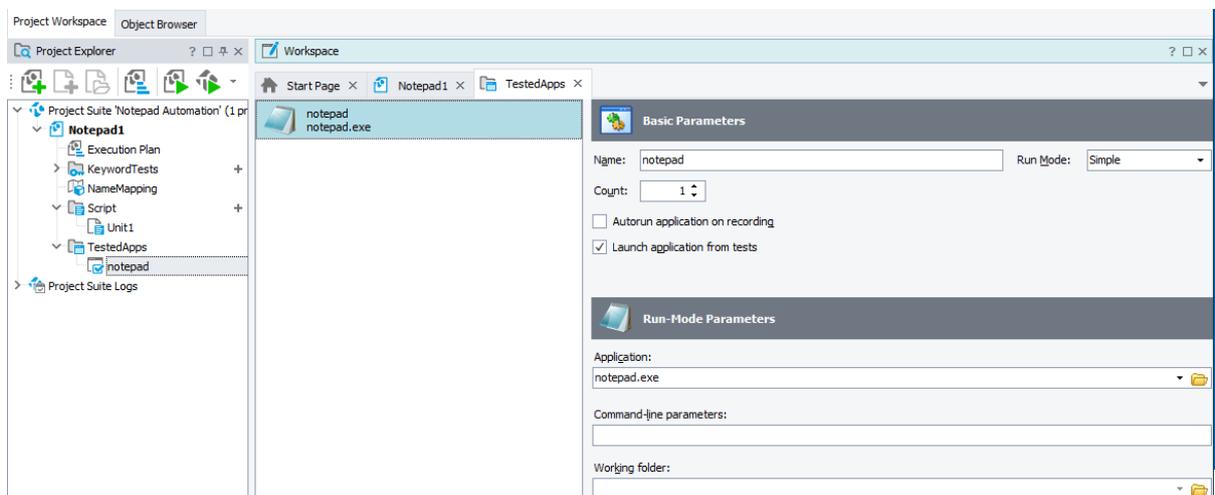
You should have something like this:



2. Add 'notepad.exe' as a Tested App

You may need to Add the 'TestedApps' item to your project. (right click on the project node, select 'New Item', then 'TestedApps'. Then create a new Tested App :

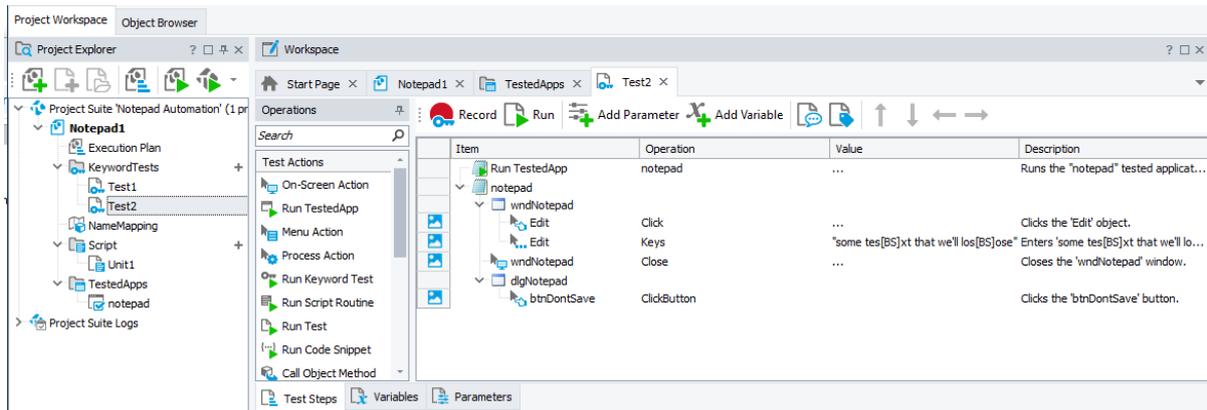
- a. select 'Generic Windows Application'
- b. for 'Application' just enter "notepad.exe"
- c. click the 'Finish' button



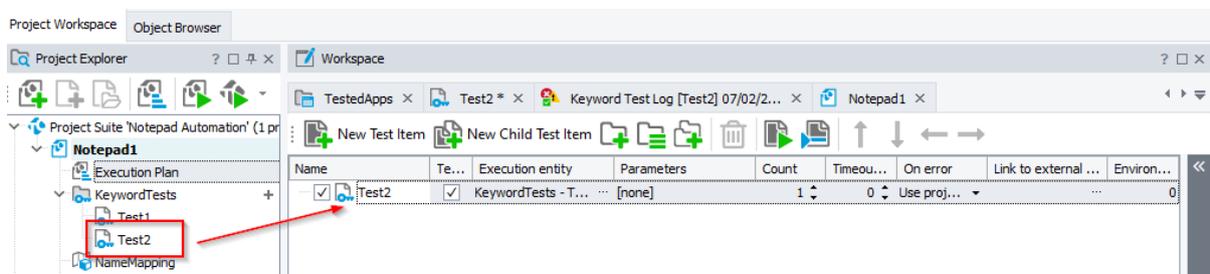
3. Record a keyword script where you...

- a. open notepad
- b. add some text
- c. close note pad (click 'Don't save' when closing notepad)
- d. discard / don't save

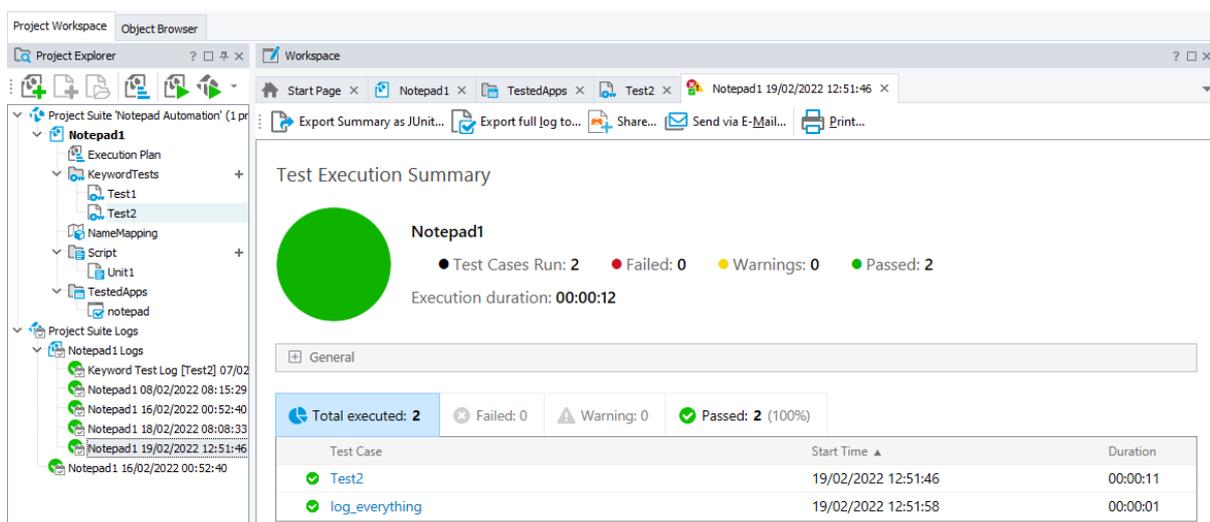
Then save this keyword script as 'Test2'.



4. Add this test to your Execution Plan



5. Run this project several times locally. Make sure that it runs consistently and passes consistently.



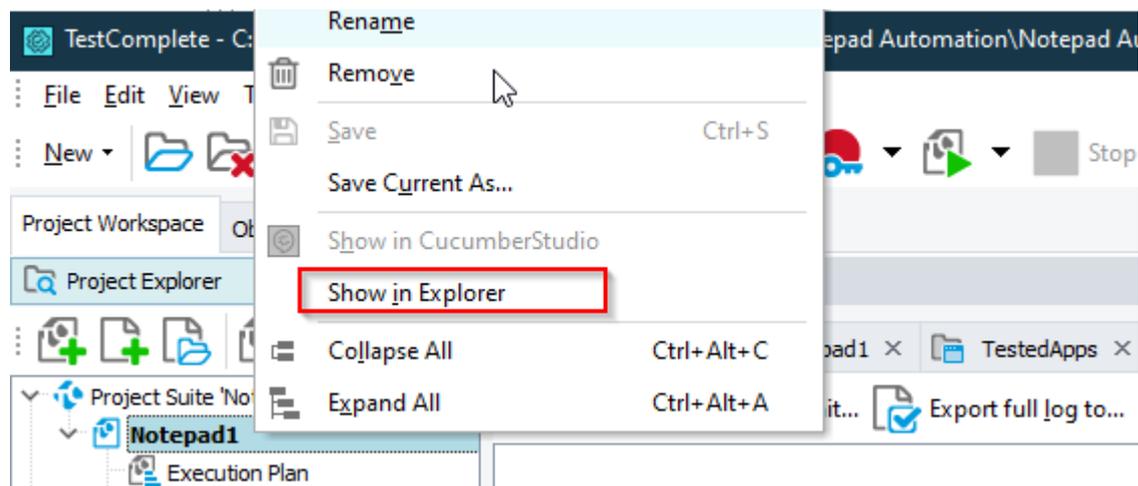
Don't get caught up in developing some advanced script or even something that looks exactly like the script above. You just need something that runs, and passes, consistently.

Stage 2 – Project Folder Share

Next, we need to make sure that this script can be shared across multiple machines. We need to share the project and the code between your local machine/laptop and your virtual machine. We'll do this using a windows file share.

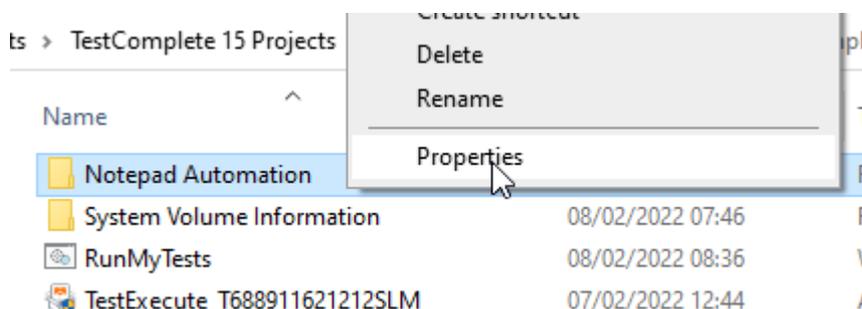
Now you may already have a file share or you may need to set one up. If you need to set one up then follow these steps:

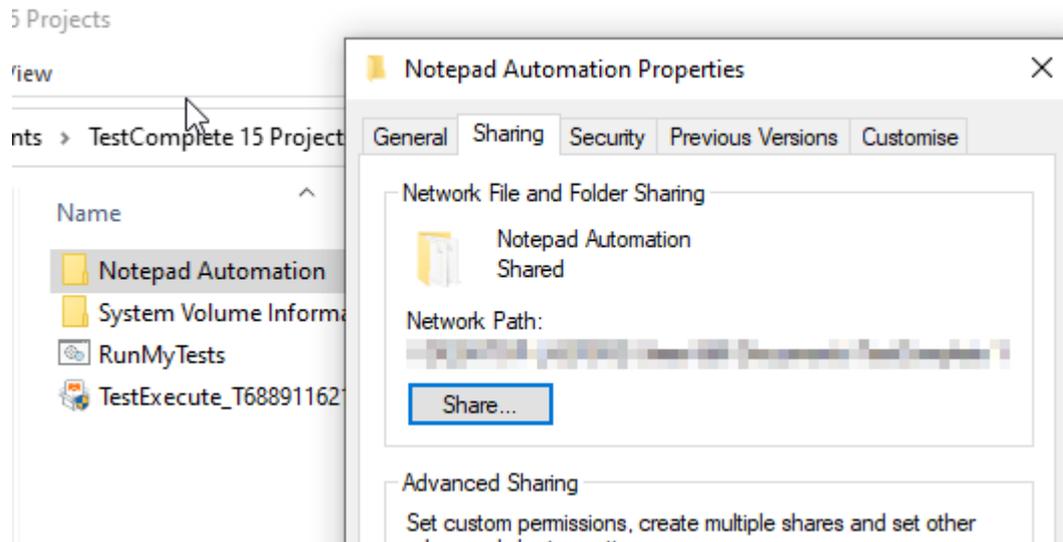
1. right click on your Project suite node and select 'Show in explorer'



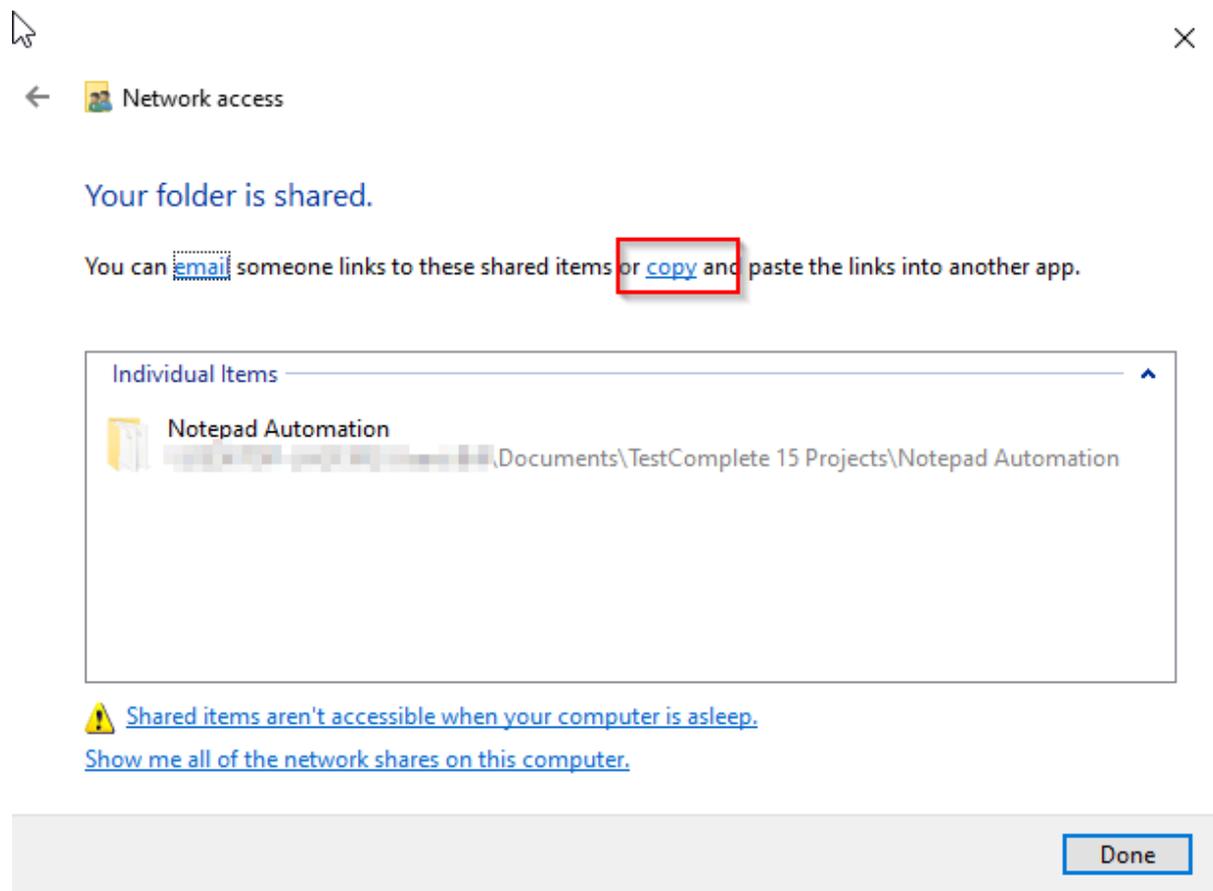
2. Now we know where our Project suite stored, we can share this folder.

- a. go up one level in explorer
- b. right click and select 'properties'
- c. on the 'Sharing' tab select the 'share' button





3. Once you've shared your folder you will want to copy the full UNC path so that you can access the share from your VM.



You'll have a UNC path something like this...

```
//YOUR_MACHINE_NAME/Users/USER_NAME/Documents/TestComplete%2014%20Projects/NotepadAutomation
```

Keep a record of this as we'll need it on your VM in a minute

Stage 3 - Virtual Machine setup

We need a machine to 'develop' our tests on. We also need a separate machine to 'execute' our tests on. You can use your desktop or laptop for development with TestComplete. Then use a Virtual Machine, running TestExecute, to execute your test. This gives you the capability for testing/shake down of your tests or for formal execution runs. Either way you free up your main machine and increase your productivity.

If you already have a Virtual Machine, you can skip through to Stage 4 below. If you don't already have a VM then you can download Virtual Box from here and run the installer:

<https://www.virtualbox.org/wiki/Downloads>

From the same web page, download and Install the Virtual Box Extension Pack (which includes support for Remote Desktop connections).

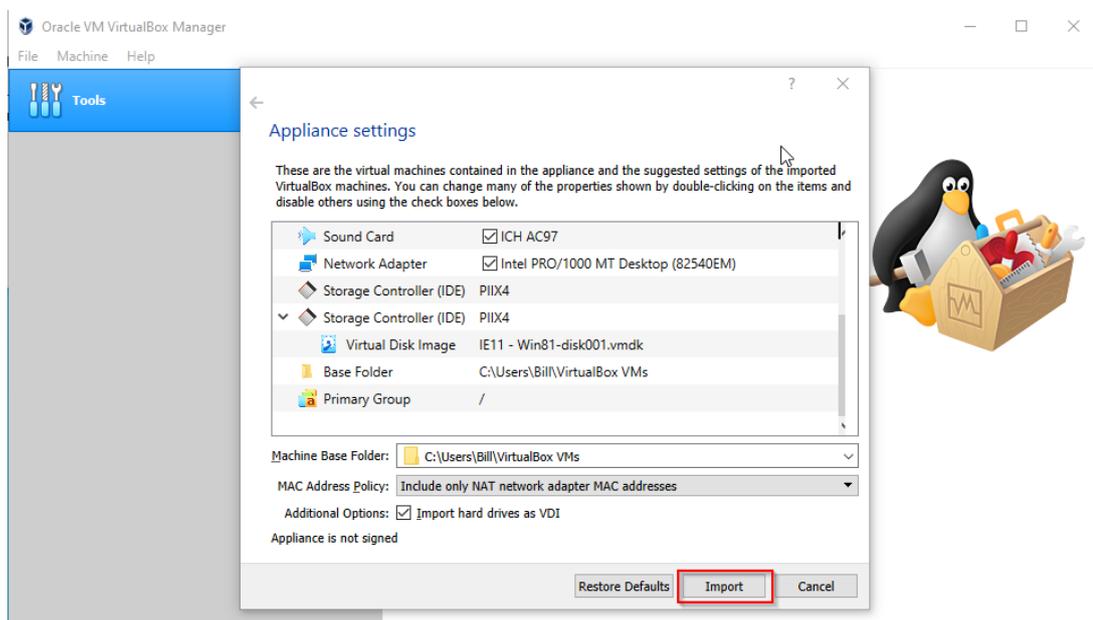
VirtualBox 6.1.32 Oracle VM VirtualBox Extension Pack

• [All supported platforms](#)

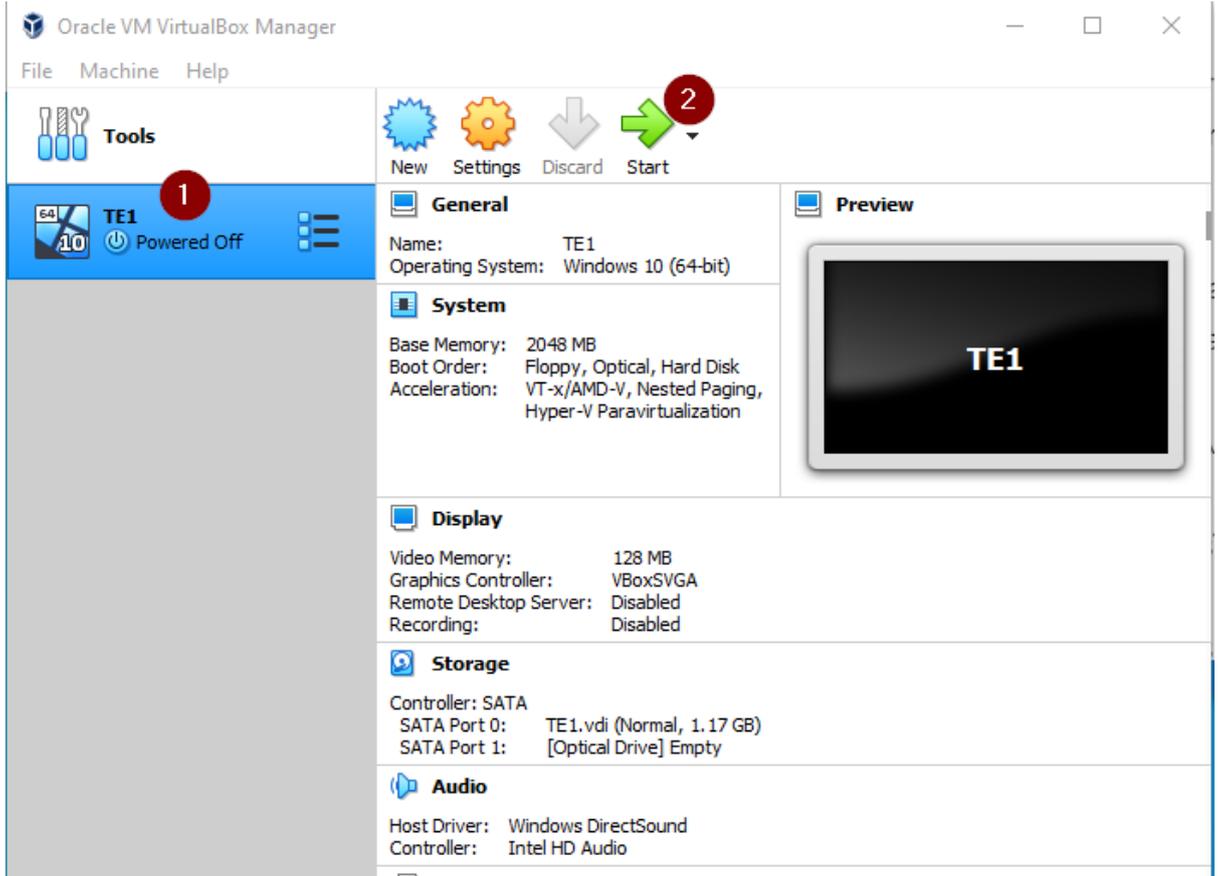
Support for USB 2.0 and USB 3.0 devices, VirtualBox RDP, disk encryption, NVMe and PXE boot for Intel cards. See [this chapter from the User Manual](#) for an introduction to this Extension Pack. The Extension Pack binaries are released under the [VirtualBox Personal Use and Evaluation License \(PUEL\)](#). Please install the same version extension pack as your installed version of VirtualBox.

Once you're running with Virtual Box then you can follow these steps to create and configure a Virtual Machine.

1. Download and extract a Virtual Box Windows image from:
<https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/>
2. Double click the .ova (Open Virtualization Archive) file and import this file in Virtual Box

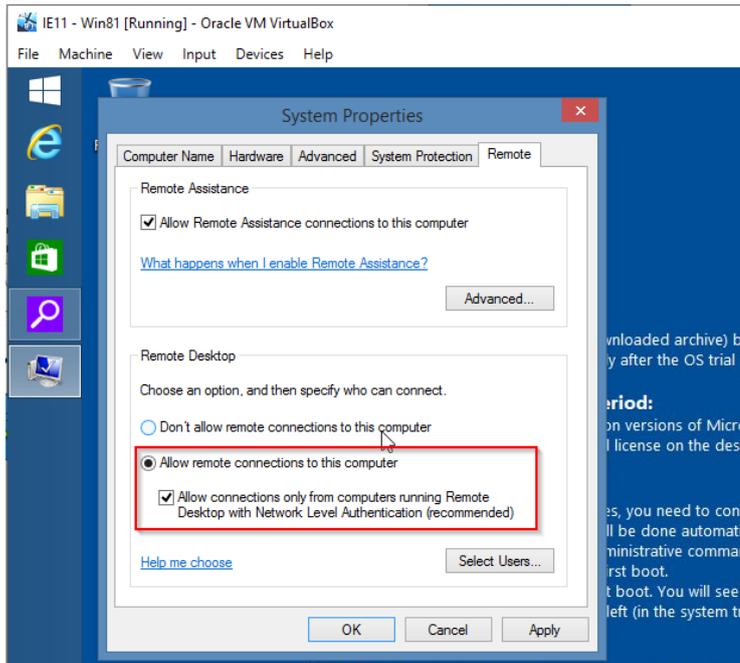


3. Start the new box

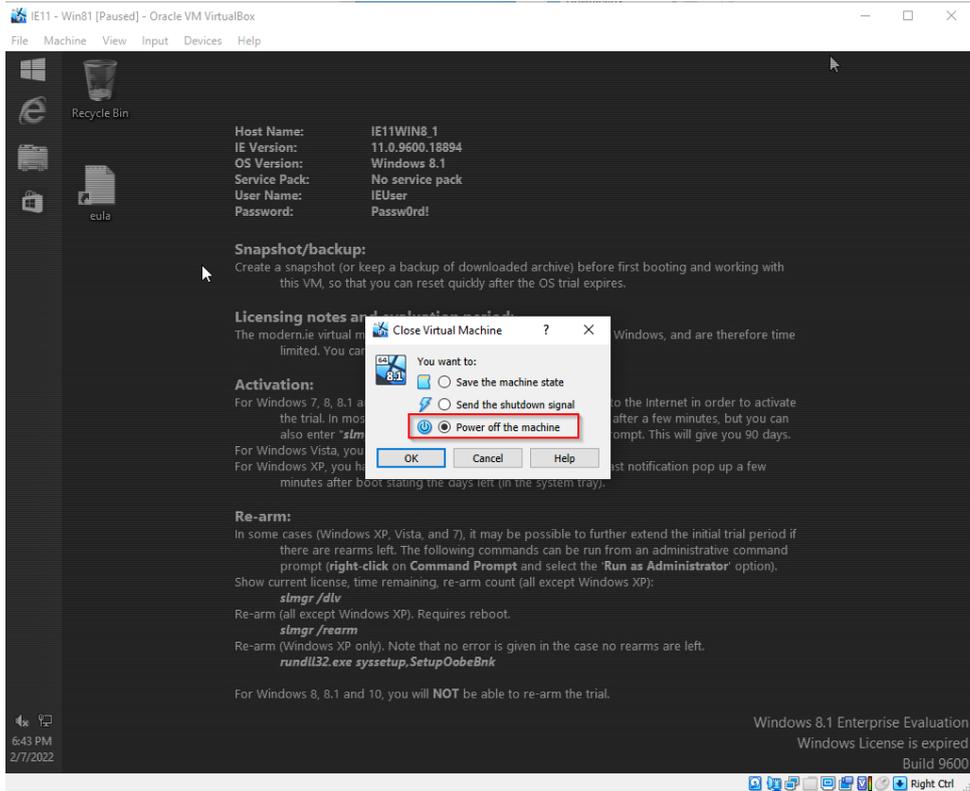


4. Check this new box runs okay, capture the IP address, User Name and password (if you've downloaded the Microsoft image then the user name will be "IEUser" and the password "PasswOrd1")

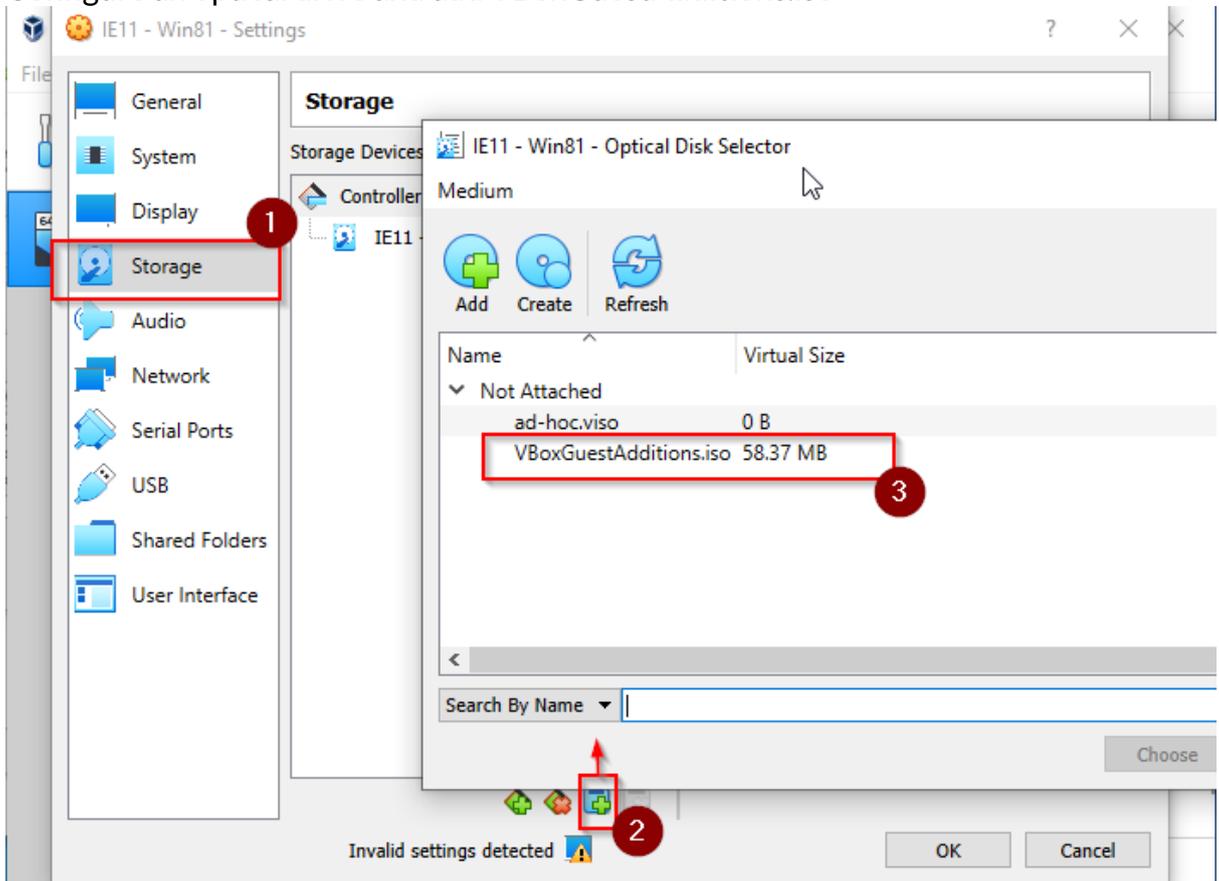
5. Enable "Remote Access" on this machine



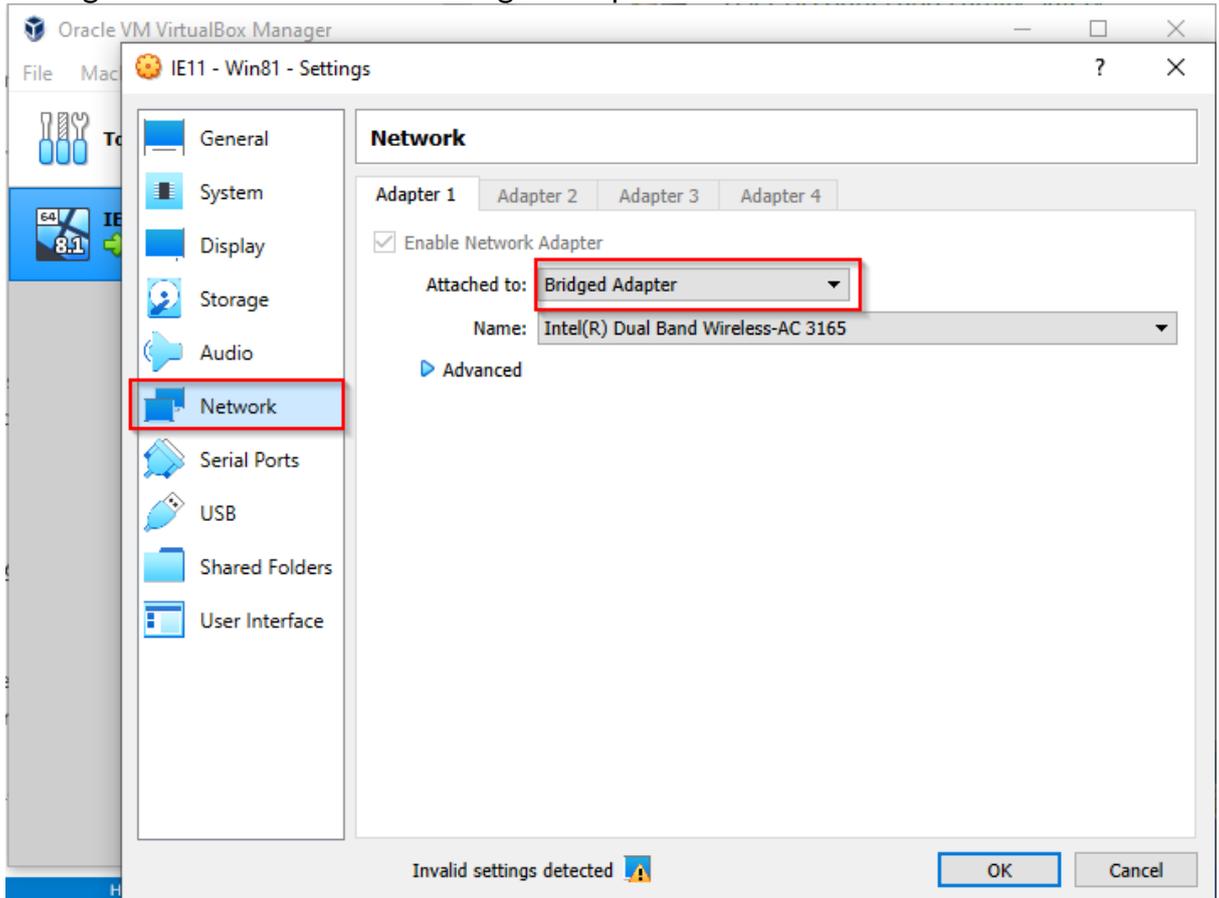
6. Close and Power Off the new box



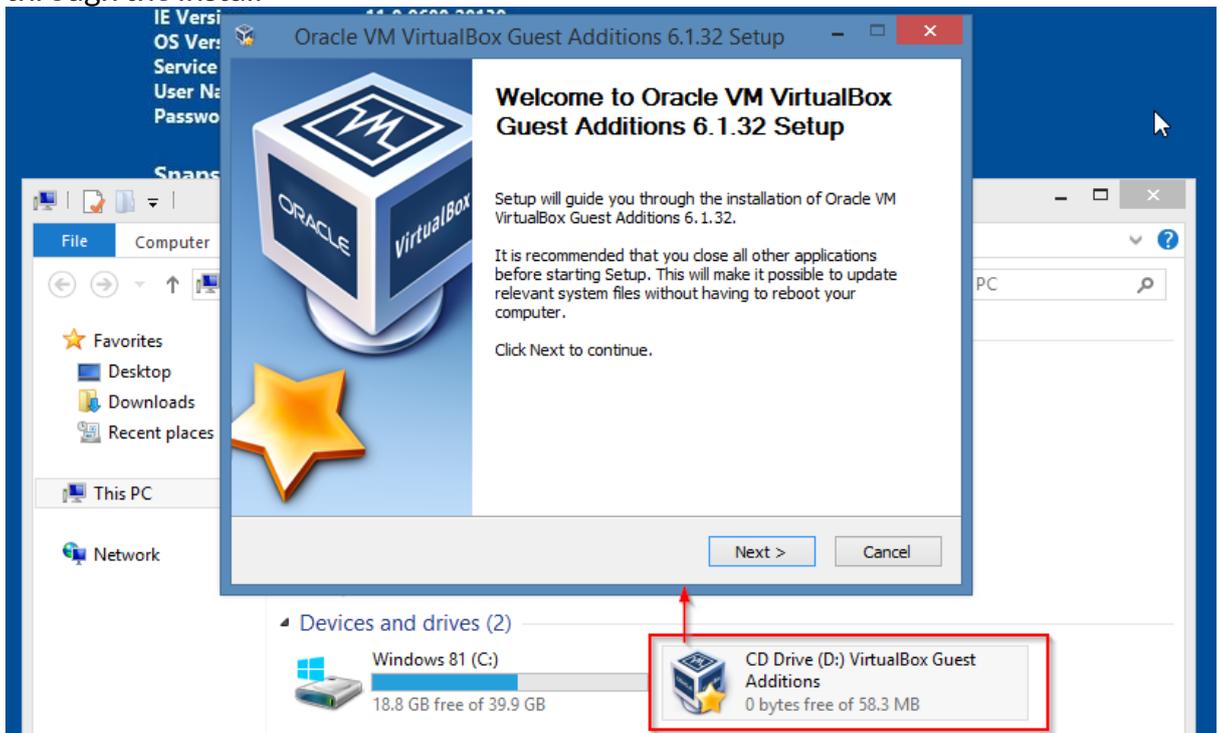
7. Configure an optical drive and add VBoxGuestAdditions.iso



8. Configure the network to be a 'Bridged Adapter'



9. Restart the box, go to file explorer, open VirtualBox Guest Additions and run through the install



7. Shut the machine down, then...

8. Restart the new box

At this point you should have a Virtual Machine available and running. You must be able to RDP on to this machine and create a desktop session. From here we'll need to install TestExecute.

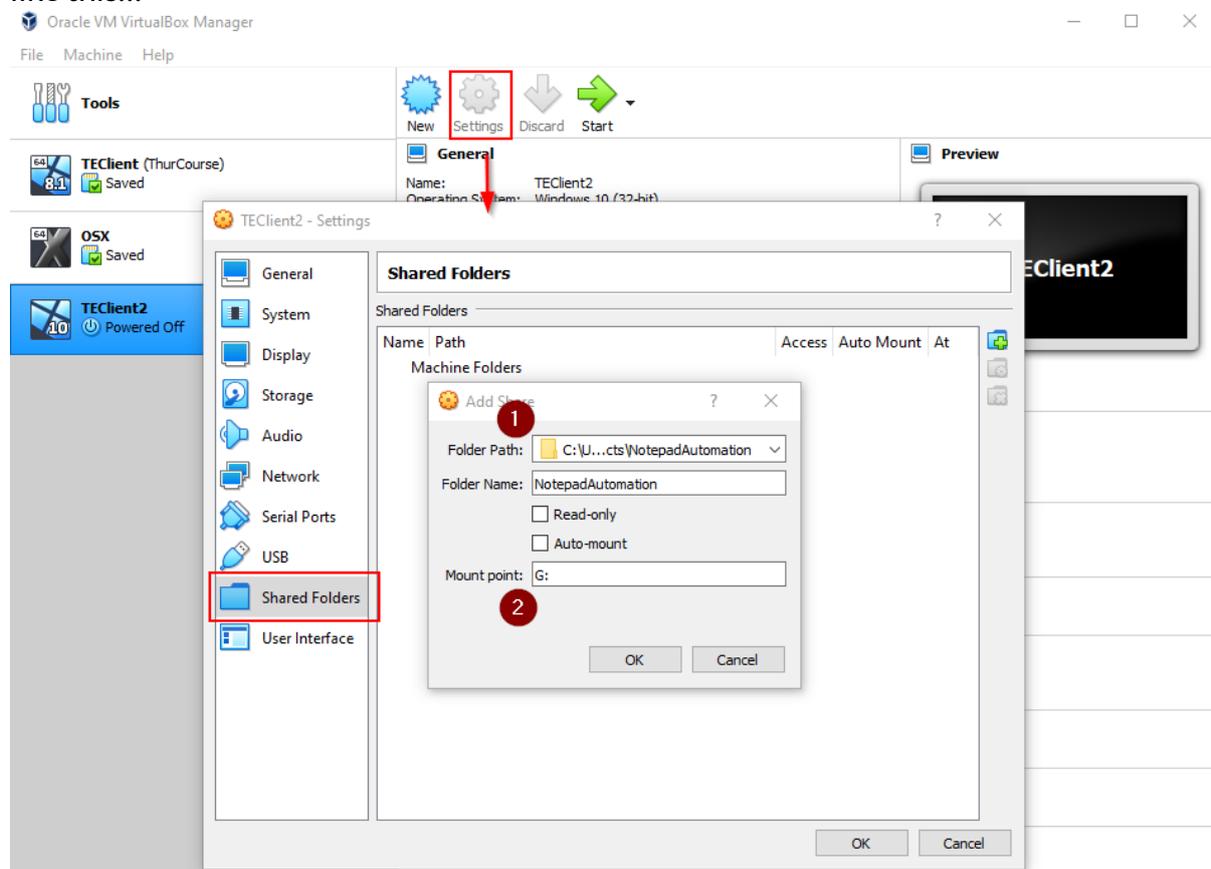
9. RDP onto the Virtual Machine - use 127.0.0.1:5958
(you must RDP onto the "host" machine on the specified port)
(more info here if you have problems:

<https://marcowen.wordpress.com/2016/10/31/troubleshooting-virtualbox-vrde-remote-desktop-connections/>)

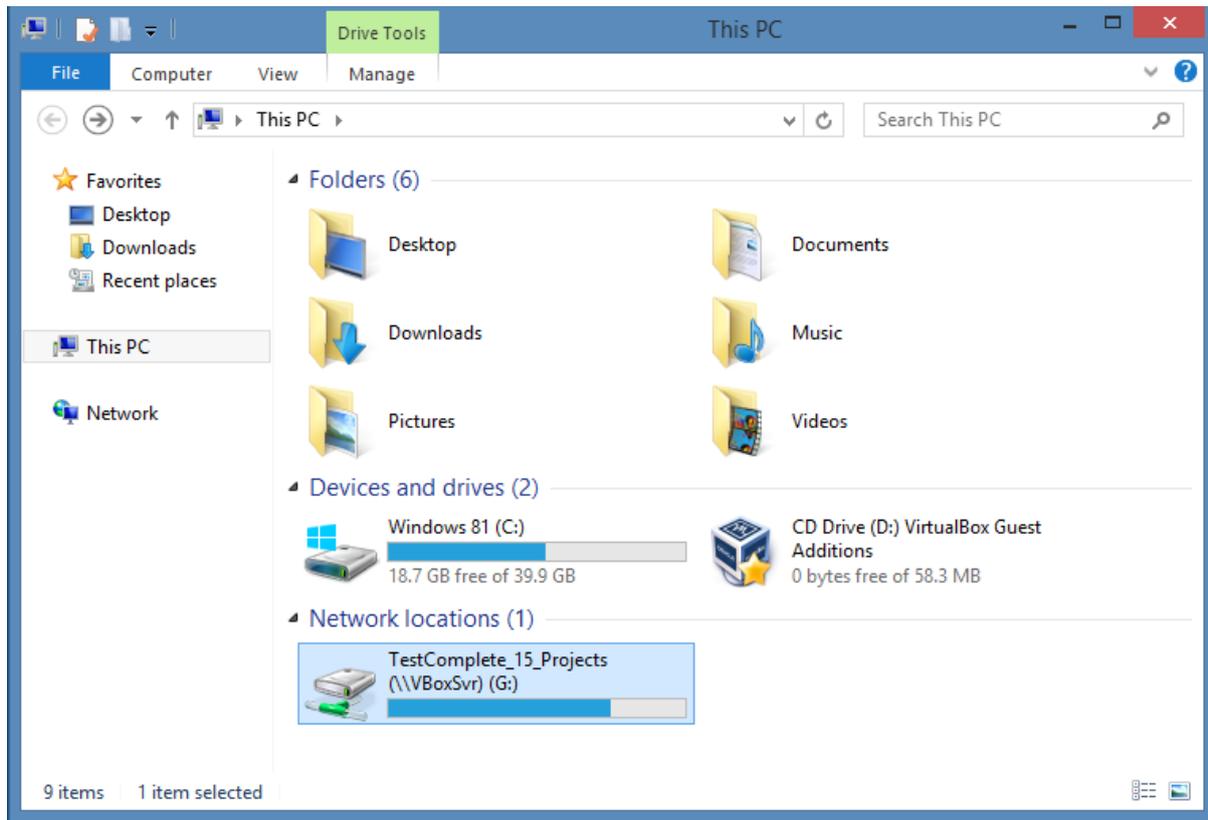
10. With the shared folder you created on Desktop/Laptop machine, use the UNC recorded (above in Stage 2) and paste the address into the File Explorer address bar (you will probably have to change the " separators to '/' seperators to get this to work)

\\YOUR_MACHINE_NAME\Users\USER_NAME\Documents\TestComplete 14
Projects\NotepadAutomation

You should be able to access this shared directory on your Virtual Machine something like this...



At this point, File Explorer on your VM should show the mounted driver that contains your TestComplete projects



11. Install TestExecute on this Virtual Machine

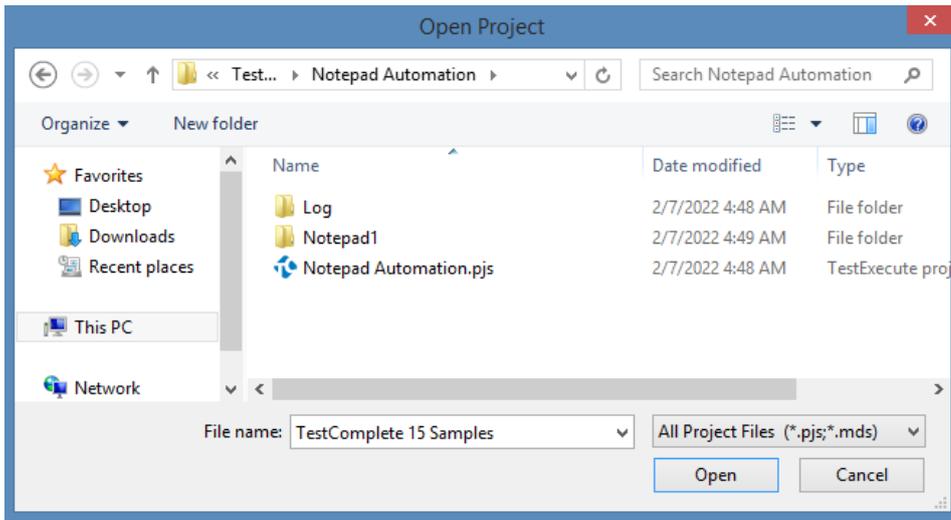
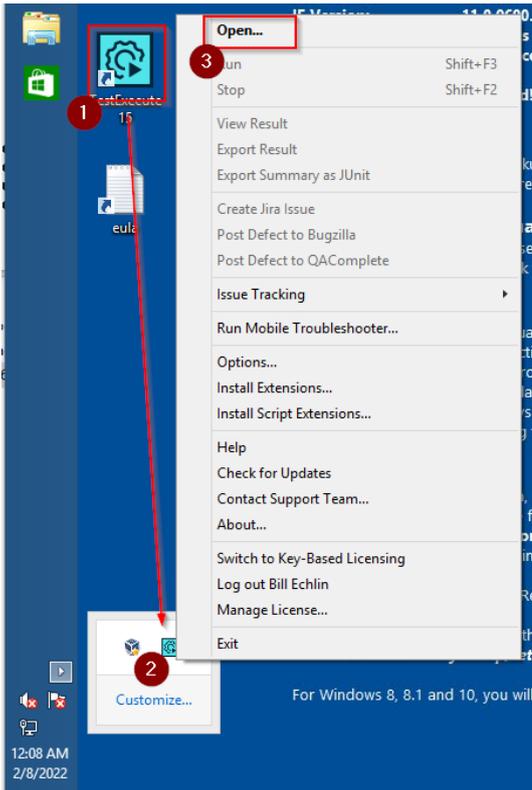
You can download a TestExecute trial using this link:

<https://smartbear.com/services/trials/downloadtrial.aspx?file=TestExecute1530.exe>

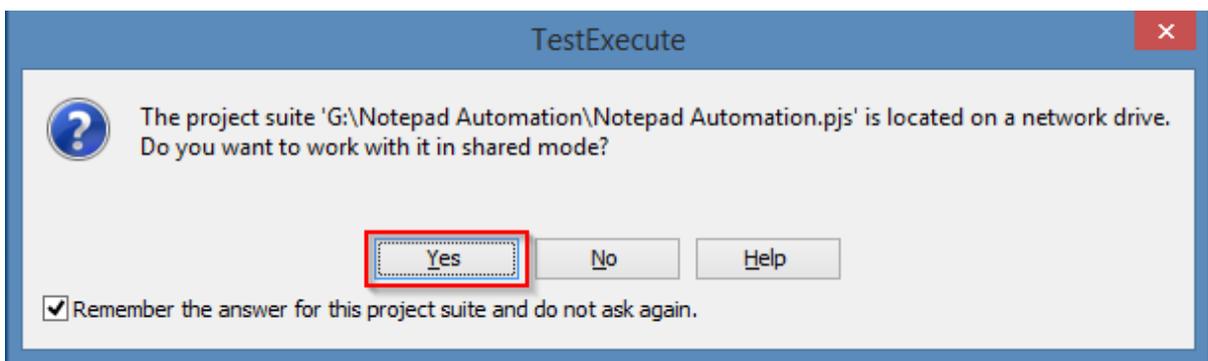
Stage 4 - Run the TestExecute Project on the Virtual Machine

We should now be able to run our project, written in TestComplete, on our VM using TestExecute. When running a project from a shared drive like this, it's known as running projects in 'Shared Mode'.

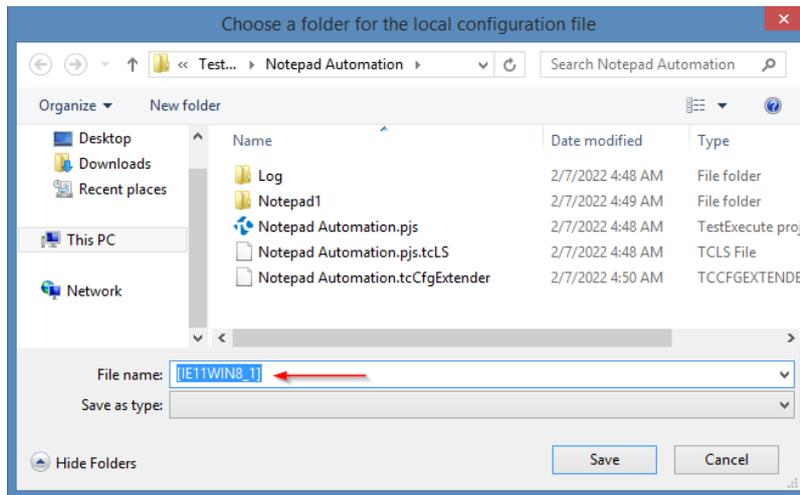
1. Start TestExecute and open the 'notepad1.mds' project file



When you open this project from a shared drive you'll be presented with this prompt...



You'll want to select 'Yes' and then save the configuration file in the same directory as your project suite. You should see a directory created like this...



In shared mode, TestComplete creates separate configuration files for each team member who runs the tests. In addition, TestComplete organizes the test logs by creating separate log folders in the log tree for each test computer where the tests run.

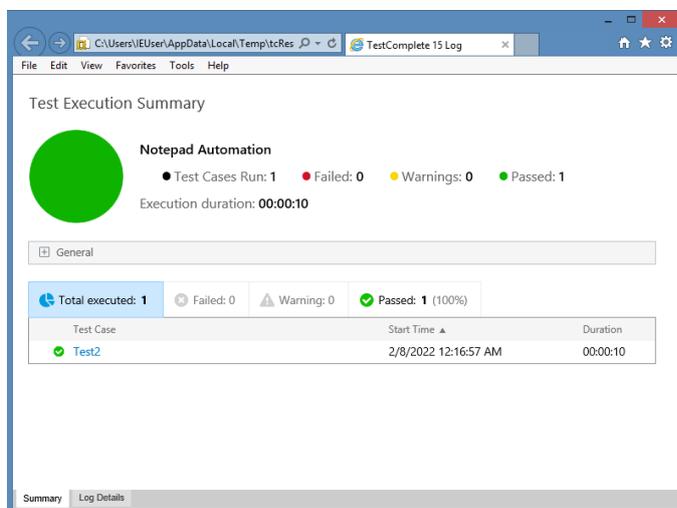
You'll learn more about this here:

<https://support.smartbear.com/testcomplete/docs/working-with/teamwork/sharing-projects.html>

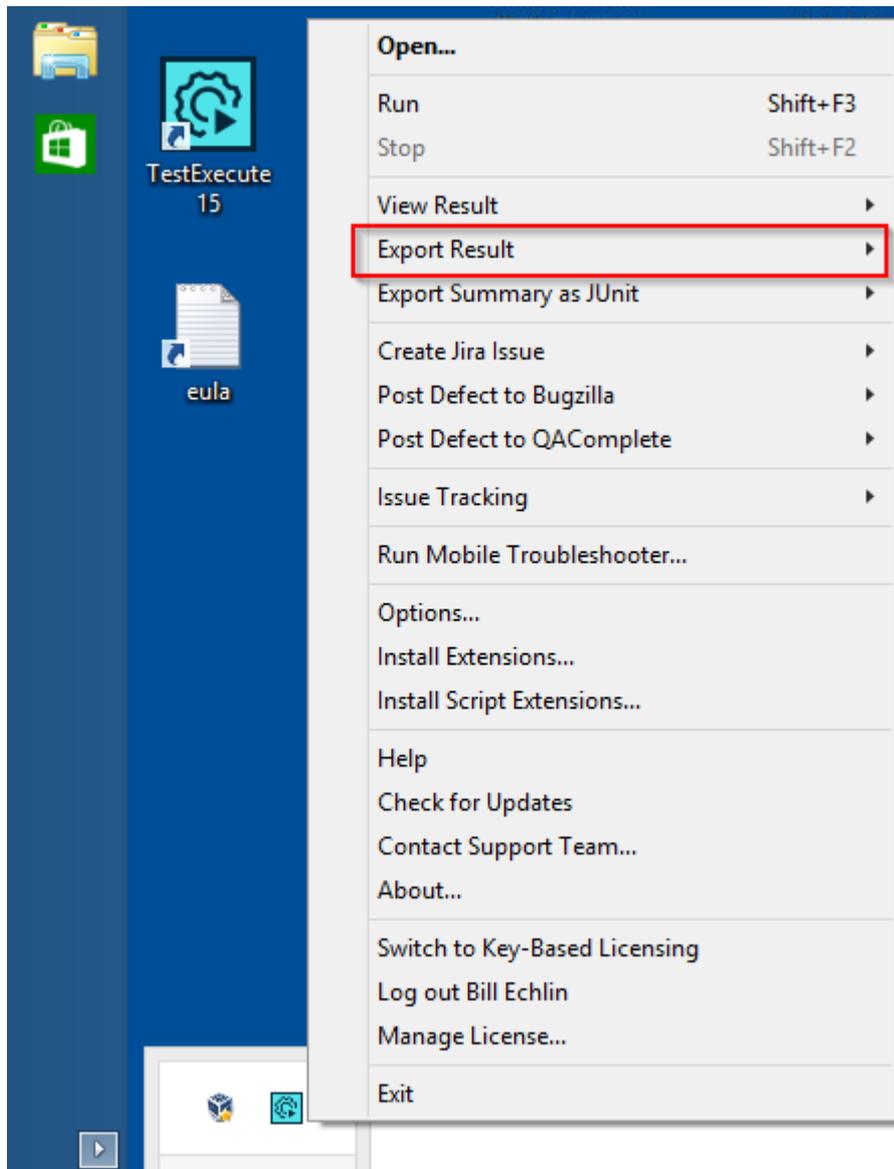
Finally run your project on your from your VM. You can this by either...

- a. clicking on the TestExecute icon in the Task Bar and selecting 'Run'
- or
- b. pressing 'shift' and "F3"

After the run has completed TestExecute may try to open the log file in your default browser. This may be a .mht file which is only supported in IE. If you have IE on your VM you should see something like this...



If you don't have IE on your machine you can click on the TestExecute icon in the Task Bar and select 'Export Result'



We'll look at creating logs and distributing logs in more detail in a bit. For now though, we should be in a position where...

1. you can develop tests on your local desktop/laptop
2. you can execute tests on your virtual machine

In the next section we'll look at how we can trigger these tests from our local Desktop/Laptop machine and then let them run remotely on our virtual machine. We'll be able to develop on one machine, run and test your tests on another machine.

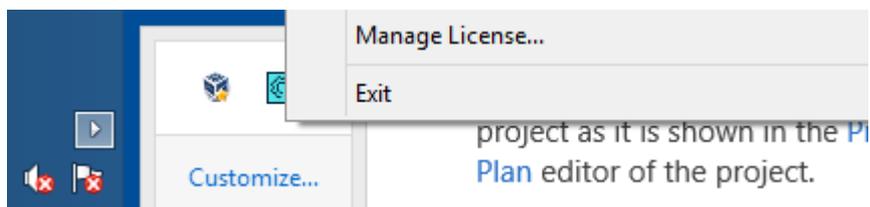
Stage 5 – Remotely Triggering Test Execution

There are a number of ways to trigger remote execution. You could use tools like PsExec.exe or Jenkins. The simplest way though is to use the inbuilt Test Runner REST API that comes with TestComplete and TestExecute. We'll set things up so that we can make a simple command line REST API request from our local Desktop/Laptop machine, to our VM where we'll start the test execution.

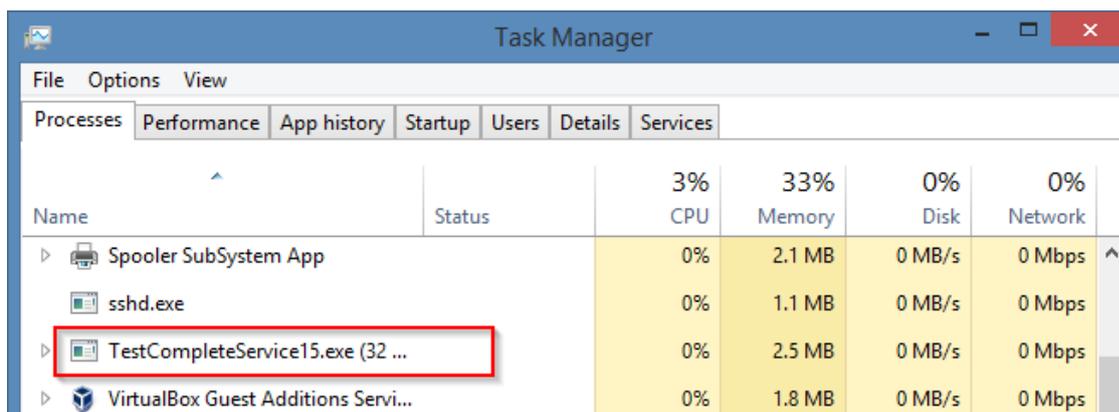
It's recommended that you only use this on internal networks. Opening up Ports that allow execution in this manner can compromise your computers security. With our VM setup running on Virtual Box we'll be okay but you do need to be aware of the important information set out here...

<https://support.smartbear.com/testcomplete/docs/reference/apis/test-runner.html>

First on the VM, from the task bar close TestExecute by clicking on "Exit".



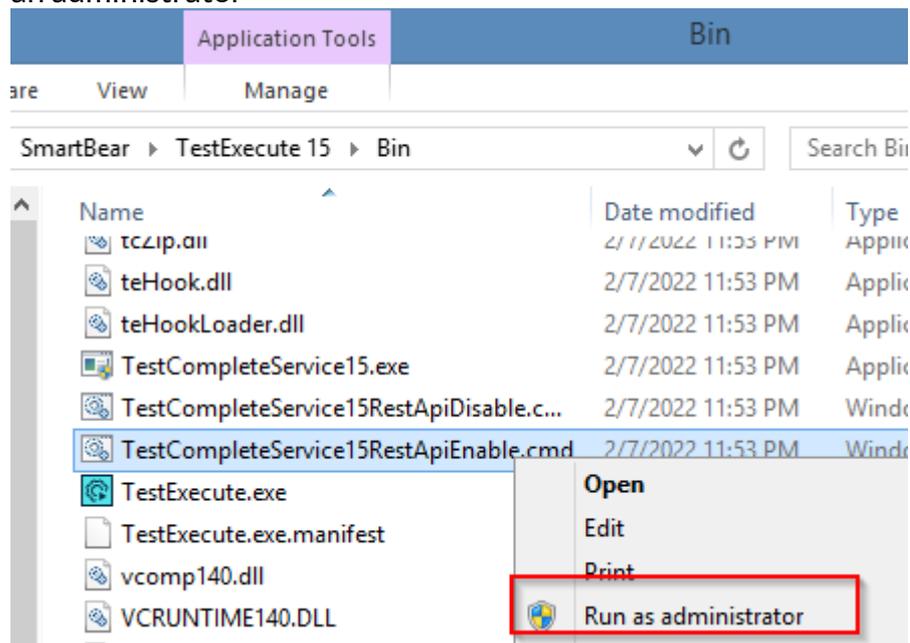
Then again on the VM start 'Task Manager' and make sure that TestCompleteService15.exe is running. This should have been installed and started when you installed TestExecute.



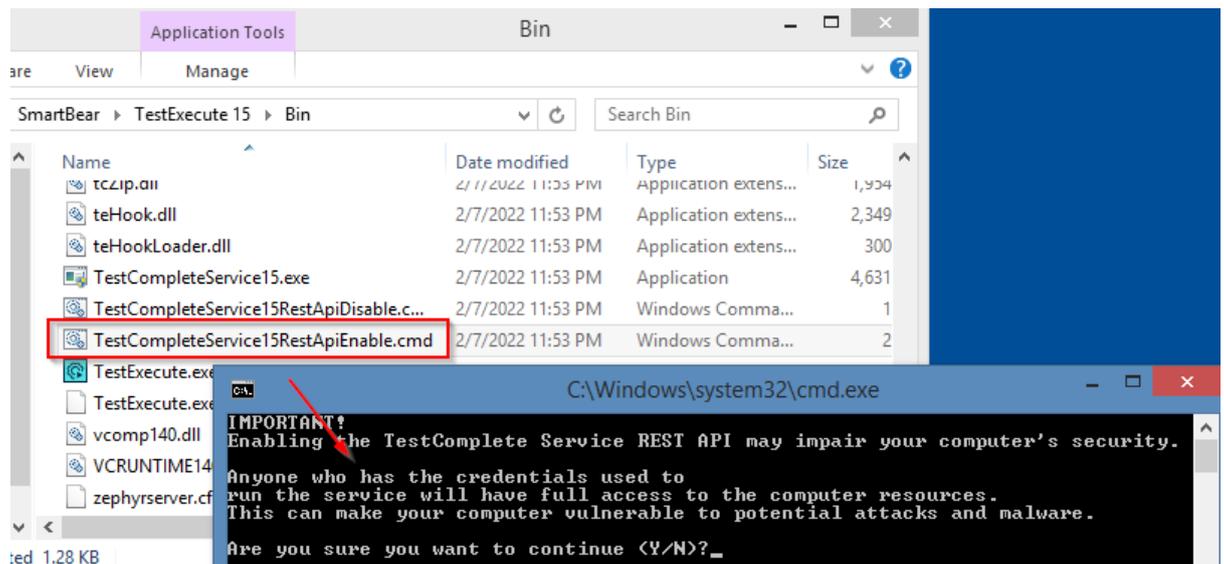
Next on the VM we need to run this batch file that will enable and turn on the api.

1. Open an explorer window and navigate to this directory:
C:\Program Files (x86)\SmartBear\TestExecute 15\Bin

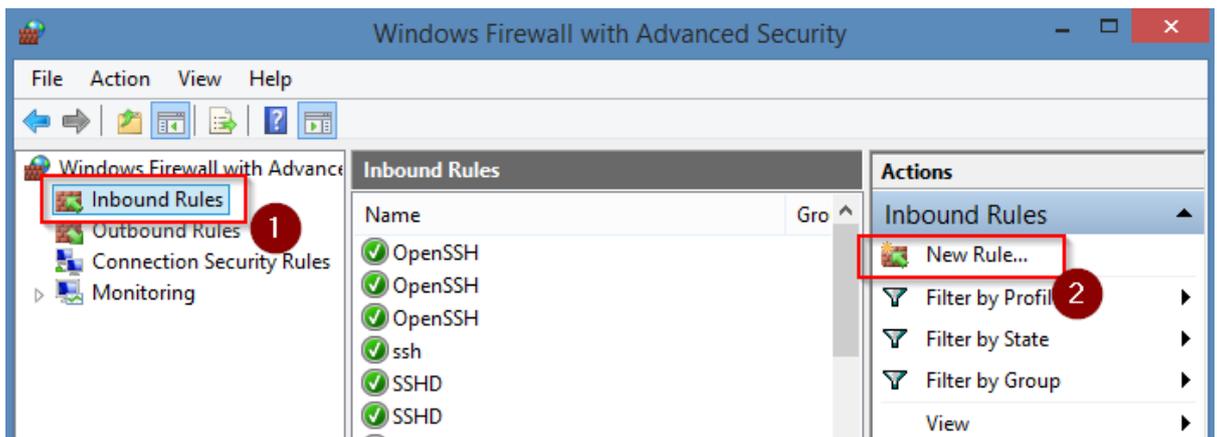
- Run (right click) the “TestCompleteService15RestApiEnable.cmd” command as an administrator



- Type “Y” at the prompt and accept the default port of 1880



4. Change the firewall settings on the VM to allow the Api calls



5. click on Inbound rules and add a New Rule with the settings below:

Rule Type: Port
Protocol and Ports: TCP and specific port 1880
Action: Allow the connection
Profile: Domain & Private & Public
Name: TestComplete REST

6. Allow interactive sessions by configuring the following group policies

Run "gpedit.msc"
Local Computer Policy > Computer Configuration > Administrative
Templates > All Settings
Disable the following policies:

- Always prompt client for password upon connection
- Prompt for credentials on the client computer

Once this is complete, from your Local Desktop/Laptop machine you should be able to access the Swagger service definition from your browser. We'll test this out next...

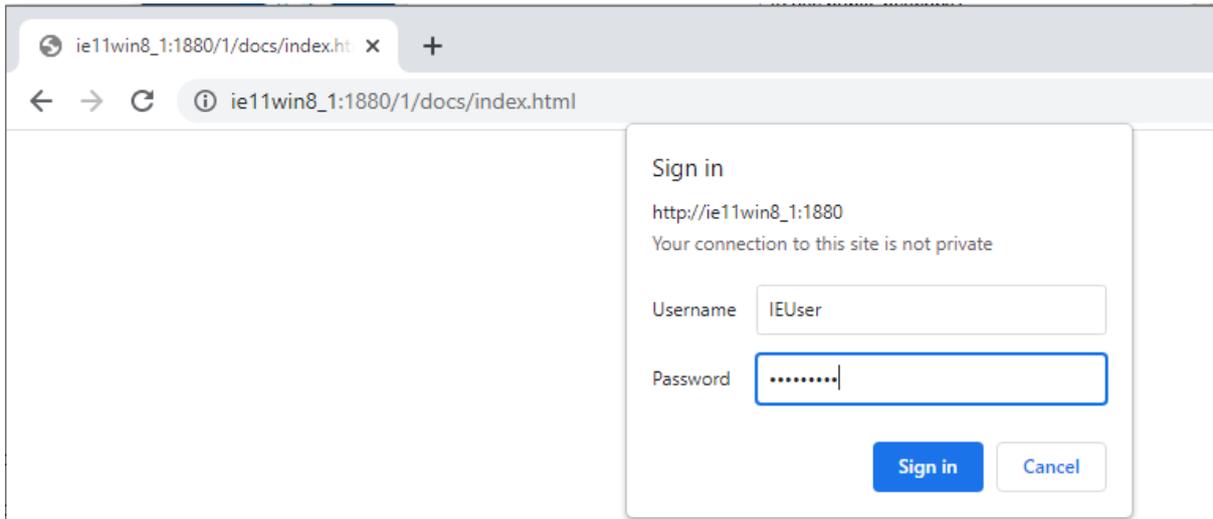
7. On your Local machine open a browser like Chrome and navigate to this url (you may find you have to do this with an IP address in place of the host name we've provided in this example).

http://ie11win8_1:1880/1/docs/index.html
(you may need to change your host name here if it's not the default ie11win8_1)

Then Login with the user name and password below

User: IEUser

Password: PasswOrd!



From here we can test our REST Api configuration by registering our existing Project Suite and making sure we can start TestExecute.

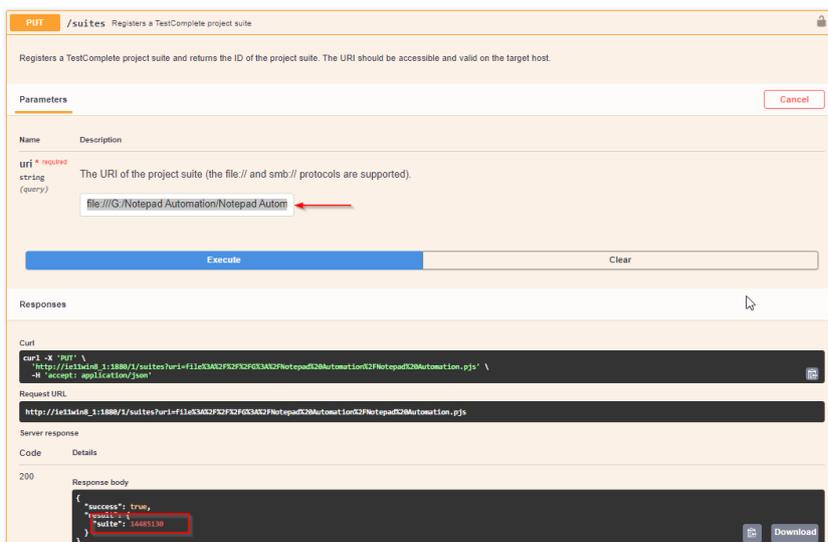
8. Register the project by going to this URL for the swagger UI that we use to register the Project Suite

http://ie11win8_1:1880/1/docs/index.html#/default/putProjectSuite
(you may need to change your host name here if it's not the default ie11win8_1)

9. Enter the following URI value for the project suites location on the VMs file system

`file:///G:/Notepad Automation/Notepad Automation.pjs`

10. Click execute and check you get back the suite id. We'll need this suite id to actually trigger the test

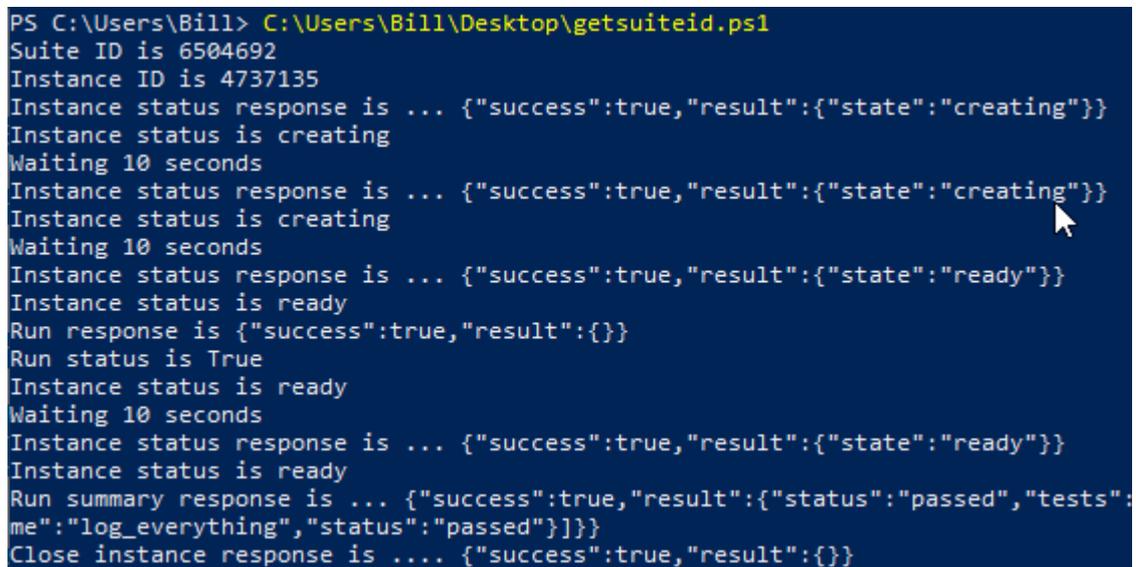


Now we know that the remote execution connection is working we can create a quick powershell script to run our tests on demand. Copy the script (below in the appendix) to create a powershell script that will trigger your test runs on the remote machine.

11. Open notepad, copy in the script and save the file as "runMyTests.ps1"
12. Edit and update the \$vm_host_name and \$vm_host_port values if they differ from the default
13. You may need to set your powershell execution policy by executing the following command in a powershell session

```
Set-ExecutionPolicy -ExecutionPolicy Unrestricted
```

14. Run the script and check the output



```
PS C:\Users\Bill> C:\Users\Bill\Desktop\getsuiteid.ps1
Suite ID is 6504692
Instance ID is 4737135
Instance status response is ... {"success":true,"result":{"state":"creating"}}
Instance status is creating
Waiting 10 seconds
Instance status response is ... {"success":true,"result":{"state":"creating"}}
Instance status is creating
Waiting 10 seconds
Instance status response is ... {"success":true,"result":{"state":"ready"}}
Instance status is ready
Run response is {"success":true,"result":{}}
Run status is True
Instance status is ready
Waiting 10 seconds
Instance status response is ... {"success":true,"result":{"state":"ready"}}
Instance status is ready
Run summary response is ... {"success":true,"result":{"status":"passed","tests":
me":"log_everything","status":"passed"}}}
Close instance response is ... {"success":true,"result":{}}
```

15. You should see the script go through the process of...

- i. registering the project suite
- ii. starting TestExecute and registering the instance
- iii. checking the status prior to execution
- iv. running your script
- v. checking status as it waits for completion
- vi. getting the execution summary response
- vii. closing the instance of TestExecute

All of this should result in the VM connection logging out whilst the test run executes in a separate windows session.

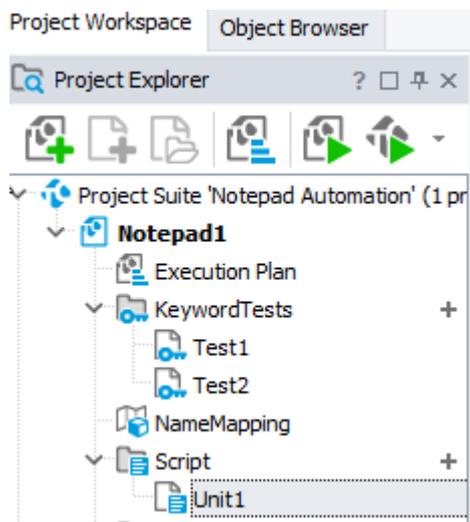
All we need to do now is sort out the logging so that we can capture the test result log on our local machine.

Stage 6 – Logging the Test Results

At this point we have a VM setup. We've created a file share which means we can develop tests on one machine and run them on the VM. We've configured the REST Api so that we have the ability to trigger executions on the VM remotely. We've developed a PowerShell script that allows us to trigger the test run on the VM with the simple execution of this script.

All we need now is to find a way to log the test results in a useful format and pull them back to the local machine where we can see them. We can do that by enhancing our original TestComplete project and making sure the log file is saved from the VM to the file share we've created.

1. Working on your local Desktop/Laptop with TestComplete, edit Unit1 in the Script section of your project

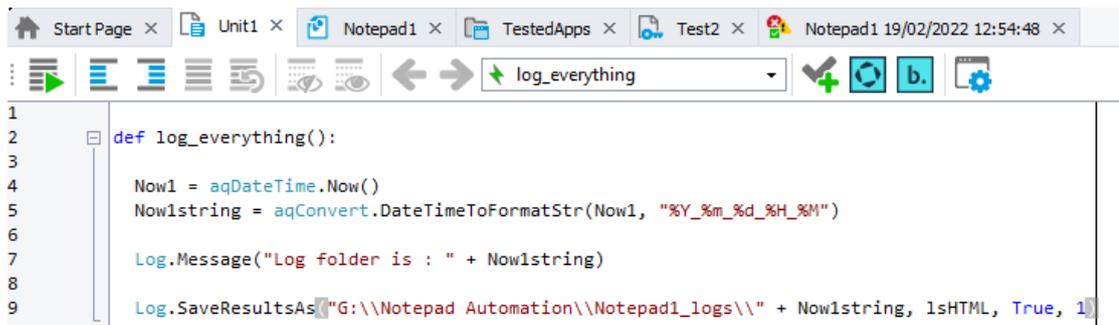


2. In your empty script unit, 'Unit1' add the following python code

```
def log_everything():  
  
    Now1 = aqDateTime.Now()  
    Now1string = aqConvert.DateTimeToFormatStr(Now1,  
"%Y_%m_%d_%H_%M")  
  
    Log.Message("Log folder is : " + Now1string)  
  
    Log.SaveResultsAs("G:\\Notepad Automation\\Notepad1_logs\\"  
+ Now1string, lsHTML, True, 1)
```

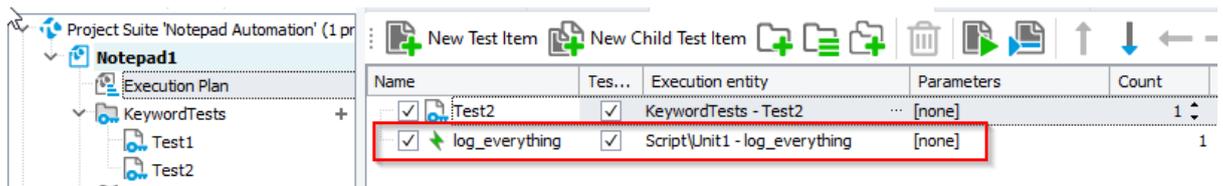
We're assuming here that your project is set to 'Python'. If you created your project with a different language you may need to adjust this code slightly.

3. In your script you may have to adjust the path in the Log.SaveResultAs method to your specific shared directory. If you've followed everything as listed so far then you should be okay with this directory. At this point your script should look something like this in TestComplete.



```
1 def log_everything():
2
3
4     Now1 = aqDateTime.Now()
5     Now1string = aqConvert.DateTimeToFormatStr(Now1, "%Y_%m_%d_%H_%M")
6
7     Log.Message("Log folder is : " + Now1string)
8
9     Log.SaveResultsAs("G:\\Notepad Automation\\Notepad1_logs\\" + Now1string, 1sHTML, True, 1)
```

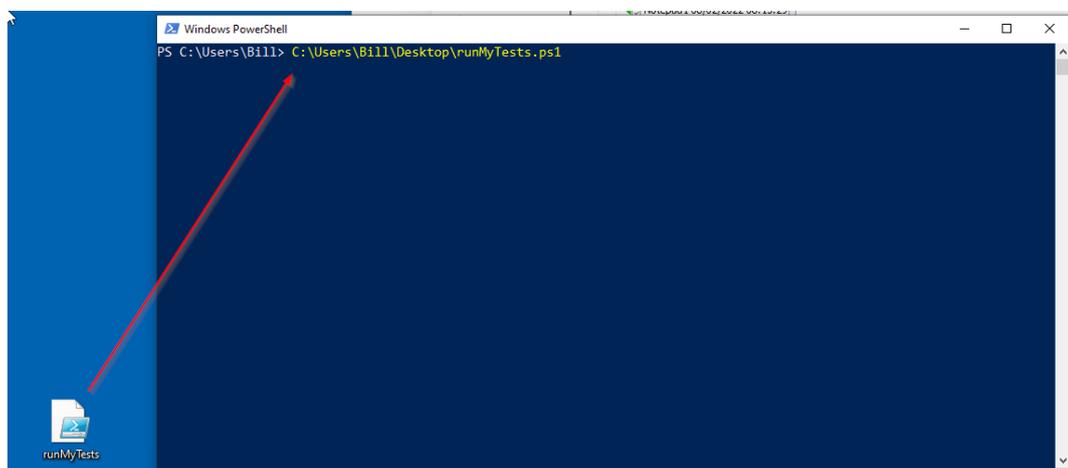
4. On your Execution plan add your new log_everything() method.



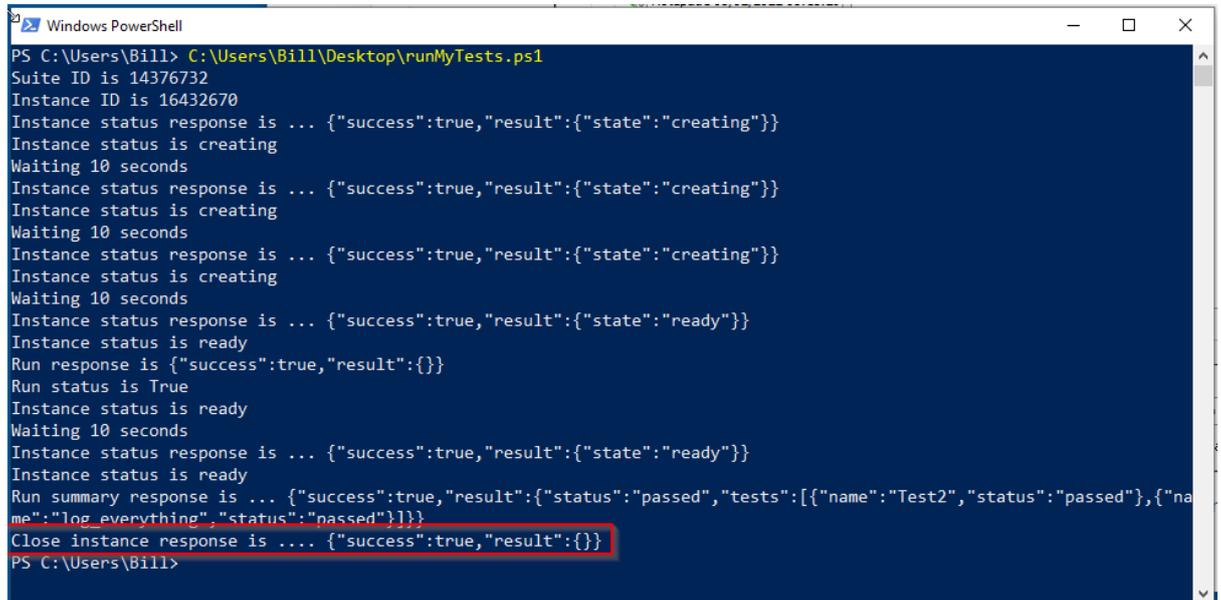
5. Save your project.

At this point you're saving on the file share and the changes made will be available to TestExecute on your VM immediately. All we need to do to check this works is run our runMyTest.ps1 PowerShell script

6. On your desktop, or wherever your runMyTest.ps1 script is located, run your PowerShell script

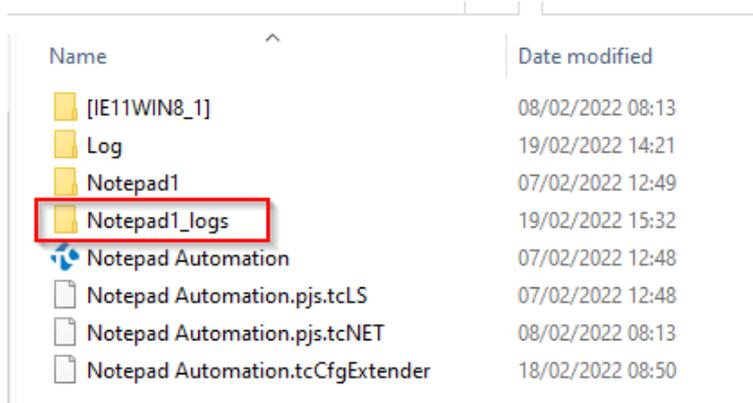


7. Check that the script runs successfully

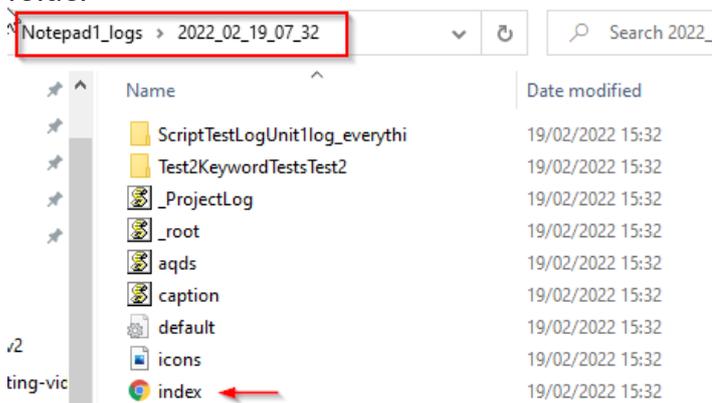


```
Windows PowerShell
PS C:\Users\Bill> C:\Users\Bill\Desktop\runMyTests.ps1
Suite ID is 14376732
Instance ID is 16432670
Instance status response is ... {"success":true,"result":{"state":"creating"}}
Instance status is creating
Waiting 10 seconds
Instance status response is ... {"success":true,"result":{"state":"creating"}}
Instance status is creating
Waiting 10 seconds
Instance status response is ... {"success":true,"result":{"state":"creating"}}
Instance status is creating
Waiting 10 seconds
Instance status response is ... {"success":true,"result":{"state":"ready"}}
Instance status is ready
Run response is {"success":true,"result":{}}
Run status is True
Instance status is ready
Waiting 10 seconds
Instance status response is ... {"success":true,"result":{"state":"ready"}}
Instance status is ready
Run summary response is ... {"success":true,"result":{"status":"passed","tests":[{"name":"Test2","status":"passed"},{"name":"log_everything","status":"passed"}]}}
Close instance response is ... {"success":true,"result":{}}
PS C:\Users\Bill>
```

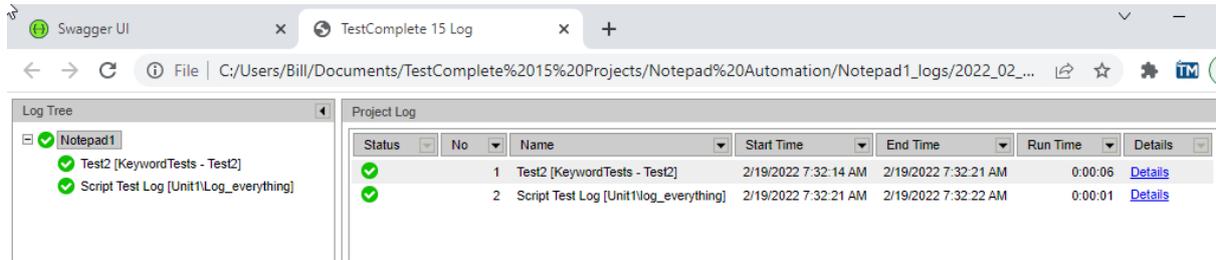
8. On your local machine, your file share, should have a new logs directory 'Notepad1_logs'



9. In this folder find the sub directory with the date / time file name and open the folder



10. In this folder you'll find the index.html file which you can open with your browser. You should see a results page like this...



Your test runs on your VM are now easily accessible on your local Desktop/Laptop machine

This last stage demonstrates perfectly how you can update your automation scripts on your local machine, run the scripts on your VM and then see the results in a browser on your local machine. You now have a very efficient way to develop, test and run your automated tests.

Appendix A – PowerShell Script To Execute Tests Remotely

```
# VM Host name
$vm_host_name = "iellwin8_1"
$vm_host_port = "1880"

# Register the TestComplete project suite
$suiteid_response = cmd /c curl -s -X "PUT"
"http://$vm_host_name:$vm_host_port/1/suites?uri=file%3A%2F%2F%2FG%3A%2FNotepad%20Automation%2FNotepad%20Automation.pjs" -H "accept: application/json" -u
"IEUser:Passw0rd!"
$suiteid = ($suiteid_response | Out-String | ConvertFrom-Json) | Where-Object
result | ForEach-Object {"${($_.result.suite)}"}
echo "Suite ID is $suiteid"

$instanceid_response = cmd /c curl -s -X "POST"
"http://$vm_host_name:$vm_host_port/1/suites/$suiteid/create_instance?screen_wi
dth=1280%26screen_height=1024%26color_depth=32" -H "accept: application/json" -u
"IEUser:Passw0rd!"
$instanceid = ($instanceid_response | Out-String | ConvertFrom-Json) | Where-Object
result | ForEach-Object {"${($_.result.instance)}"}
echo "Instance ID is $instanceid"

$instancestatus_response = cmd /c curl -s -X "GET"
"http://$vm_host_name:$vm_host_port/1/instances/$instanceid" -H "accept:
application/json" -u "IEUser:Passw0rd!"
echo "Instance status response is ... $instancestatus_response"

$instancestatus = ($instancestatus_response | Out-String | ConvertFrom-Json) |
Where-Object result | ForEach-Object {"${($_.result.state)}"}
echo "Instance status is $instancestatus"

$a = 1
```

```

Do
{
echo "Waiting 10 seconds"
Start-Sleep -Second 10

$instancestatus_response = cmd /c curl -s -X "GET"
"http://${vm_host_name}:${vm_host_port}/1/instances/${instanceid}" -H "accept:
application/json" -u "IEUser:Passw0rd!"
echo "Instance status response is ... $instancestatus_response"
$instancestatus = ($instancestatus_response | Out-String | ConvertFrom-Json) |
Where-Object result | ForEach-Object {"${$_.result.state}"}
echo "Instance status is $instancestatus"

$a++
if ($a -eq 7)
{
echo "Giving up"
Exit
}

} While ($instancestatus -ne "ready")

$run_response = cmd /c curl -s -X "POST"
"http://${vm_host_name}:${vm_host_port}/1/instances/${instanceid}/run?project=Notepad
1" -H "accept: application/json" -u "IEUser:Passw0rd!"
echo "Run response is $run_response"
$run_status = ($run_response | Out-String | ConvertFrom-Json) | Where-Object result
| ForEach-Object {"${$_.success}"}
echo "Run status is $run_status"

$instancestatus = ($instancestatus_response | Out-String | ConvertFrom-Json) |
Where-Object result | ForEach-Object {"${$_.result.state}"}
echo "Instance status is $instancestatus"

$a = 1
Do
{
echo "Waiting 10 seconds"
Start-Sleep -Second 10

$instancestatus_response = cmd /c curl -s -X "GET"
"http://${vm_host_name}:${vm_host_port}/1/instances/${instanceid}" -H "accept:
application/json" -u "IEUser:Passw0rd!"
echo "Instance status response is ... $instancestatus_response"
$instancestatus = ($instancestatus_response | Out-String | ConvertFrom-Json) |
Where-Object result | ForEach-Object {"${$_.result.state}"}
echo "Instance status is $instancestatus"

$a++
if ($a -eq 7)
{
echo "Giving up"
Exit
}

} While ($instancestatus -ne "ready")

$runsummary_response = cmd /c curl -s -X "GET"
"http://${vm_host_name}:${vm_host_port}/1/instances/${instanceid}/summary" -H
"accept: application/json" -u "IEUser:Passw0rd!"
echo "Run summary response is ... $runsummary_response"

$closeinstance_response = cmd /c curl -s -X "DELETE"
"http://${vm_host_name}:${vm_host_port}/1/instances/${instanceid}" -H "accept:
application/json" -u "IEUser:Passw0rd!"
echo "Close instance response is .... $closeinstance_response "

```