**Vagrant and Vector**

Azure Provider

Draft B

22nd October 2015

****

**Actian Europe Limited**

16/17 Pavilion Business Park

Royds Hall Road

Leeds

LS12 6AJ

www.actian.com

© Actian Europe Limited, 2015

**Contents**

1 Introduction 3

2 Azure 4

2.1 Gather Essential Information 4

2.2 Create Certificates 7

3 Vagrant 10

3.1 Prerequisites for Azure 10

3.2 Vagrantfile Changes 10

3.3 Gotchas 12

4 Azure Linux VM Access 15

4.1 Putty Private Key 15

4.2 Putty Access 16

5 External Access 19

5.1 Linux Ports 19

5.2 Azure Ports 20

5.3 Ingres Net Connection 23

5.4 ODBC Connection 24

5.5 JDBC Connection 25

# Introduction

This document is intended for those less familiar with Vagrant and Azure to aid in getting Vagrant to create a VM using in the Microsoft Azure cloud service.

Other approaches may be possible, but the one documented here has been developed to work around current limitations in Vagrant’s support for Azure. Better alternative approaches may be possible and any valid contribution is gladly accepted.

This document is necessary because, whereas creating a local VM using Vagrant against say the Oracle Virtual Box (VB) provider is fairly straight forward, this is not the case for Azure with the current Vagrant release (1.7.4). There are quite a few ‘nuances’ with Azure as well as components, such as Chef, that just don’t seem to work properly in the Azure environment and hence need to be worked around.

Additionally, unlike VB Azure requires account-specific credentials to be entered before it will work, which are specific to you and your Azure Account Subscription.

This document assumes that you are running Vagrant pre-installed on a Windows machine.

Currently, a Microsoft is offering a free trial of Azure which may be used to try out this approach and that platform.

# Azure

First thing to do is sign up for Azure. At the time of writing a 30 day free trial with $200/£125 credit is available. This should be ample to get you off the ground as usage can be kept very low by simply destroying your VMs at the end of the day.

Next you need the image name for the box, as follows.

## Gather Essential Information

For Azure it does not seem possible to get a list of box image names from the ‘Management Interface’ so you have to proceed as follows:

1. Download Azure PowerShell

<http://go.microsoft.com/fwlink/p/?LinkId=320376>

Once you have installed it run up the PowerShell:



1. Configure Access to Azure

In the PowerShell download the settings files for your subscriptions, e.g. the free trial. In the PowerShell enter:

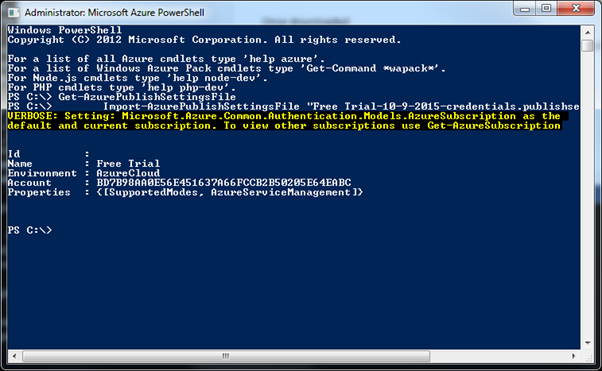
Get-AzurePublishSettingsFile

If you are not currently logged on to your Azure account you will be prompted for your user and password.

In your downloads folder you will now have a file named something like ‘Free Trial-10-13-2015-credentials.publishsettings’. Copy this to C: which is the default directory for the PowerShell

Now import these settings by typing the following in the PowerShell:

Import-AzurePublishSettingsFile "Free Trial-10-9-2015-credentials.publishsettings"



1. Subscription ID

Note the ‘Id’ that is displayed (Author’s blanked out). This is your Free Trial Subscription ID and you will need this later when updating your Vagrant file to use the Azure provider.

1. OS Name in Azure

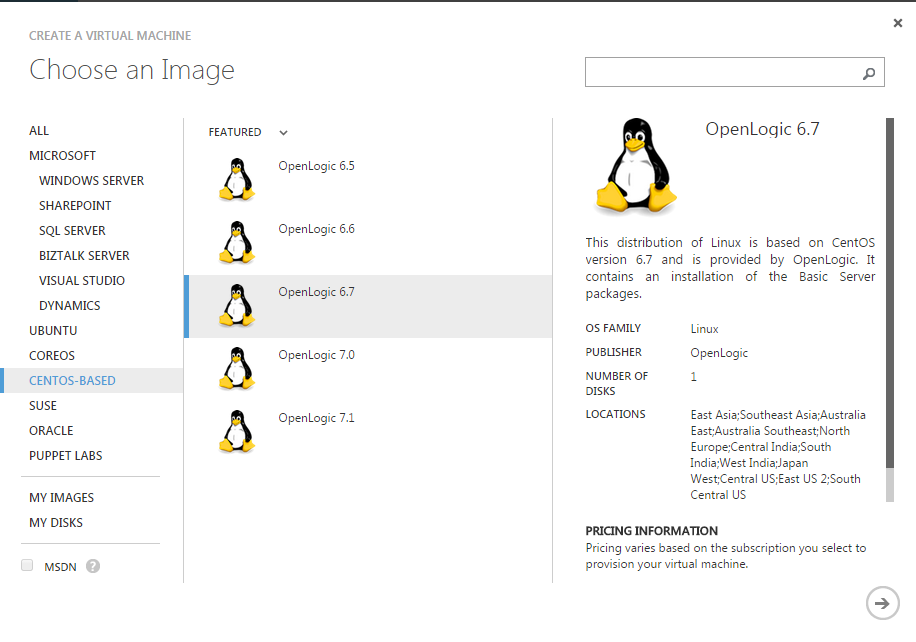
Before you can get the actual OS image name we use in the Vagrant file you need to get the OS name as Azure sees it.

Logon to your Microsoft Azure account.

Select ‘Virtual Machines’ 🡪 ‘CREATE A VIRTUAL MACHINE’

Select ‘FROM GALLERY’

Select the appropriate image. In this example ‘CentOS 6.7’ which is published and maintained by OpenLogic as ‘OpenLogic 6.7’.

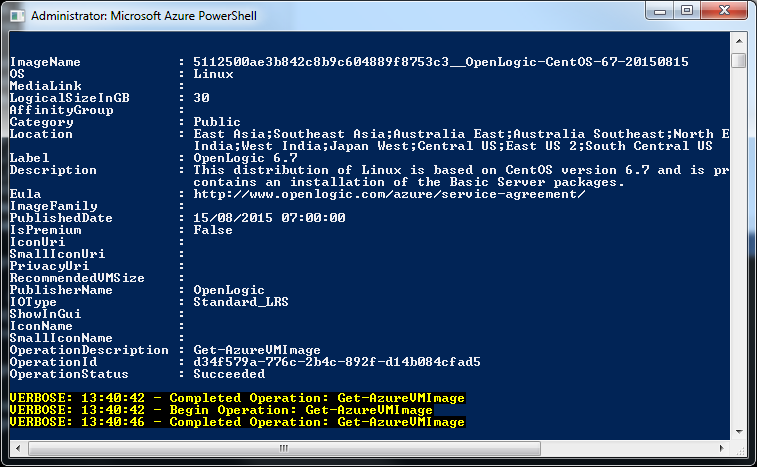


Note the OS name e.g. ‘OpenLogic 6.7’

1. OS Image Name

Now return the PowerShell and run the following command with the OS name substituted:

Get-AzureVMImage | where-object { $\_.Label -like "OS Name" }



Note the long image name at the top as you will need this later for your Vagrant file.

## Create Certificates

In order for Vagrant to have access to your Azure account you need to create a certificate to upload to your Azure Account.

Run these command from a DOS prompt in the directory where the Vagrantfile was located e.g. C:\Azure\_Example.

The certificates were created by using openssl on Windows:

openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout azurevagrant.pem -out azurevagrant.pem

Note

‘Openssl’ needs to be installed before this point; commonly it is installed as part of other software such as the GitHub client. However, if you don’t have it the following command from your Azure PowerShell will install it:

choco install openssl.light

You’ll need to answer a few questions here but most can be left blank. For the purposes of this examples the only ones completed are highlighted.

Generating a 2048 bit RSA private key

........................+++

............+++

writing new private key to 'azurevagrant.pem'

-----

You are about to be asked to enter information that will be incorporated into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

-----

Country Name (2 letter code) [AU]:**UK**

State or Province Name (full name) [Some-State]:

Locality Name (eg, city) []:

Organization Name (eg, company) [Internet Widgits Pty Ltd]:**Actian**

Organizational Unit Name (eg, section) []:

Common Name (e.g. server FQDN or YOUR name) []:

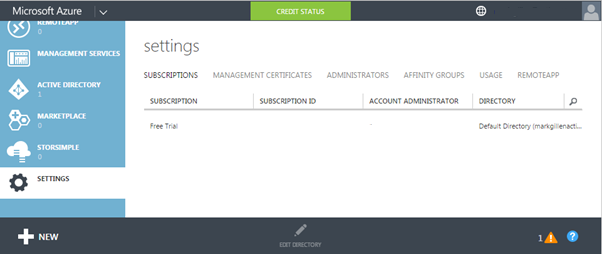
Email Address []:**my.name@actian.com**

Then run this command to create a certificate:

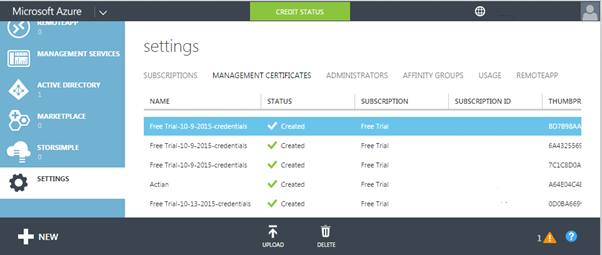
openssl x509 -inform pem -in azurevagrant.pem -outform der -out azurevagrant.cer

The .cer file is the one you are going to import into your Azure account and the .pem and will be used in conjunction with it in the Vagrant provider configuration. Additionally, later you will use the .pem file to generate a private key file to give Putty access to the Azure CentOS VM, if that is your preferred terminal access method.

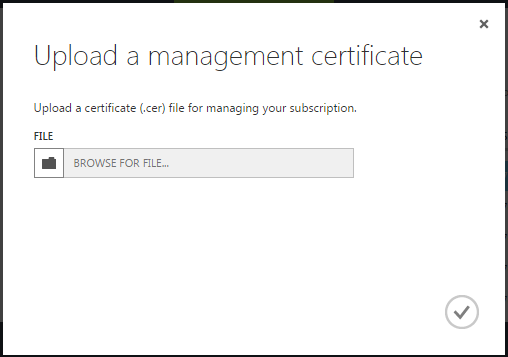
In your Microsoft Azure account select ‘SETTINGS’ at the bottom of the left scrollable panel:



Select ‘MANAGEMENT CERTIFICATES’



Select the ‘UPLOAD’ icon at the bottom



Select the .cer file you created earlier and select the tick to upload.

# Vagrant

## Prerequisites for Azure

You should already have installed vagrant. There are a couple of additional essential installs to create a VM against Azure:

1. Install the Vagrant Azure plugin:

vagrant plugin install vagrant-azure

1. Install a dummy Vagrant Azure Box:

vagrant box add azure https://github.com/msopentech/vagrant-azure/raw/master/dummy.box

There was reference to the need for a plug-in to manage the version download but it seems to work without this step.

vagrant plugin install vagrant-omnibus

## Vagrantfile Changes

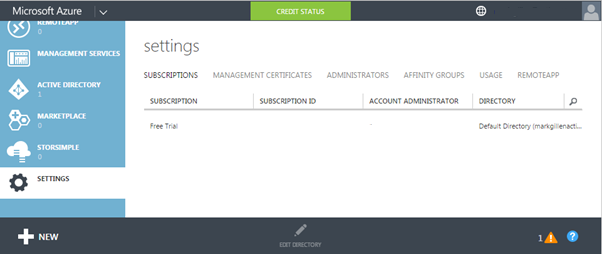
First thing to know is that unlike VB but like all other Cloud providers, you can’t have a Vagrant file for the Azure provider that works for everyone because there are details specific to you and you only, which are:

1. Your Azure Subscription ID;
2. Your certificate:

* The .pem file.

You obtained these earlier in Sections 2.1 and 2.2. It should be noted however that most of section 2.1 is related to getting the VM image name.

If you just want to stick with CentOS 6.7 then Section 2.1 is not necessary and the essential Subscription ID can be obtained from the subscription details in your Azure Account. It is in your Microsoft Azure account select ‘SETTINGS’ at the bottom of the left scrollable panel (Author’s blanked out):



In your chosen folder e.g. c:\Azure\_Example copy in the files from Github for the Vagrant package to which this document is associated – i.e. from <https://github.com/ActianCorp/Vagrant-Vector-Install>

In the ‘vagrantfile’ the settings that you will need to change related to configuring an Azure provider are documented in the code snippet below though the full file is available in Github.

A colour scheme has been used to denote the key settings to change:

**RED** – Settings specific to your Azure Account and must be changed.

**AMBER** – Settings that can remain unchanged if you choose to follow the author’s naming standard when completing Section 2.2.

**GREEN** – Settings that affect the naming of Azure components created which you may wish to reflect your own naming standards.

config.vm.provider :azure do |azure, override|

override.vm.box = 'azure'

override.ssh.private\_key\_path = **'azurevagrant.pem'**

override.ssh.pty = true

override.vm.boot\_timeout = 1500

# Mandatory Settings

azure.mgmt\_certificate = **'azurevagrant.pem'**

azure.mgmt\_endpoint = 'https://management.core.windows.net'

azure.subscription\_id = **'########-####-####-####-############'**

azure.vm\_image = '5112500ae3b842c8b9c604889f8753c3\_\_OpenLogic-CentOS-67-20150815'

azure.vm\_name = **'VectorEvaluationVM'**

azure.ssh\_private\_key\_file = **'azurevagrant.pem'**

# Optional Settings

azure.cloud\_service\_name = **'VectorEvaluationVM'**

azure.vm\_location = 'North Europe'

azure.ssh\_port = '22'

# Need larger than default Standard A1 to run Vector

azure.vm\_size = 'Basic\_A2'

end

## Gotchas

1. ResourceNotFound

The following message is frequently displayed on vagrant up:

ResourceNotFound : The hosted service does not exist.

According to some reports this caused the VM creation to abort, but in practice it causes no problems and can’t be suppressed. According to the Vagrant documentation the cloud service is auto generated if you leave azure.cloud\_service\_name blank.

2. Time Out

The default timeout for vagrant when waiting for the VM to start is 360 seconds. This is just not sufficient for an Azure VM initially being created as this is a minimum of 5 minutes, although at times even 10 minutes is not long enough:

override.vm.boot\_timeout = 1500

Note – ‘override’ rather than config as this is an example from the dual provider Vagrantfile. This allows the default for the non-Azure providers to remain the same.

3. pty

The following setting seems to be mandatory to get things working on Azure:

override.ssh.pty = true

It was found that, despite the dire Vagrant documentation warning below, this setting had to be used otherwise the following error was found:

sorry, you must have a tty to run sudo

The SSH command responded with a non-zero exit status.

Vagrant assumes that this means the command failed.

The output for this command should be in the log above.

Please read the output to determine what went wrong.

Be aware, from Vagrant online documentation:

config.ssh.pty - If true, pty will be used for provisioning. Defaults to false.

This setting is an advanced feature that should not be enabled unless absolutely necessary. It breaks some other features of Vagrant, and is really only exposed for cases where it is absolutely necessary. If you can find a way to not use a pty, that is recommended instead.

4. Chef

For some as yet undiagnosed reason Chef will not auto install on an Azure CentOS VM. When you perform a “chef\_apply” in a Vagrantfile it checks and installs Chef if it not present. Whatever, method is employed by Vagrant appears not to be acceptable to the Azure VM and results in the message:

Vagrant attempted to execute the capability 'chef\_install' on the detect guest OS 'linux', but the guest doesn't support that capability.

This capability is required for your configuration of Vagrant. Please either reconfigure Vagrant to avoid this capability or fix the issue by creating the capability.

This problem was circumvented by pre-installing Chef from a shell provisioner as follows:

sudo su - -c 'curl -L https://www.opscode.com/chef/install.sh | bash'

Additionally Vagrant does not appear to be able to apply Chef Recipes using the expected:

config.vm.provision ‘chef\_apply’ do ….

This appears to work but the results in subsequent failures. As a result this was circumvented by running it via a shell after uploading the required Chef Recipe script:

config.vm.provision 'file', source: 'chef\_script.rb', destination: '/tmp/chef\_script.rb'

config.vm.provision 'shell', name: Run Chef Recipe', privileged: true, inline: <<-SHELL

sudo su - -c 'chef-apply /tmp/chef\_script.rb'

SHELL

# Azure Linux VM Access

For this example, Putty is used for Unix/Linux terminal access to Azure, though other approaches will no doubt also work.

## Putty Private Key

In order to access the Azure VM using Putty you will need a private key to match the certificate uploaded to your Azure account. You generated one, the .pem file, earlier. However, Putty does not understand this and requires a .ppk.

First create an rsa file using openssl used previously. At your DOS command prompt:

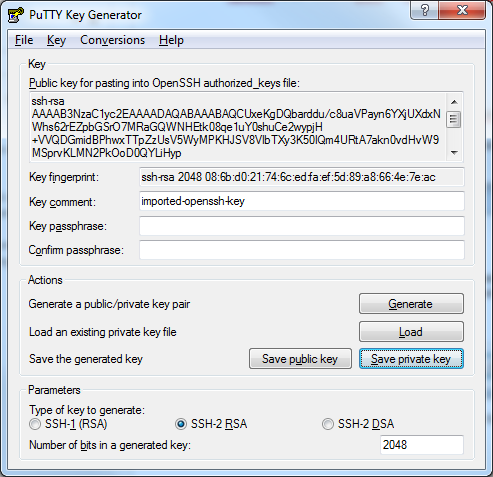
openssl rsa -in azurevagrant.pem -out azurevagrant\_rsa

Only problem is that Putty still does not understand this format and it needs to be converted. This is very easy to achieve because if you have Putty you should also have PuttyGen. Start PuttyGen:

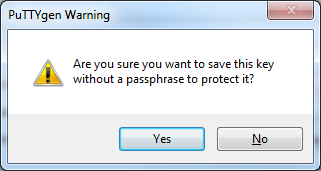


Select ‘Conversions’ -> ‘Import key’ from the menu bar.

From the file browser displayed select the file you created above.



Select the ‘Save private key’ button and respond ‘Yes’ as you don’t want a pass phrase.

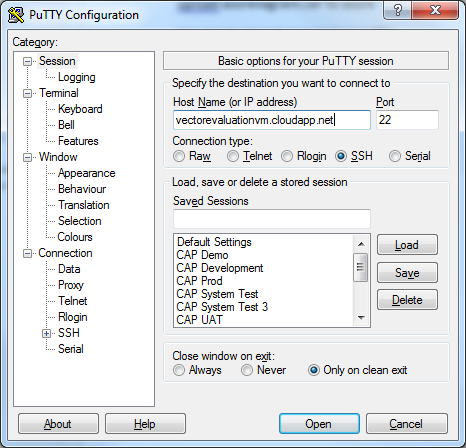


This will create you a .ppk file that Putty will understand. The private key obviously already matches the certificate we uploaded to the Azure account in Section 2.2.

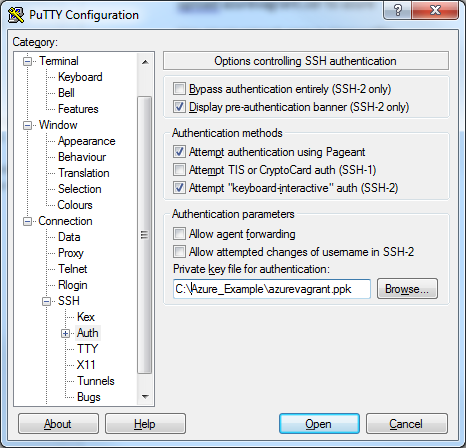
## Putty Access

Now that you have the private key, access is fairly straight forward.

Start Putty and set the hostname. You can get this from your Azure Account but this is nearly always your azure.vm\_name set in the Vagrant file + ‘cloudapp.net’ e.g. vectorevaluationvm.cloudapp.net.



Expand ‘SSH’ and select ‘Auth’ and ‘Browse’ for the .ppk file you created earlier:



Select ‘Open’ to connect to your running Azure VM.

Since the private key is associated with the ‘vagrant’ user this is what you should logon as then switch to actian e.g. sudo su – actian

# External Access

This section documents how to get access to Vector on Azure from tools such as Tableau. It covers the configuration of access via:

1. Ingres Net;
2. ODBC;
3. JDBC.

## Linux Ports

Firstly, your VM has to be configured to open the ports in the firewall on Linux that are required to communicate with Vector. These are:

1. The Management Server Discovery Port (DIS);
2. The Management Server Command Port (Mgmtsrvr);
3. The Communication Server Port (GCC);
4. The Data Access Server Port (DAS).

The DIS is fixed at 16902 but the others are calculated values based on the installation ID e.g. VH, VW etc. There is a formula for this but just look in the Vector error log $II\_SYSTEM/ingres/files/errlog.log of the installation as they are output on start-up

The Vagrant Vector installation uses ID VH and as a result the above resolve to 16902, 27712, 27719 and 44103 respectively.

To ensure that each of the of these ports is open you can open each or if you are simply in a testing situation and not concerned with security just close the firewall.

Open each port individually:

sudo su -

iptables -I INPUT -p tcp --dport 16902 --syn -j ACCEPT

iptables -I INPUT -p tcp --dport 27712 --syn -j ACCEPT

iptables -I INPUT -p tcp --dport 27719 --syn -j ACCEPT

iptables -I INPUT -p tcp --dport 44103 --syn -j ACCEPT

service iptables save

Output from above command:

iptables: Saving firewall rules to /etc/sysconfig/iptables:[ OK ]

Close firewall:

sudo su –

chkconfig iptables off

service iptables off

Output from above command:

iptables: Setting chains to policy ACCEPT: filter [ OK ]

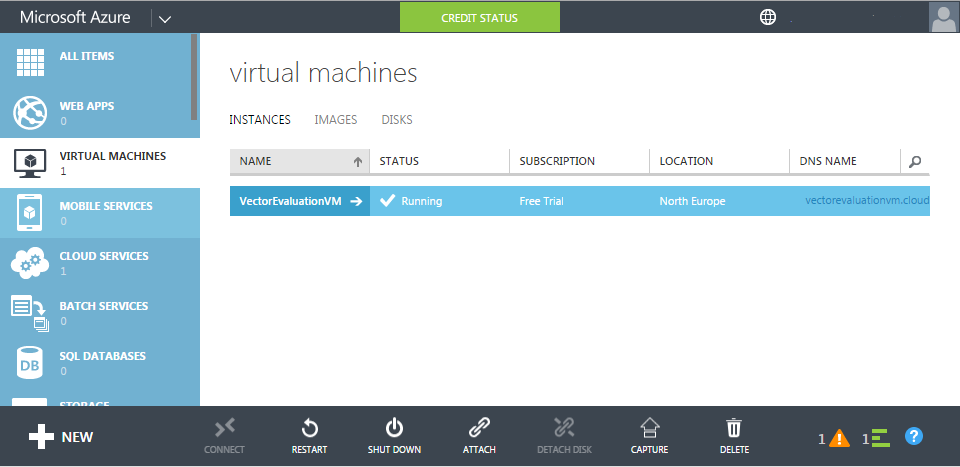
iptables: Flushing firewall rules: [ OK ]

iptables: Unloading modules: [ OK ]

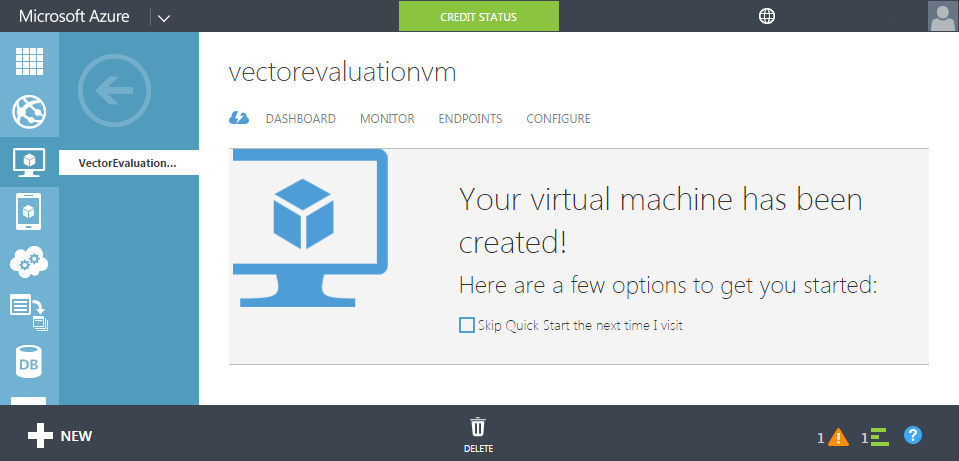
## Azure Ports

The next step is to create ‘endpoints’ to your VM using your Azure Account.

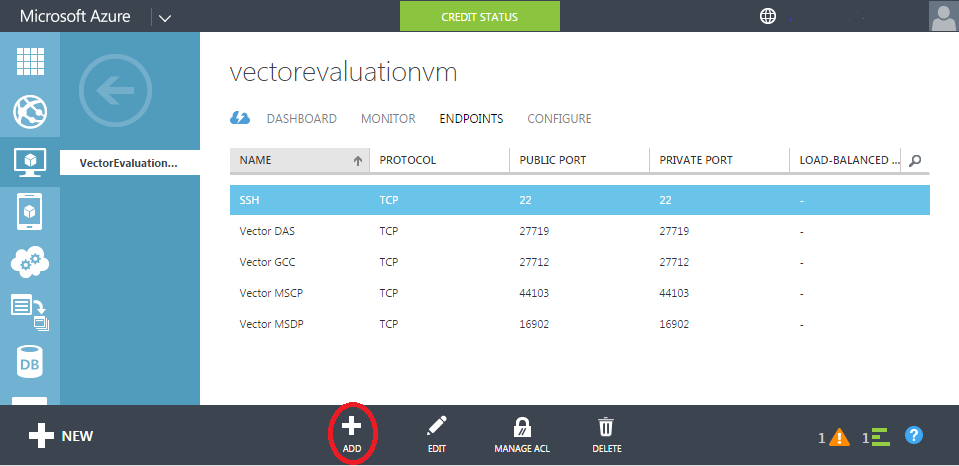
Login to your Azure Account and Select ‘VIRTUAL MACHINES’ from the left panel.



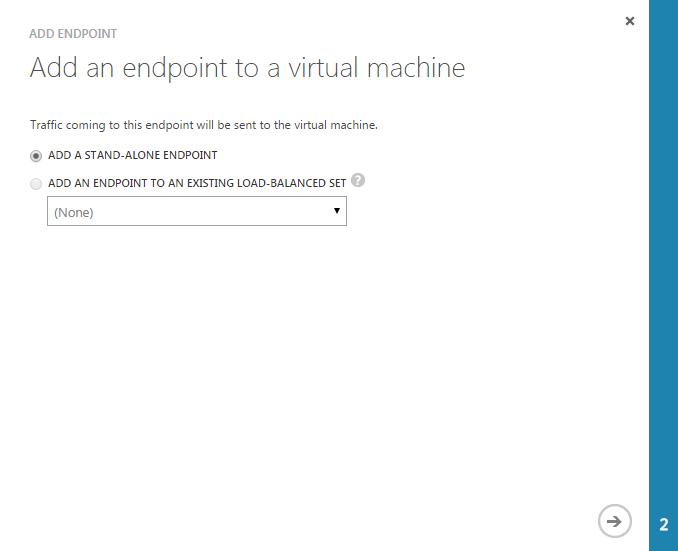
Select the VM created by Vagrant e.g. VectorEvaluationVM.

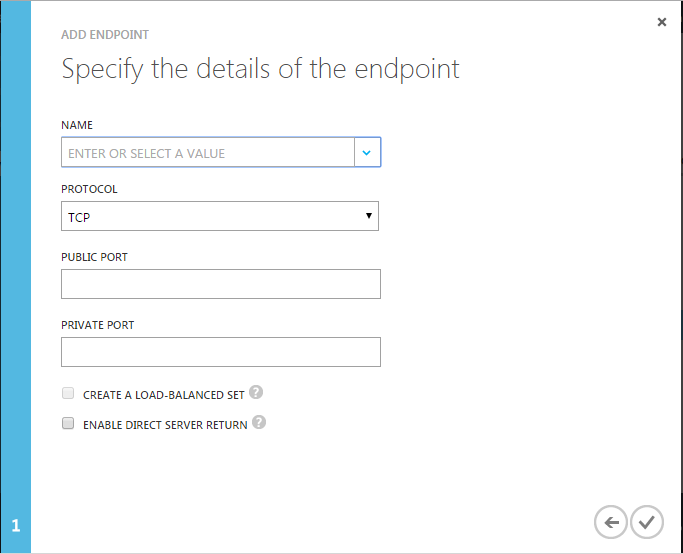


Select ‘ENDPOINTS’ the Add the 4 endpoints corresponding to the ports that need to be accessible using the ‘ADD’ button.



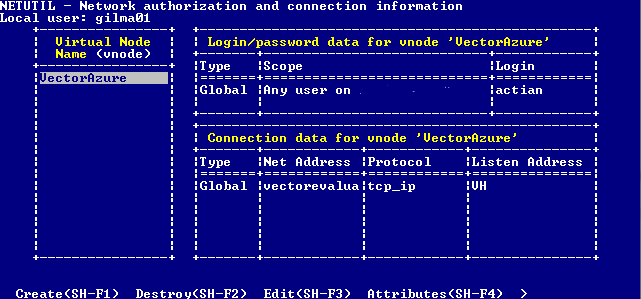
The entries are fairly self explanatory but always map the public and private ports to be the same.





## Ingres Net Connection

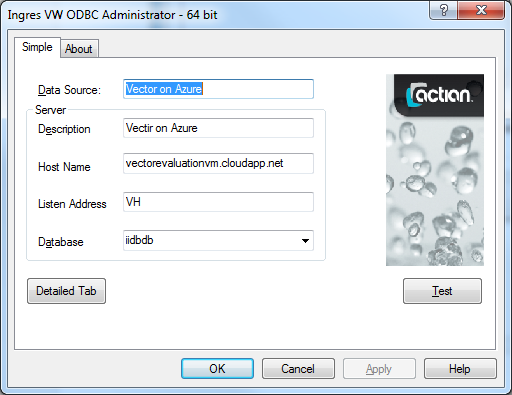
This connection relies on having an Ingres Client to create a NetUtil entry as follows:



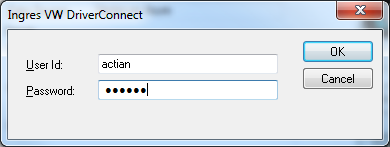
Although not visible, it is possible to use the full name of the Azure VM as the ‘Net Address’ e.g. vectorevaluationvm.clodapp.net

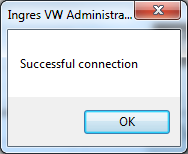
## ODBC Connection

To configure your Windows ODBC connection is little different from that for any other. You will however need the Vector ODBC driver from Actian. This is installed by default if your client (In the example in this document the Windows machine from which you ran Vagrant) has Vector, Ingres or the Ingres Client installed.



Once you have configured the connection test in the usual fashion:





## JDBC Connection

A JDBC connection is demonstrated by means of the Open source product ‘Squirrel’.

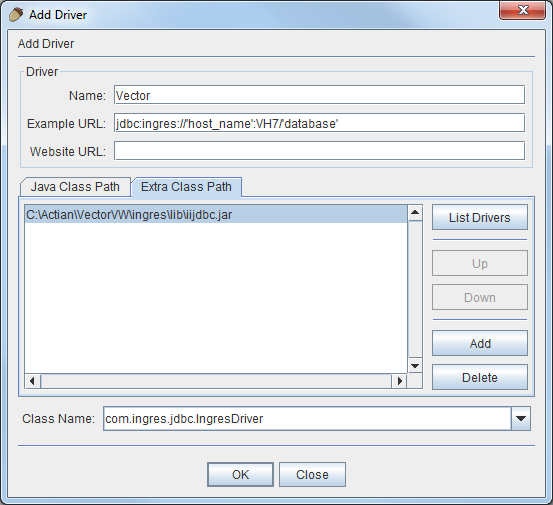
To make any JDBC connection the appropriate Vector JDBC driver JAR file is required and added to the CLASSPATH as demonstrated by the screen shot below. In this example the JAR file was used from a local Vector installation but it can be downloaded separately from esd.actian.com.

Configure a Vector driver with the following details.

File : iijdbc.jar

ClassName : com.ingres.jdbc.IngresDriver

Example URL : jdbc:ingres://'host\_name':VH7/'database'



Add an Alias to access your Azure VM Vector installation.

