**Quick Start Guide**

**Actian Vector on Amazon AWS**

Vagrant AWS Provider

Draft A

5th November 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 AWS 4

2.1 Access Keys 4

2.2 Port Access 9

2.3 Key Pairs - Certificate 11

2.4 Amazon Machine Image 13

3 Vagrant 15

3.1 Prerequisites for AWS 15

3.2 Vagrantfile Changes 15

3.3 Linux Provisioning User 16

3.4 Gotchas 18

4 AWS Linux Instance Access 20

5 External Access 21

5.1 Linux Ports 21

5.2 AWS Ports 22

5.3 Ingres Net Connection 23

5.4 ODBC Connection 24

5.5 JDBC Connection 25

# Introduction

This document is intended to show a quick way for a developer to get up and running with a live Vector installation running on Amazon AWS.

The guide is based on the use of a developer tool called Vagrant. It is intended for those unfamiliar with Vagrant and AWS as an aid in getting Vagrant to create a Linux machine using in the AWS cloud service, and then installing Actian Vector on that virtual machine (VM).

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

Vagrant is a developer-oriented tool that is designed to simplify and automate the process of defining, creating and configuring a virtual machine with a defined set of software packages.

This document is necessary because, whereas creating a local VM using Vagrant with the default Oracle Virtual Box (VB) provider is fairly straight forward, this is not the case for AWS since information needs to be gathered from and settings changed in your AWS Account to get it working. There are many superficial examples of a Vagrant file for AWS on the internet but these often skip over critical details of what you need to do in AWS - which is the more difficult aspect if one is unfamiliar with that environment.

This document assumes that you are running Vagrant pre-installed on a Windows machine. This open source tool can be downloaded and installed from [www.vagrantup.com](http://www.vagrantup.com).

Currently, Amazon are offering a free trial of AWS which may be used to try out this approach and that platform. However, the machines available in the trial are limited to 1Gb of memory which prevents the installation of Actian Vector (which requires a minimum of 3Gb RAM to install, but Actian recommends 8Gb of RAM per CPU core for more real-world workloads).

# AWS

First thing to do is sign up for Amazon AWS. At the time of writing a 12 month free trial is available allowing you access to the free tier. This in terms of computing power is free EC2 instance though these are limited to 1 CPU and 1 GB of memory.

The next step is to gather all the essential AWS information as well as making the appropriate setting changes to allow Vagrant to use AWS as a provider.

## Access Keys

For Vagrant to access AWS a set of access keys are required:

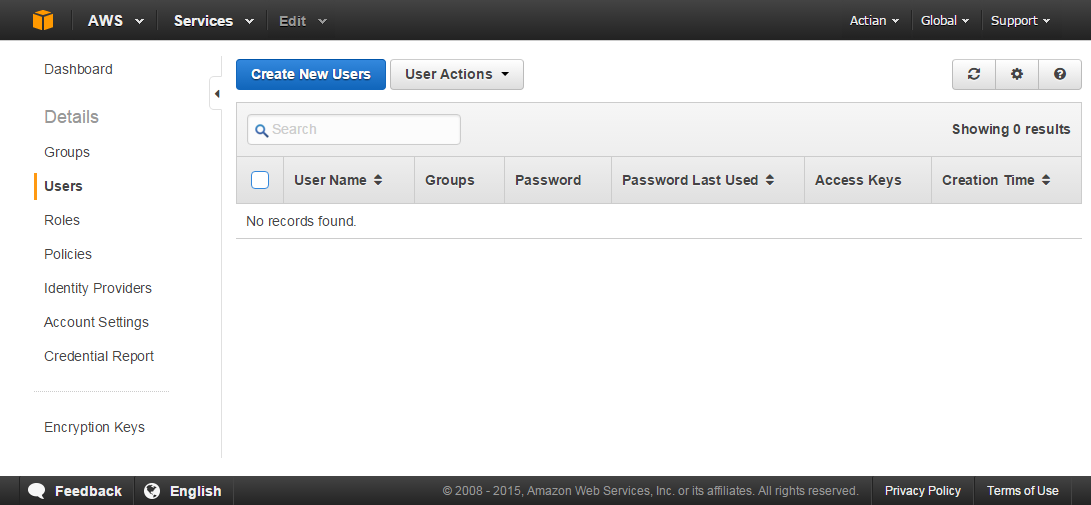
1. Access Key ID;
2. Secret Access Key.

These are not available directly from your AWS Account but relate to a ‘user’ which you need to create under your Account and give the necessary permissions to create your AWS Instance (VM) via Vagrant.

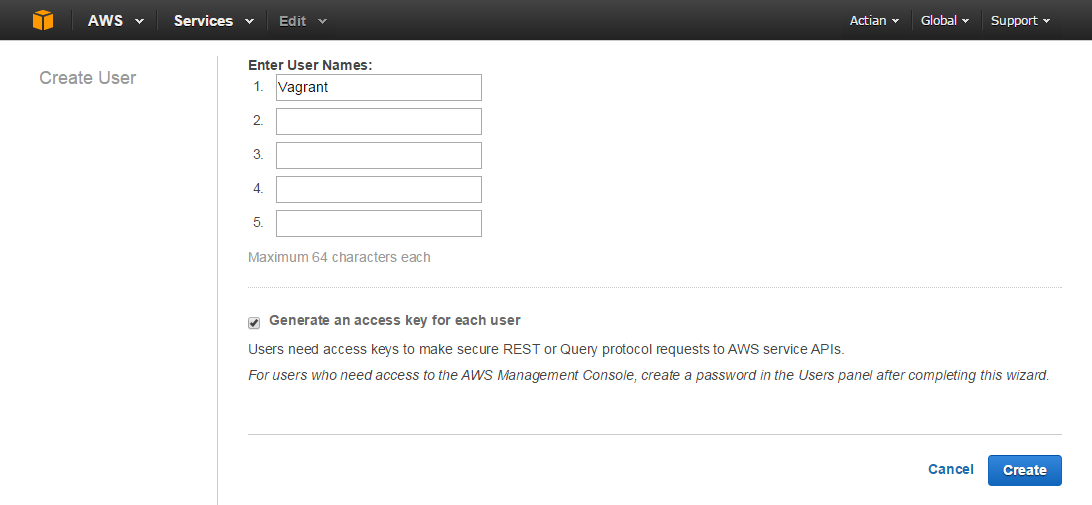
From the AWS Console select ‘Identify & Access Management’

Select ‘Users’ from the left panel.

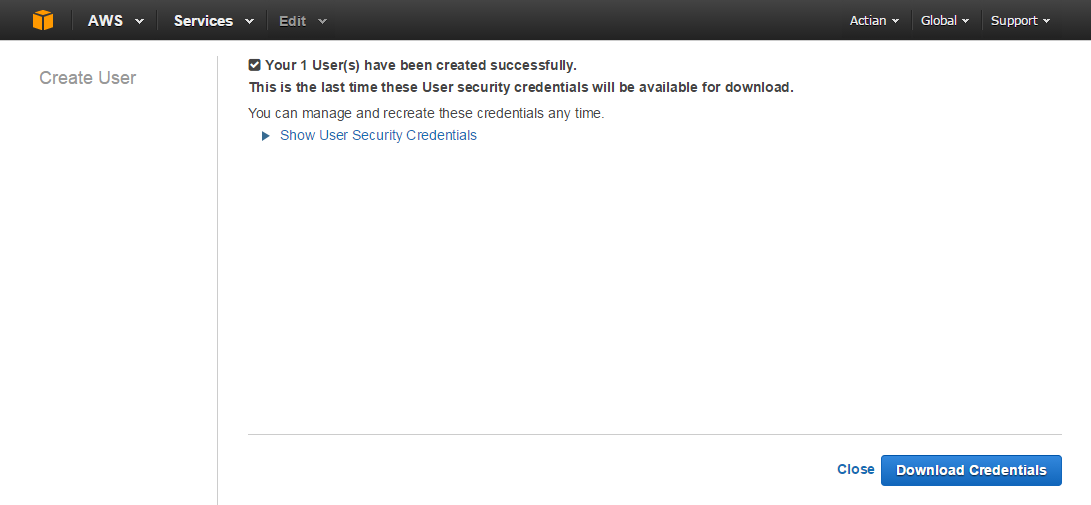
Select ‘Create New Users’



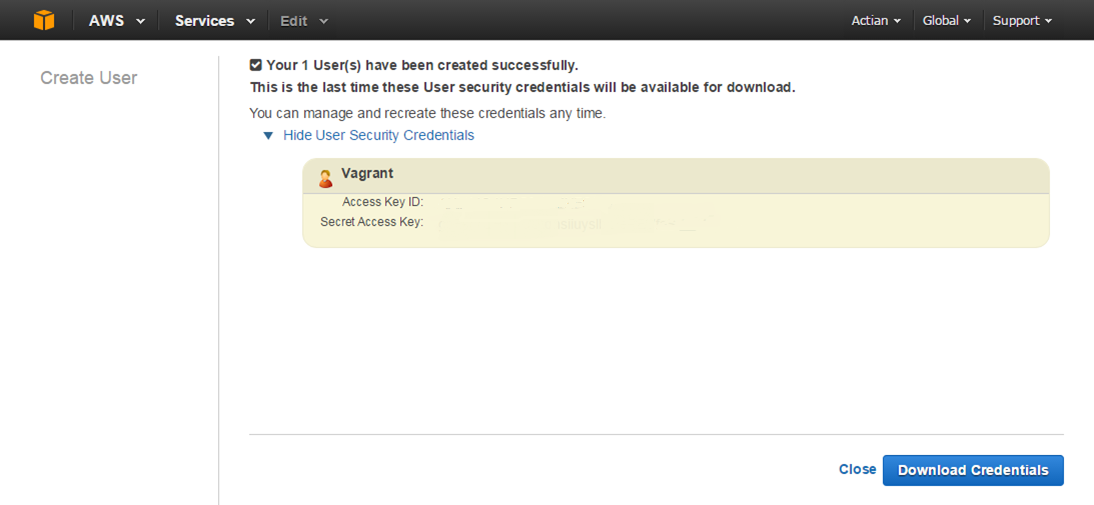
Enter ‘Vagrant’ as your user



Select ‘Create’



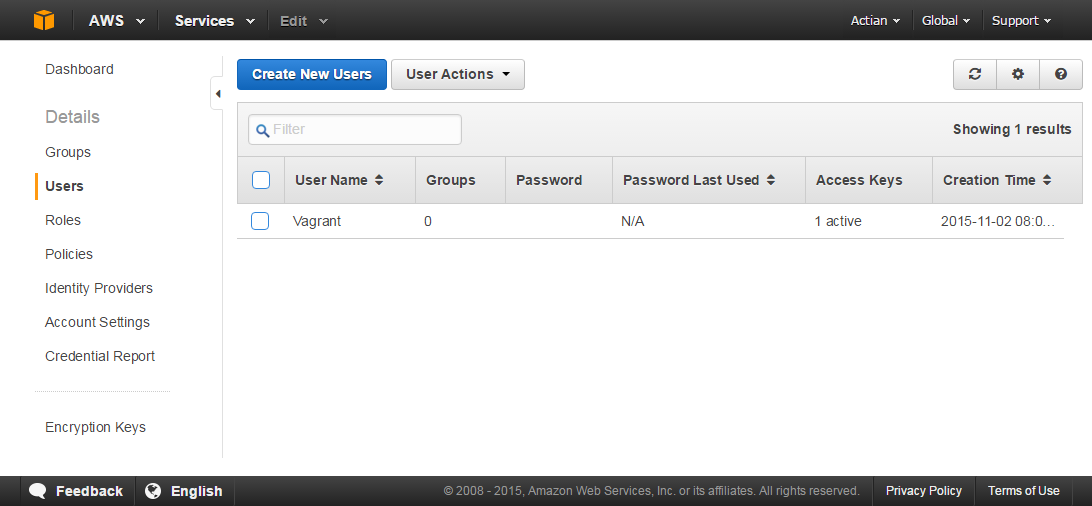
Select the ‘Show User Security Credentials’



Note – The actual keys of the author have been removed.

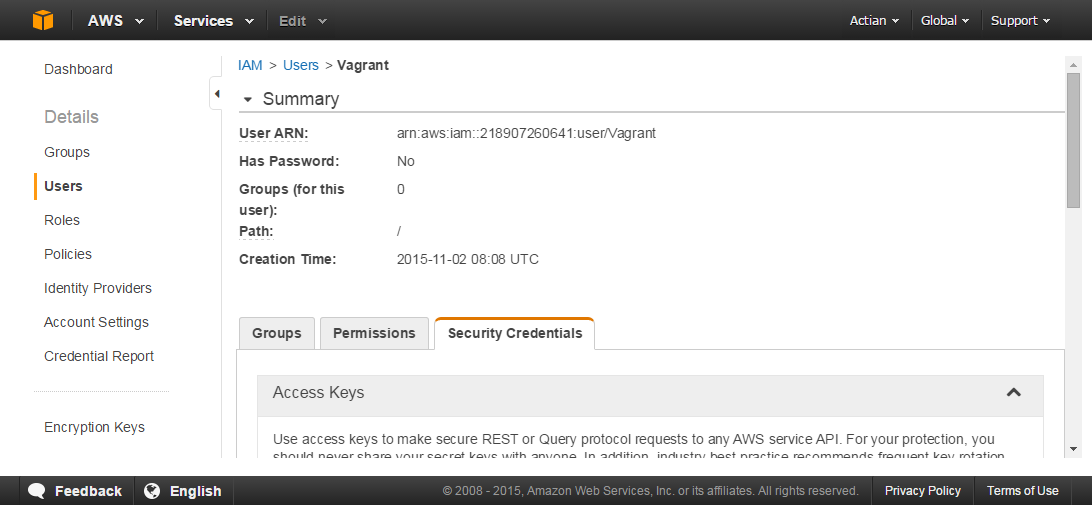
These Security Credentials are required to access your AWS Account using Vagrant. You can copy them or use the ‘Download Credentials’. This will download a permanent copy as ‘credentials.csv’ because as it states this is the last time you can see them in the AWS Console.

Select ‘Close’

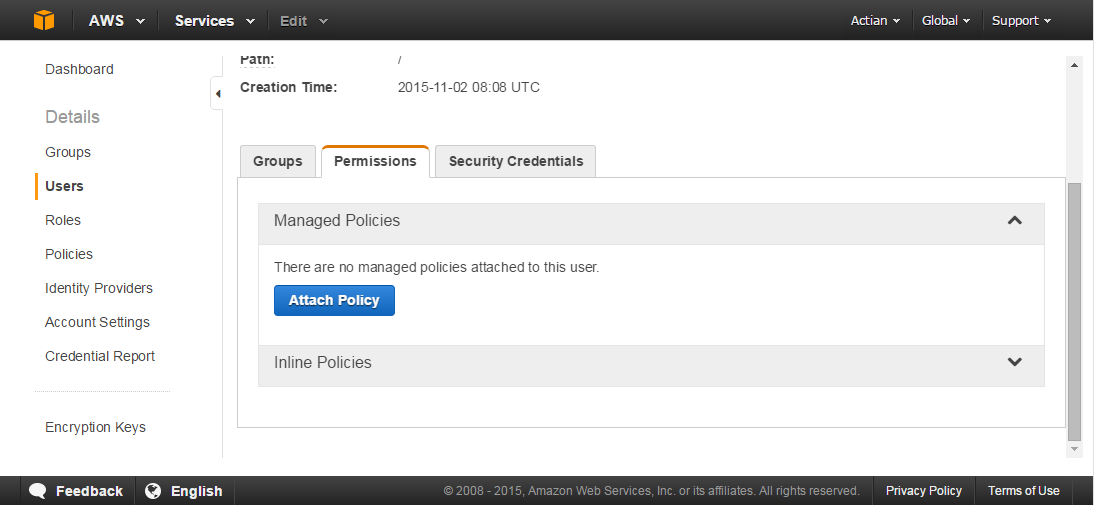


Access is required to create instances on AWS. The ‘Vagrant’ user created above has no permissions by default, so next we need to grant these permissions.

Click on the ‘Vagrant’ user.



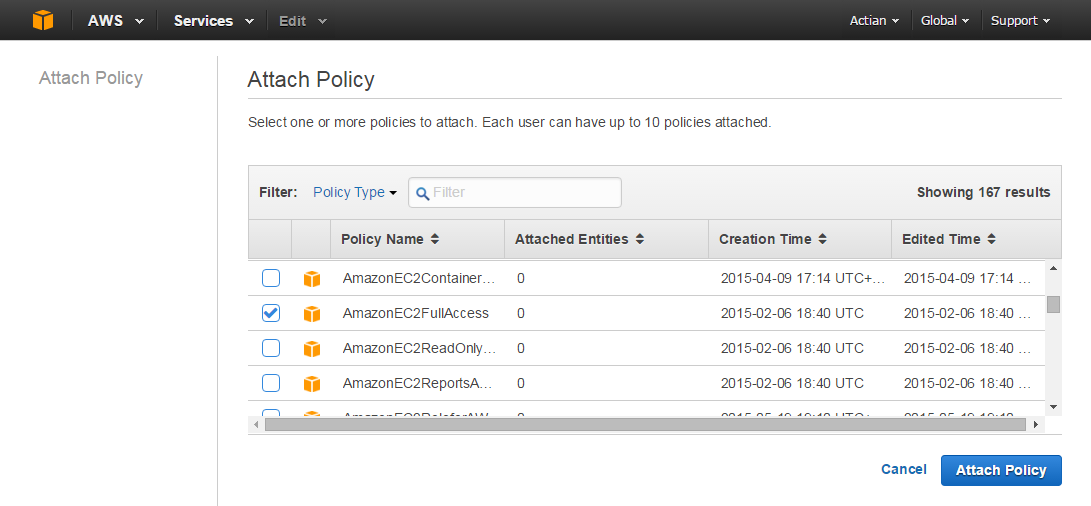
Select the ‘Permissions’ TAB.



Select ‘Attach Policy’

Scroll down and select ‘AmazonEC2FullAccess’.

More restrictive policies may be appropriate based on your own security requirements.



Select ‘Attach Policy’.

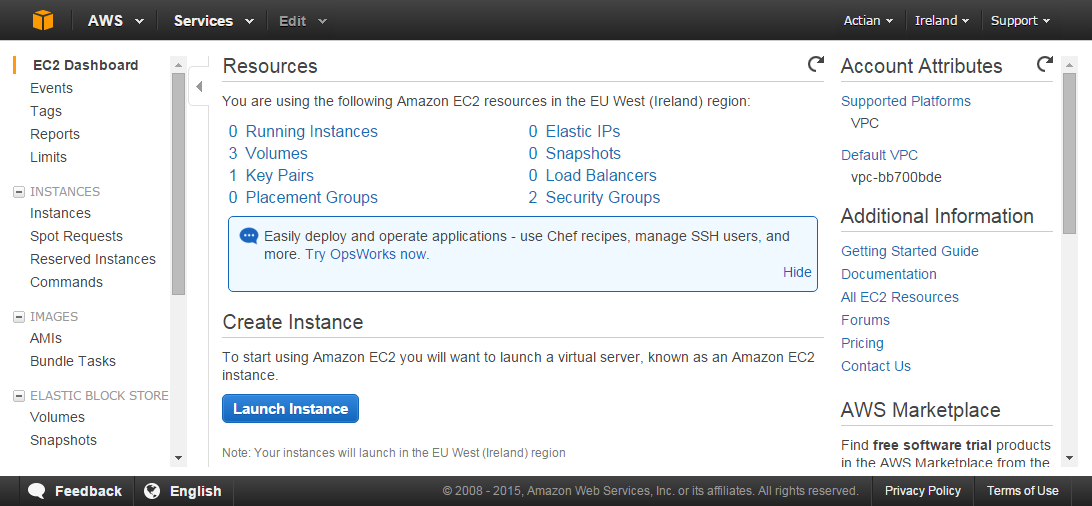
Your AWS user is now complete and this will be used by Vagrant to create an Instance using the Security Credentials related to this user.

## Port Access

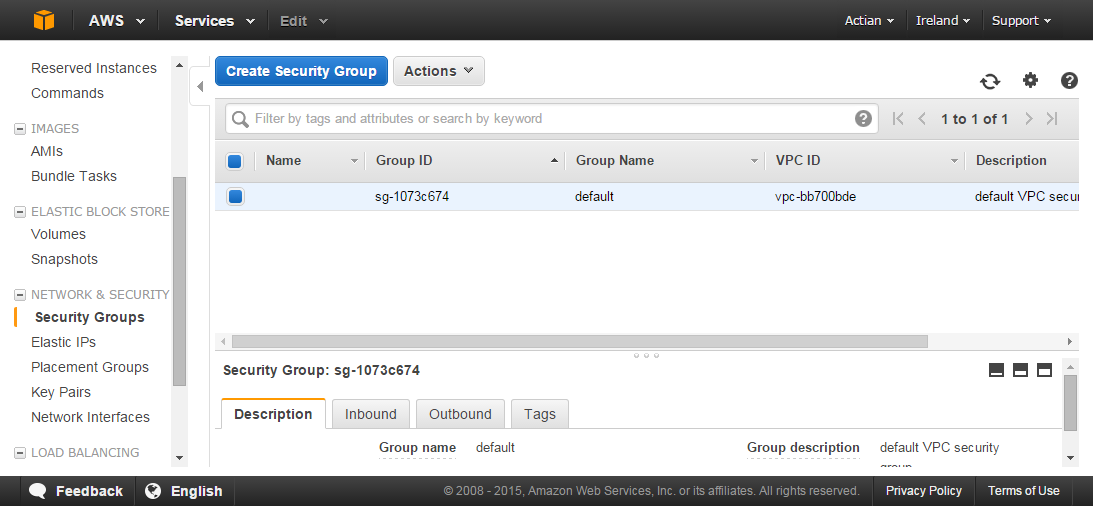
When an AWS Linux instance is created, unlike Azure, there is no SSH port open by default. For this example we will modify the default group to open this port as this is essential to allow Vagrant to provision the machine. The user ‘Vagrant’ created previously is a member of the ‘Default’ group.

In the real world where you are creating multiple machines for a specific purpose e.g. to host Vector, you may wish to create your own group that opens this and other ports such as those required for ODBC and JDBC access.

From the AWS Console select ‘EC2’.



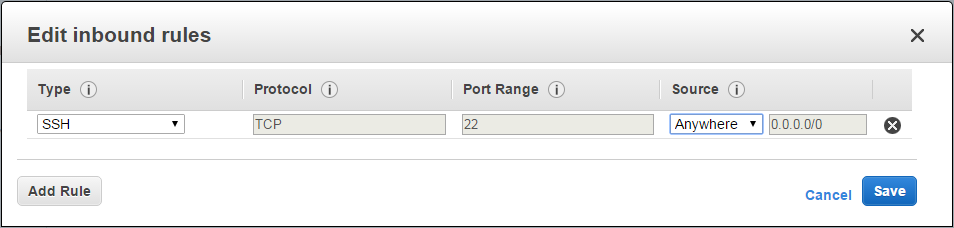
Select ‘Security Groups’ from the left panel (near bottom of left panel scroll list).



The default group should be selected automatically as it is your only group.

From the ‘Actions’ dropdown select ‘Edit Inbound Rules’ and select from the drop down options:

1. ‘Type’ as ‘SSH’;
2. ‘Port Range’ as ‘22’;
3. ‘Source’ as ‘Anywhere’.

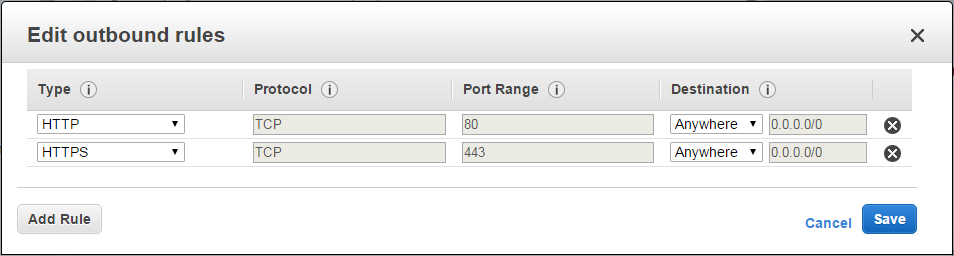


Again, for real-world use it would be better to specify ‘My IP’ with an IP range to restrict access to your Instance.

Select ‘Save’.

Opening Port 22 gives access to provision the machine via Vagrant as well as providing terminal access. However, this is not sufficient to fully provision the Instance as curl and yum are used and these use Ports 443 and 80 respectively (outbound).

From the ‘Actions’ dropdown select ‘Edit Outbound Rules’ and add the following rules:

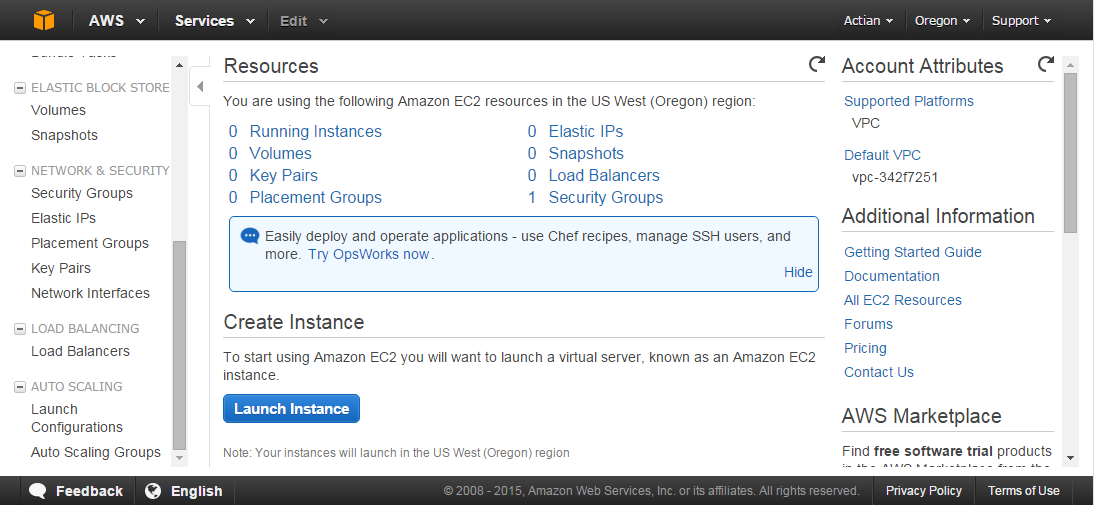


In Section 5.2 is documented the ports that also need to be opened to allow access to Vector installed on an AWS Instance. It is important to note, as stated in that section, unlike Azure these changes do not take effect immediately and only affect the next creation of an Instance with the Security Group **so need to be actioned before creating an Instance with Vector installed.**

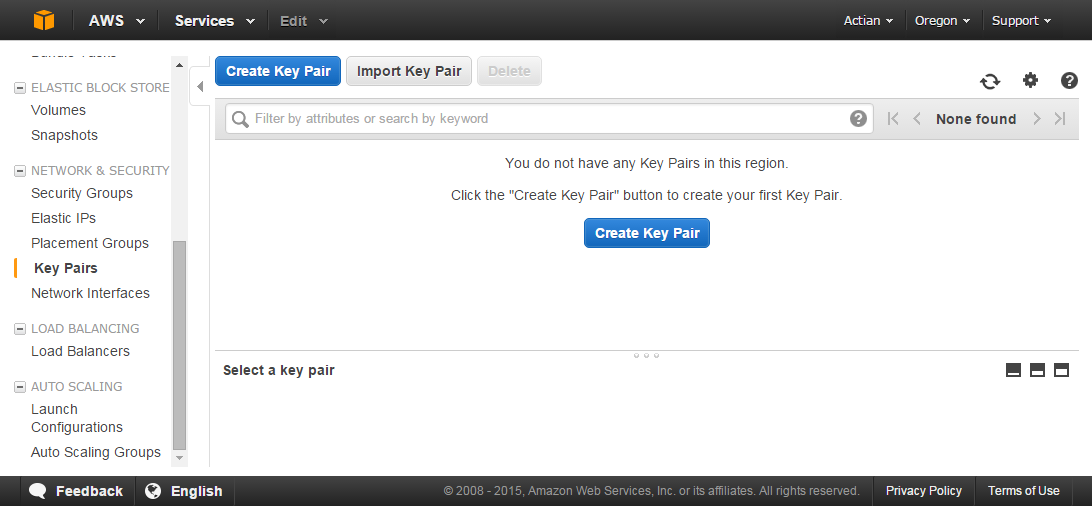
## Key Pairs - Certificate

For SSH access, a key pair is required. This can be generated via something like OpenSSL and then imported into AWS. However, it seems the most reliable approach is to generate this in your AWS Account and export it for use by Vagrant.

From the AWS Console select ‘EC2’.

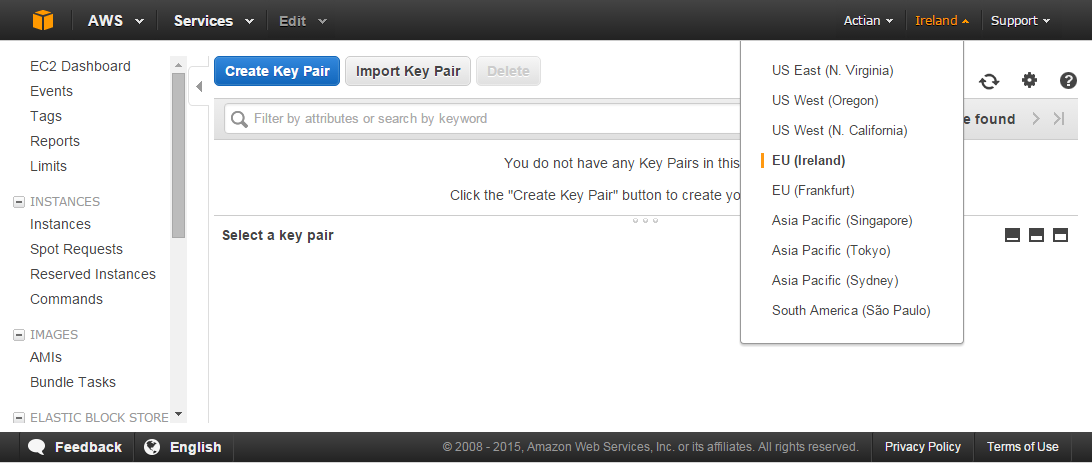


Select ‘Key Pairs’

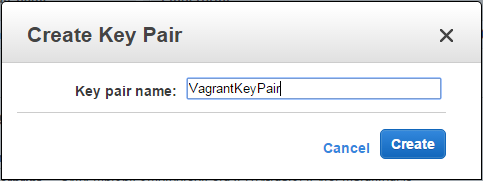


Before creating a key pair it is important to realise that the key pair is associated with the region in which you intend to create an Instance. In this example the AWS Account had as its default region US West (Oregon) as can be seen from the top right of the previous screenshot.

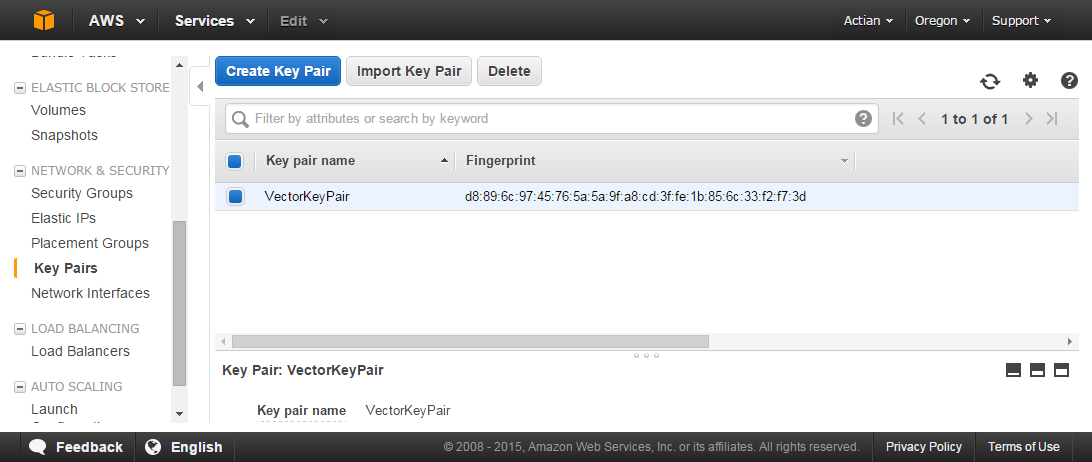
In this case, the region was changed to EU-WEST-1 (Ireland) as demonstrated below.



Select ‘Create Key Pair’



Give the key pair a name and select ‘Create’

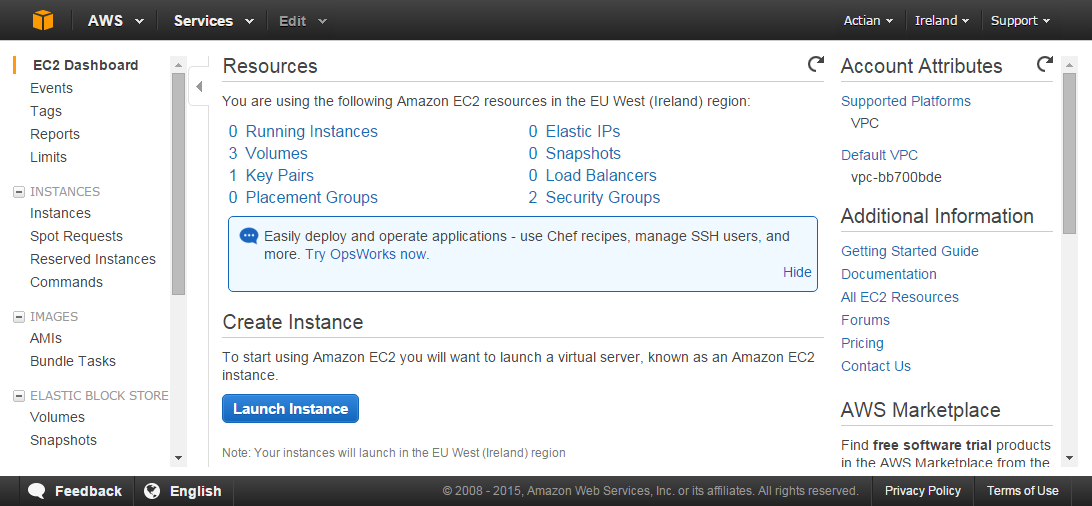


The key pair file ‘VagrantKeyPair.pem’ is automatically downloaded. Copy or move this to your Vagrant folder e.g. C:\AWS\_Example.

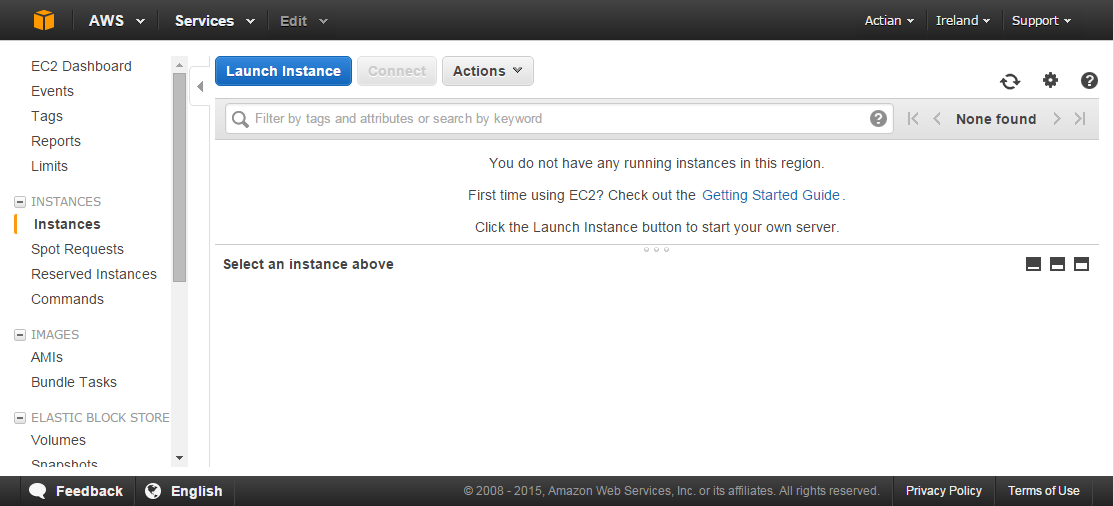
## Amazon Machine Image

An Amazon machine image name (AMI) is required of the particular box you wish to build. This is easier than with Azure as the Amazon image names can be viewed from the AWS Console.

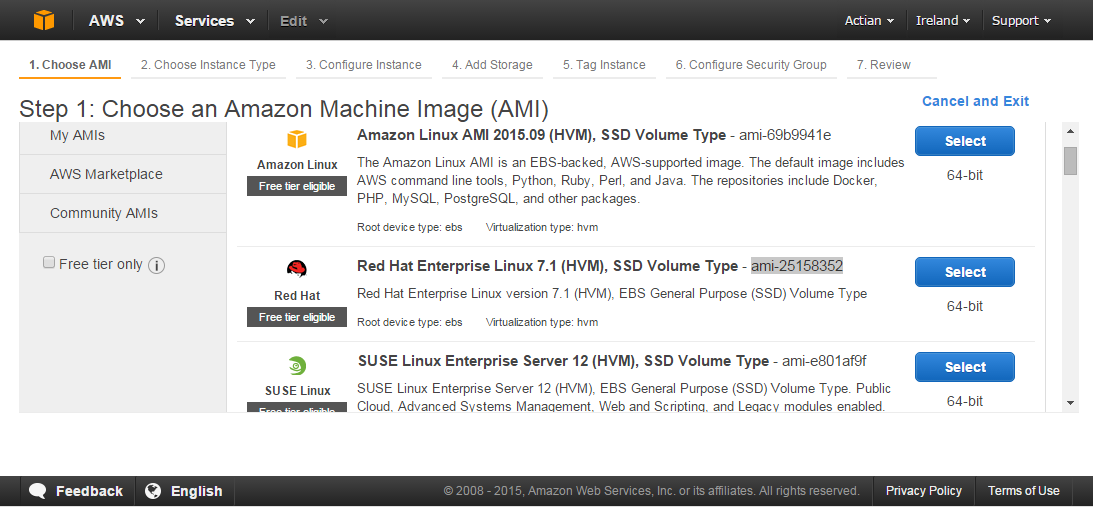
From the AWS Console select ‘EC2’.



Select ‘Instances’ from the left panel.



Select ‘Launch Instance’.



From this page in the Console you can search for an appropriate image and the image name can clearly be seen.

BEWARE! – The AMI name for the same image is different across Amazon regions.

# Vagrant

## Prerequisites for AWS

You should already have installed vagrant. There are a couple of additional essential installs to create an Instance (VM) against AWS:

1. Install the Vagrant AWS plugin:

vagrant plugin install vagrant-aws

1. Install a dummy Vagrant AWS Box:

vagrant box add dummy https://github.com/mitchellh/vagrant-aws/raw/master/dummy.box

## Vagrantfile Changes

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

1. Your user Access Key ID and Secret Access Key;
2. Your certificate:

* The .pem file.

You obtained these earlier in Sections 2.1 and 2.3.

In your chosen folder e.g. C:\AWS\_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 AWS 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 AWS 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 AWS components created which you may wish to reflect your own naming standards.

# Provider - Amazon Web Services (AWS)

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

override.vm.box = 'dummy'

override.vm.boot\_timeout = 1000

# Vagrant share does not work for AWS provider.

override.vm.synced\_folder '.', '/vagrant', disabled: true

aws.region = 'eu-west-1'

# You may wish to set this to something appropriate to your location.

# The region must be a match for the region under which the key pair was created.

# Official AWS RedHat 7.1

aws.instance\_type = 't2.medium'

aws.ami = 'ami-25158352'

override.ssh.username = 'ec2-user'

aws.tags['Name'] = **'VectorEvaluationVM'**

aws.access\_key\_id = **'#####################'**

# Your AWS Account 'user' Access Key ID.

aws.secret\_access\_key = **'########################################'**

# Your AWS Account 'user' Secret Access Key.

aws.keypair\_name = **'VagrantKeyPair'**

## You can stick with the naming of this key pair.

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

# You can stick with the naming of this file but you must generate

# your own.

override.ssh.insert\_key = false

override.ssh.pty = true

end

Note – Instance type ‘t2.medium’ is NOT free in the AWS EC2 trial. It has been used as it is the minimum machine in terms of memory to support a Vector installation. The free options do not have sufficient memory.

## Linux Provisioning User

It is worthy of note that the logon user name for an AWS Image is pre-set in an AMI. In the Vagrant example above the default ‘vagrant’ user cannot be created during the provisioning of the Instance so has been overridden to match one already available in the AMI used.

Each machine image has an associated user which can be used for access. For free AMIs, and generally those from the Community, there is a standard approach to the user and any password (usually there is not a password). However, this standard is not always adhered to and can prevent any SSH access to the Instance when created.

The general convention for this is explained here:

<https://alestic.com/2014/01/ec2-ssh-username/>

This is page is also useful reference for any SSH access problems and also references the user name convention.

<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html>

In this example the RedHat distribution was used with a user of ‘ec2-user’ and this is reflected in the Vagrant file.

With AWS, switching images requires a check to find the correct connection user. In the case of Community AMIs this is not always straight forward to find these details. For more information:

## Gotchas

1. Time Out

The default timeout for vagrant when waiting for the VM to start is 360 seconds. This was set to 1000 for AWS in the light of experience with Azure. However, it was found that AWS was much quicker and the default may suffice:

override.vm.boot\_timeout = 1000

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.

2. pty

The following setting seems to be mandatory to get things working on AWS as it is for the Azure Provider:

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.

3. Chef

Chef will not auto install on Azure.

It is as yet undetermined whether this is the case for AWS. The same command line approach was taken for AWS as it was for Azure to preserve a ‘one for all’ provider Vagrant file.

4. Vagrant Share

The Vagrant share does not work in AWS so must be disabled otherwise the provisioning will fail.

override.vm.synced\_folder '.', '/vagrant', disabled: true

5. Wrong SSH Shell

An attempt was made to use a CentOS AMI from the Community. The following error was encountered as soon as the Instance was running and Vagrant attempted to provision:

The configured shell (config.ssh.shell) is invalid and unable to properly execute commands.

The most common cause for this is using a shell that is unavailable on the system.

Please verify you're using the full path to the shell and that the shell is executable by the SSH user.

At the time of writing this problem was not resolved, other than using a different AMI.

# AWS Linux Instance Access

The approach is identical to that for Azure and is covered in detail by the sister document ‘Quick Start Guide to using Actian Vector on Microsoft Azure’.

For AWS the basis for generating the private key for access is based on the key pair .pem file created in Section 2.3.

Please Note:

Since the private key is associated with the ‘ec2-user’ user this is what you should logon as then switch to actian.

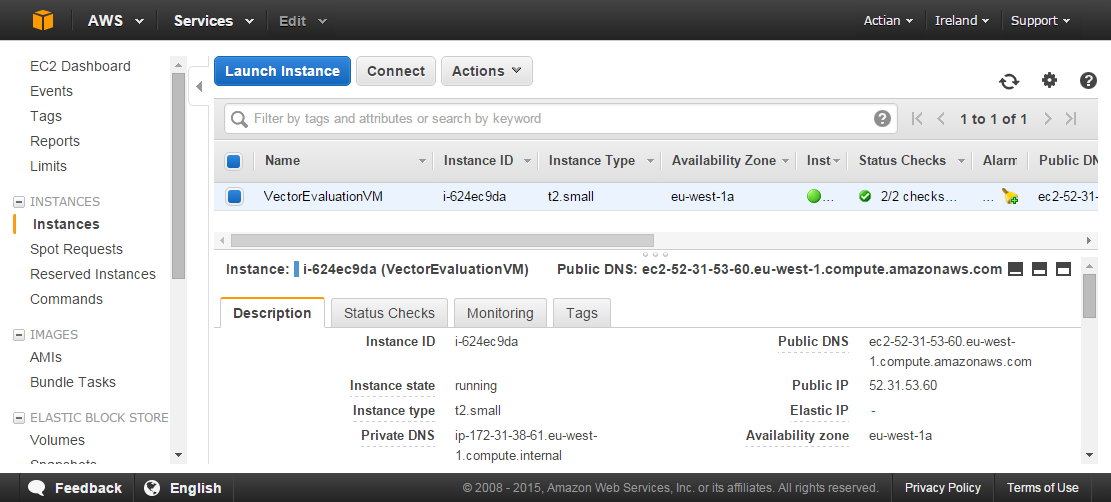
sudo su – actian

To configure access the Public DNS name or Public IP are required. Unfortunately, unlike Azure the DNS name is not predictable. To get these details for an Instance once Vagrant has created it:

From the AWS Console select ‘EC2’

Select ‘Instances’ from the left panel.

Select the new Instance:



The ‘Description’ Tab shows the Public DNS and Public IP.

The IP allocated and the DNS (which is based on the IP) appear not to be fixed if the machine is ‘halted’. It may be possible to fix these but details are not currently known.

# External Access

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

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

The examples that follow are based on the settings for the Vector installation created by the Vagrantfile at <https://github.com/ActianCorp/Vagrant-Vector-Install>for which this is a supporting document.

## 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, 44103, 27712 and 27719 respectively.

To ensure that each of these ports is open you can open each individually 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:

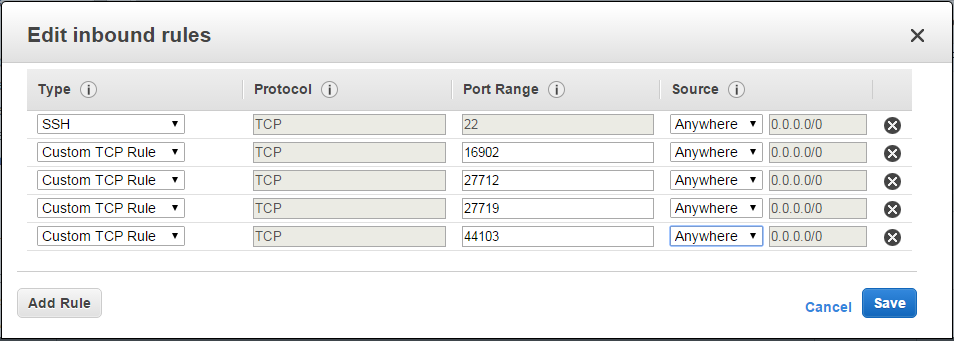
iptables: Setting chains to policy ACCEPT: filter [ OK ]

iptables: Flushing firewall rules: [ OK ]

iptables: Unloading modules: [ OK ]

## AWS Ports

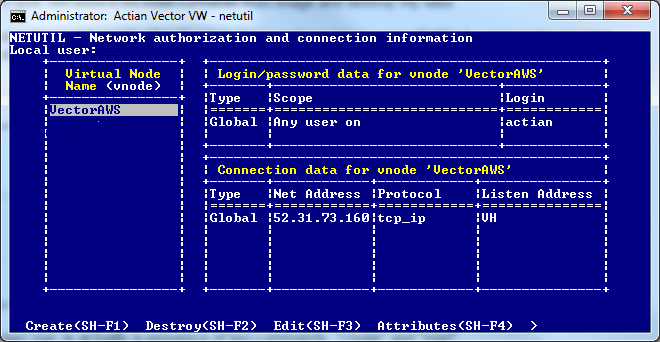
The next step is to open these ports to your Instance using your Amazon Account. Opening ports was covered in Section 2.2 so it is just a case of adding ports 16902, 27712, 27719 and 44103 to the inbound rules:



PLEASE NOTE – Unlike Azure these changes do not take effect immediately and only affect the next creation of an instance using the Security Group.

## Ingres Net Connection

This connection relies on having an Ingres Client runtime (downloaded from [esd.actian.com](file:///C:\Users\posda01\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\1ZMC5XT1\esd.actian.com)) to create a NetUtil entry as follows:



The Net Address is set from either the Public DNS or Public IP allocated as documented in Section 4.

In some cases a dynamic vnode, an equivalent to the above, may be preferable and this takes the format:

@**host**,tcp-ip,**InstallationID**[**user**,**password**]::**database**/ingres

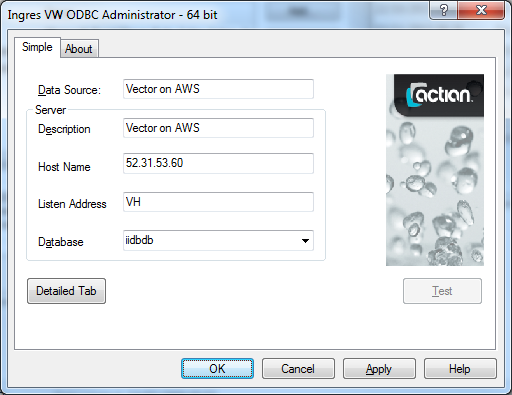
For our example above which has a Vector Installation of ‘VH’, user ‘actian’ (password same) and accessing the Vector installation database ‘iidbdb this would result in:

@52.31.73.160,tcp-ip,VH[actian,actian]::iidbdb/ingres

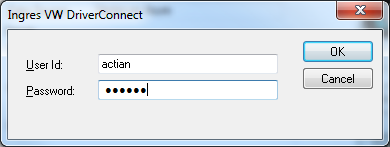
## ODBC Connection

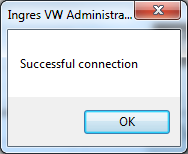
To configure your Windows ODBC connection is little different from that for any other. You will however need the Vector ODBC driver from Actian. This is installed by default if your client (in the example in this document the Windows machine from which you ran Vagrant) has Vector, Ingres or the Ingres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

The Ingres Client runtime is downloadable from [esd.actian.com](file:///C:\Users\posda01\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\1ZMC5XT1\esd.actian.com) under the ‘Actian Vector and Ingres Drivers’ menu item.



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’, but the process will be very similar for any other tool that uses JDBC as a connection protocol.

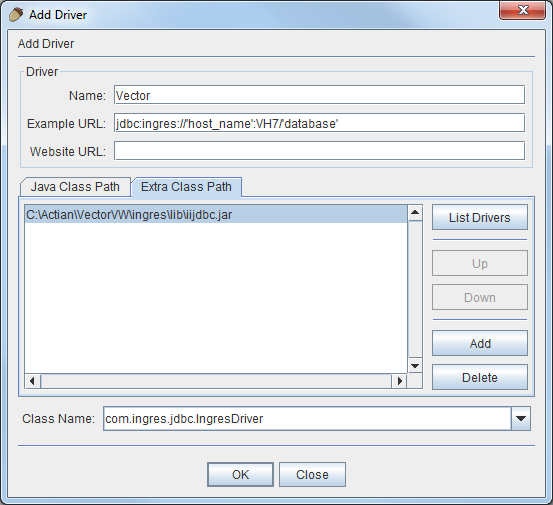
To make any JDBC connection the appropriate Vector JDBC driver JAR file is required and added to the CLASSPATH as demonstrated by the screen shot below. In this example the JAR file was used from a local Vector installation but it can be downloaded separately from [esd.actian.com](file:///C:\Users\posda01\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\1ZMC5XT1\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 AWS Instance Vector installation.

