EC2 Windows

Dr Peadar Grant

September 23, 2021

1 Remote desktop protocol

Windows Server is traditionally managed using its GUI, although remote PowerShell use through various means is becoming more popular.

GUI management can be done via a directly or KVM-connected video display, mouse and keyboard. As with linux instances, AWS doesn't provide emulated KVM access. Only a screenshot picture of the instance can be taken.

When learning Linux, we used SSH to connect. Remote Desktop Protocol (RDP) is the standard way to manage remote windows servers (and clients) and is built into Microsoft Windows. Key things to note:

- RDP runs over TCP on port 3389.
- RDP must be enabled on the Windows control panel of the server for it to work. AMIs for Windows on AWS have RDP enabled already.
- RDP sessions are not as "independent" of each other as SSH sessions are:
  - Generally Windows Server supports a maximum of two concurrent separate user logins over a combination of RDP (and the local console, which isn't present on AWS).
  - Windows Clients (e.g. XP, 7, 8, 10) normally support only one login at a time (over all methods). If the same user logs in, the existing session should be picked-up/transferred.
- RDP allows a number of different enhancements to be selected when connecting:
  - The resolution of the remote session can be set when connecting.
  - Current RDP versions allow multi-monitor support.
  - File sharing can be enabled for the session so that necessary files can be easily transferred between the client and RDP server.
  - Ports can also be attached.
  - Audio and printer output can be redirected back to the client.
- RDP does not support user managed keys like SSH, but can support single sign-on through Active Directory if both the client and server share the same domain.

1.1 Terminal Services

Terminal Services is a licensing extension to remote desktop to enable a number of different concurrent remote sessions to be open. It is sometimes used to allow users to "pick up" their desktop at different computers, often thin clients.

1.2 Alternatives

VNC is often used in Windows to present the console remotely, in either view-only or view & control mode. VNC is cross platform and has clients for almost every operating system, but is limited somewhat by its weak authentication.

Proprietary solutions like TeamViewer and GoToMyPC require additional client software, can be easier to setup for non-technical users and often can traverse NAT routers automatically.

1.3 Command-line access

- **PowerShell remoting** gives access to the PowerShell interface of a remote Windows computer. It can be used on a basic level like SSH, but underneath is quite different, with different strengths and weaknesses.
- **SSH** access to Windows is possible using OpenSSH, offering access to `cmd.exe` or `PowerShell` from remote computers.

2 VPC setup

VPC for this week in the file `ec2_windows_vpc_setup.ps1`

This is largely the same as the linux Lab except that:

- Checks for the existence of your `LAB_KEY` before continuing.
- Creates the security group `LAB_SG`
- Adds a rule to `LAB_SG` to permit RDP (Port 3389) access.

Output should be similar to:

```
KeyPair: key-09a3afe83db8e8381
VPC: vpc-0209bd43322117a41
Subnet: subnet-05fa961d022b72730
IGW: igw-0d070969dfd43c2cf
Route Table: rtb-0b56038dfec860e3d
{
    "Return": true
}
Security Group: sg-0326166674b0b68f0
Modified security group to permit RDP access
```

3 Lookup Image Ids automatically (move for 2021)

Image IDs are region and account-dependent. They also get updated as Amazon update the images.

```
# print out list of Windows AMIs
ami ssm get-parameters-by-path -path /aws/service/ami-windows-latest --query "Parameters[].Name
```

The "standard" windows image we will use is `Windows_Server-2019-English-Full-Base`.

4 Provisioning a Windows Instance

These steps assume you are familiar with provisioning a linux instance from the previous week's lab.

4.1 Launching the instance

To start a Windows instances, we mainly need the same set of information:

- The AMI of the Windows image. We will use *Microsoft Windows Server 2019 Base*. Check the AMI as it appears on your own account.
- The Subnet ID to launch the instance into
- The instance type (from `aws ec2 describe-instance-types`)
- Security Group ID

The Key Pair is still needed, but it serves a different function in Windows instances compared to Linux instances. We use exactly the same launch command as for Linux instance last week.

We can use Systems Manager to launch an instance using the following syntax. Instead of giving an AMI directly we use `resolve:ssm:` to tell AWS to look this value up in SSM.

This week though we are are going to capture the command output into a variable $Output to work with it in Powershell.

```
# Capture the command output
$Output = (aws ec2 run-instances `
--subnet-id subnet-05fa961d022b72730 `
--instance-type t2.micro `
--image-id resolve:ssm:/aws/service/ami-windows-latest/Windows_Server-2019-English-Full-Base `
--key-name LAB_KEY `
--security-group-ids sg-0326166674b0b68f0 `
)

# Print the JSON to screen (as if we didn't capture)
$Output

# Do the conversion
$InstanceInfo=$Output | ConvertFrom-Json

# Extract the instance ID
$InstanceId = $InstanceInfo.Instances[0].InstanceId
```

5 Connecting over RDP

To connect to our Windows instance over RDP we need to know the Public IP and the password for the administrator account.

5.1 Determining the public IP

We can request the full information (including the public IP). Here we use PowerShell to extract the public IP.

```
# Print the JSON
aws ec2 describe-instances --instance-id $InstanceId

# Get the instance information for the new instance as PS object
$Instance=(aws ec2 describe-instances --instance-id $InstanceId | ConvertFrom-Json).Reservation

# Extract the public IP
$PublicIp = $Instance.NetworkInterfaces[0].Association[0].PublicIp
```

Other information can be found in a similar way.

5.2 Getting password data

To get the password data for a given instance ID:

```
aws ec2 get-password-data --instance-id i-01248802a6d0dd331
```

which will return something like

```
{
    "InstanceId": "i-01248802a6d0dd331",
    "PasswordData": "\r\nJT8Nj6e/8FhnCjpHfzyBlRB8xiAUFblToIQjn8kstz7R6Vz1yN67Mp76zMZFvR1Pn0Divo
    "Timestamp": "2020-10-27T22:07:39.000Z"
}
```

If the password is blank, it means that the post-launch setup hasn't yet completed. Just pause here and try it again in a couple of minutes. When the PasswordData field is non-empty you can continue.

The PasswordData is the password encrypted with your public key. You can pass an additional parameter to `get-password-data` to decrypt this using your private key. Obviously replace the private key path with your own!

```
aws ec2 get-password-data --instance-id i-01248802a6d0dd331 --priv-launch-key C:\Users\peadar\.
```

If successful, this will produce the output:

```
{
    "InstanceId": "i-01248802a6d0dd331",
    "PasswordData": "5G.XW752i?VoJm;Y8Ch-rEI5e?i8fFC5",
    "Timestamp": "2020-10-27T22:07:39.000Z"
}
```

The decrypted password shown matches the Windows local Administrator account.

5.3 Connecting with the graphical client

Look for Remote Desktop Connnection in the start menu. Use the IP address, username Administrator, and the password and hit Connect.

You should be at a standard Windows desktop running on the remote machine. At this point you can administer the machine any way you want, including setting up a new admin user account and/or changing the password.

5.4 Starting RDP client from PowerShell

The RDP client is a program called `mstsc.exe`. To show its options type

```
mstsc /?
```

We will connect by typing:

```
mstsc.exe /v:$PublicIp
```

Put in Administrator and the password. (It's not simple to launch RDP directly with the password pre-filled.)

6 SSH server on Windows

Remote Desktop is just one way to access a Windows server for management. Windows now supports SSH access to the command-prompt (and to PowerShell).

6.1 Checking if OpenSSH is installed

On your AWS Windows Server open PowerShell and type

```
Get-WindowsCapability -Online -Name Open*
```

You will see something like:

```
Name         : OpenSSH.Client~~~~0.0.1.0
State        : Installed
DisplayName  : OpenSSH Client
Description  : OpenSSH-based secure shell (SSH) client, for secure key management and access to

Name         : OpenSSH.Server~~~~0.0.1.0
State        : NotPresent
DisplayName  : OpenSSH Server
Description  : OpenSSH-based secure shell (SSH) server, for secure key management and access fr
DownloadSize : 1297677
InstallSize  : 4946932
```

To install the OpenSSH server we type

```
Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0
```

If it succeeded you'll see

```
Path          :
Online        : True
RestartNeeded : False
```

Re-run the `Get-WindowsCapability` command to confirm it's now installed.

6.2 Server startup

SSH is provided by a server process called `sshd`.

First we confirm the service exists:

```
Get-Service sshd
```

From the output we note that it is installed but isn't currently running:

```
Status   Name               DisplayName
------   ----               -----------
Running  sshd               OpenSSH SSH Server
```

This server needs to be started and to be configured to start automatically on reboot:

```
# start immediately
Start-Service sshd

# Set to start automatically on boot
Set-Service -Name sshd -StartupType automatic
```

6.3 Testing SSH server from same EC2 instance

Now we can test it by connecting from the EC2 instance itself.

```
ssh Administrator@localhost
```

and entering the admin password. We're actually at the older command.com prompt, but this can be changed to Powershell using system configuration.

6.4 Connecting from our computer to EC2 SSH server

The main reason to use SSH would be to connect from our own local computer to the EC2 instance. SSH uses TCP port 22, which isn't currently permitted by the security group. There are two possibilities: create a second security group and assign it to the machine, or modify the existing group.

We will add a new rule to additionally permit SSH (as well as RDP) in:

```
aws ec2 authorize-security-group-ingress --group-id sg-08e934eb80d1d8f70 --cidr 0.0.0.0/0 --pro
```

Then

On our local computer we type:

```
ssh Administrator@$PublicIp
```

and put in the machine's password. (Notice here that although AWS did setup the private keys for us, that it doesn't inject the private key into the EC2 instance for us as with Linux.)

You have successfully connected over SSH if you see output similar to the following:

```
Microsoft Windows [Version 10.0.17763.1518]
(c) 2018 Microsoft Corporation. All rights reserved.

administrator@EC2AMAZ-3DE14BE C:\Users\Administrator>
```

This is actually the older command.com shell, rather than PowerShell. We can run powershell from here by just typing:

```
powershell
```

Typing `exit` will get us back out of Powershell, and `exit` again will close the SSH session.

6.5 Configuring SSH

If you intend to use SSH to work with Windows regularly you should take the time to configure it properly. Here are a few tips:

- There are a number of ways to reconfigure SSHd to launch PowerShell instead, with various drawbacks.
- Use the local user/group management to limit who is allowed to login over SSH.
- Use public/private keys and disable password authentication. You will have to consult the OpenSSH docs for Windows as the configuration is slightly different to Linux.
- If using SSH keys, learn about agent forwarding.