An Introduction to Using Python with Microsoft Azure

If you build technical and scientific applications, you're probably familiar with Python. What you might not know is that there are now tools available that make it easy for you to put your Python applications on Microsoft Azure, Microsoft's cloud computing platform. Microsoft Azure offers a comprehensive set of services that enable you to quickly build, deploy and manage applications across a global network of Microsoft-managed data centers. Some of the services that Microsoft Azure offers include blob storage, databases, and messaging mechanisms such as queues. All Microsoft Azure services are available by using Python.

Here are some types of activities that you can perform by using Python and Microsoft Azure:

- Remote debugging on Windows, Linux, and Mac OS.
- Cluster debugging.
- Use Python in-line with webpages that you serve.
- Create IPython Notebooks.
- Use the Message Passing Library (MPI) for High Performance Computing (HPC).

This article introduces you to two Python development tools. One is the IPython Notebook, which can be deployed to Linux and Windows virtual machines (VM) on Microsoft Azure. The other is Python Tools for Visual Studio (PTVS), which is for Windows users. After familiarizing you with the tools, the article discusses the following topics.

- Tools for developing Python applications on Microsoft Azure
- Deploying an IPython Notebook to Microsoft Azure
- Using Python to create VMs
- Using Python to create an application

Note: To use Python with Microsoft Azure, make sure you've downloaded and installed the Python Azure SDK. You also need to have a Microsoft Azure account. For more information about using Python with Microsoft Azure, go to the Python Developer Center.

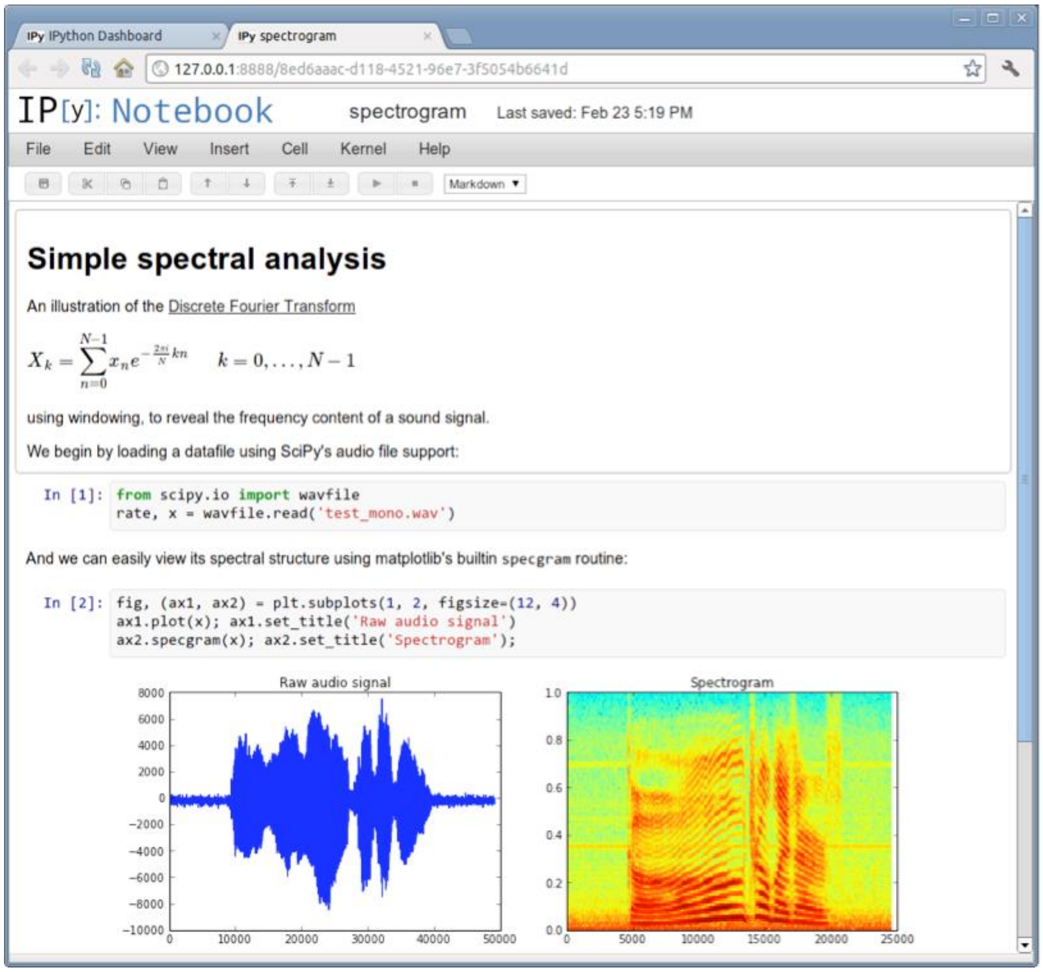
Tools for Developing Python Applications on Microsoft Azure

This section gives you an overview of the IPython Notebooks and PTVS development tools.

Using IPython Notebooks

The IPython project provides a collection of tools for scientific computing that include interactive shells, high-performance and easy to use parallel libraries, and a web-based development environment called the IPython Notebook. The notebook provides a working environment for interactive computing that combines code execution with the creation of a live computational document. These notebook files can contain text, mathematical formulas, input code, results, graphics, videos and any other kind of media that a modern web browser is capable of displaying.

The following screen shot shows an IPython Notebook that, in combination with the SciPy and matplotlib packages, analyzes the structure of a sound recording.


IPy IPython Dashboard
IPy spectrogram
127.0.0.1:8888/8ed6aaac-d118-4521-96e7-3f5054b6641d
IP[y]: Notebook
spectrogram
Last saved: Feb 23 5:19 PM
File Edit View Insert Cell Kernel Help
Markdown
Raw audio signal
Spectrogram

Simple spectral analysis

An illustration of the Discrete Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \dots, N-1$$

using windowing, to reveal the frequency content of a sound signal.

We begin by loading a datafile using SciPy's audio file support:

```
In [1]: from scipy.io import wavfile
        rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin `specgram` routine:

```
In [2]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
        ax1.plot(x); ax1.set_title('Raw audio signal')
        ax2.specgram(x); ax2.set_title('Spectrogram');
```

Figure 1

Using PTVS

If you're a Windows user you can take advantage of PTVS. This is a free download and you don't have to buy Microsoft Visual Studio. Instead, you can use PTVS combined with the integrated/isolated Visual Studio shell, which is also free.

PTVS has many features. It supports CPython, IronPython, editing, browsing, IntelliSense (auto-completion), mixed Python/C++ debugging, remote Linux/MacOS debugging, profiling, HPC clusters, multiple REPL's, IPython, and Django.

Note: IronPython is an open-source implementation of the Python programming language which is tightly integrated with the .NET Framework. IronPython can use the .NET Framework and Python libraries, and other .NET languages can use Python code just as easily. IPython is a different open source project. IPython provides a rich architecture for interactive computing with interactive shells, and a browser-based notebook with support for code, text, mathematical expressions, inline plots and other rich media.

For Windows machines, PTVS provides a number of facilities to debug applications launched from within Visual Studio, as well as the ability to attach to existing Python processes on both local and remote machines. If your application runs on a different operating system, such as Linux or OS X, or if you do not have permissions to install or run MSVSMon on a Windows machine, PTVS has an alternative remote debugging option that does not require any separate processes, and runs on any OS capable of running Python itself. For more information, see Cross-platform Remote Debugging on the Python Tools for Visual Studio website.

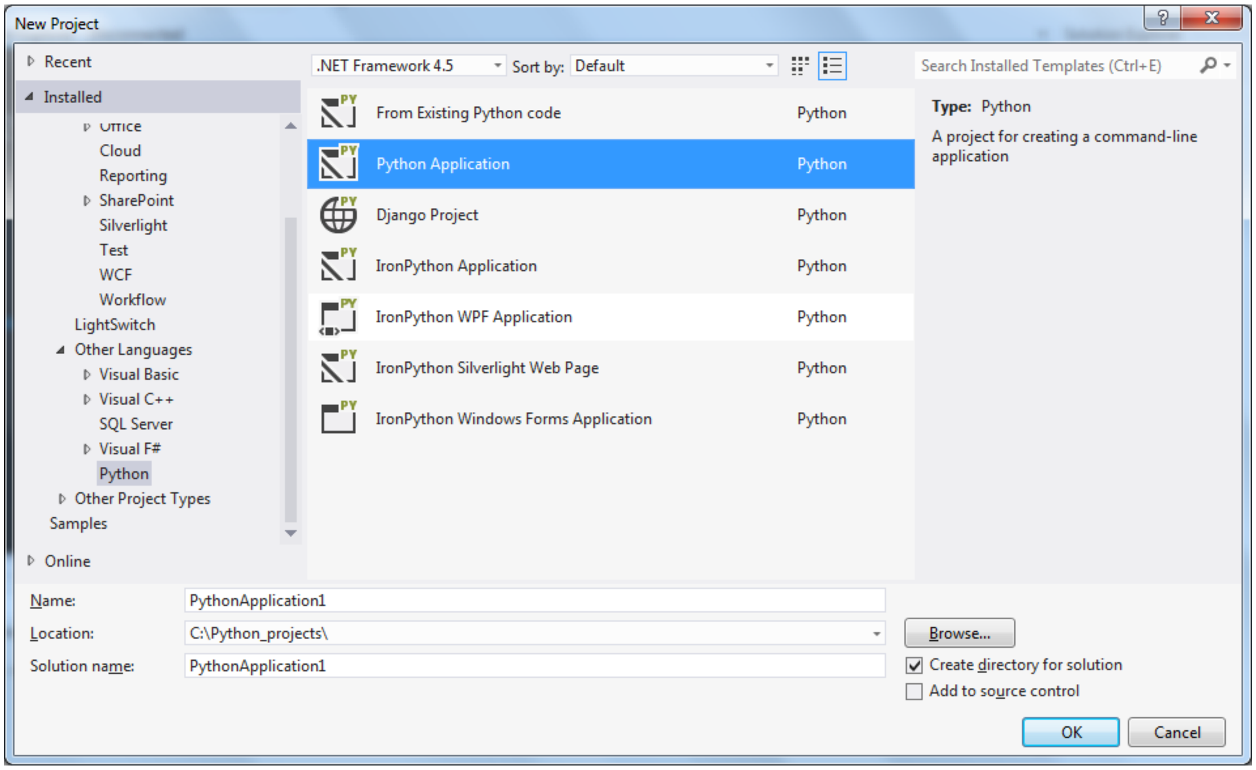
Installing PTVS

If you don't have Visual Studio, go to the CodePlex Python for Visual Studio page and download PythonToolsIntegrated.exe. You then need to download some version of Python. You can do this from the Python site.

If you have Visual Studio 2010 or later, go to the CodePlex Python for Visual Studio page and download the appropriate version of PTVS.

Note: With the release of Visual Studio 2013, the integrated/isolated Visual Studio Shell and PTVS are integrated in a single installer. You can download it at Python Tools for Visual Studio. Select PTVS 2.0 VS 2013.msi.

Once you've installed everything, check to see that the tool is working correctly. Start Visual Studio or the integrated Visual Studio shell. Select the **File** menu and then select **New Project**. The following screen shot shows an example of what you should see, depending on what else you have installed.



Type: Python

A project for creating a command-line application

Name: PythonApplication1

Location: C:\Python_projects\

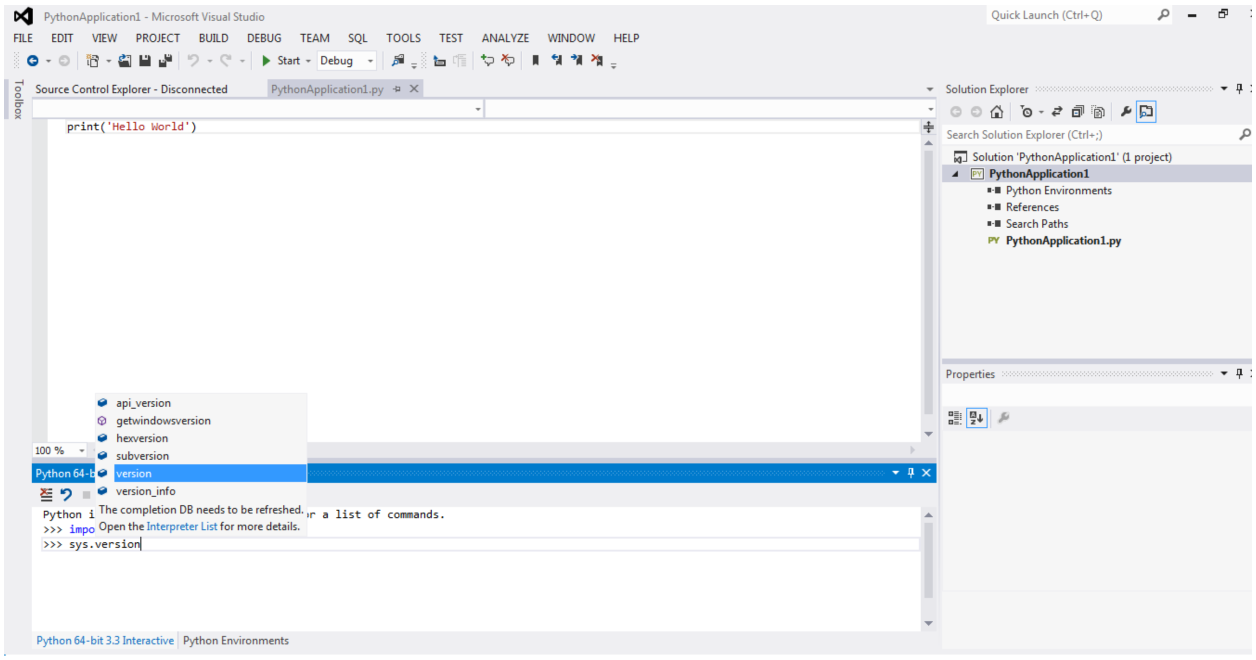
Solution name: PythonApplication1

[✓] Create directory for solution

[ ] Add to source control

Figure 2

Once you click **OK**, you should see the development environment. To open an interactive window, select the **Tools** menu, select **Python Tools**, and then select the **Interactive** menu item. In the interactive window, first enter **import sys** and then enter **sys.version**. The following screen shot shows an example of what you should see.


PythonApplication1 - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ANALYZE WINDOW HELP
Start
Debug
Toolbox
Source Control Explorer - Disconnected
PythonApplication1.py
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'PythonApplication1' (1 project)
PythonApplication1
Python Environments
References
Search Paths
PythonApplication1.py
Properties
100 %
api_version
getwindowsversion
hexversion
subversion
version
version_info
Python 64-bit 3.3 Interactive
Python Environments

```
print('Hello World')
```

The completion DB needs to be refreshed.
Open the Interpreter List for more details.

```
Python i                                          r a list of commands.
>>> impo
>>> sys.version
```

Figure 3

Figure 3 shows an example of Visual Studio IntelliSense. IntelliSense is an auto-completion feature that shows you the possible ways to complete a statement. IntelliSense helps you to develop your Python cloud applications quickly and to run those applications either from the desktop or by deploying them to the cloud. IntelliSense provides three types of help.

- Completion, which displays a list of words that are appropriate to the context.
- Signatures, which appears when you are writing a function call. It includes documentation and any available parameter information.
- Quick info, which is a helpful tip that appears when you place your mouse over an identifier. It can show you potential values or types, any available documentation, return types and definition locations.

Deploying an IPython Notebook to Microsoft Azure

In order to deploy an IPython Notebook to Microsoft Azure, you must first create a VM. It can be either a Windows VM or a Linux VM. In this example, we'll create an Ubuntu 13.04 VM. If you want to learn how to create a Windows VM or an OpenSUSE VM, see the IPython Notebook on Microsoft Azure documentation.

Create a New VM

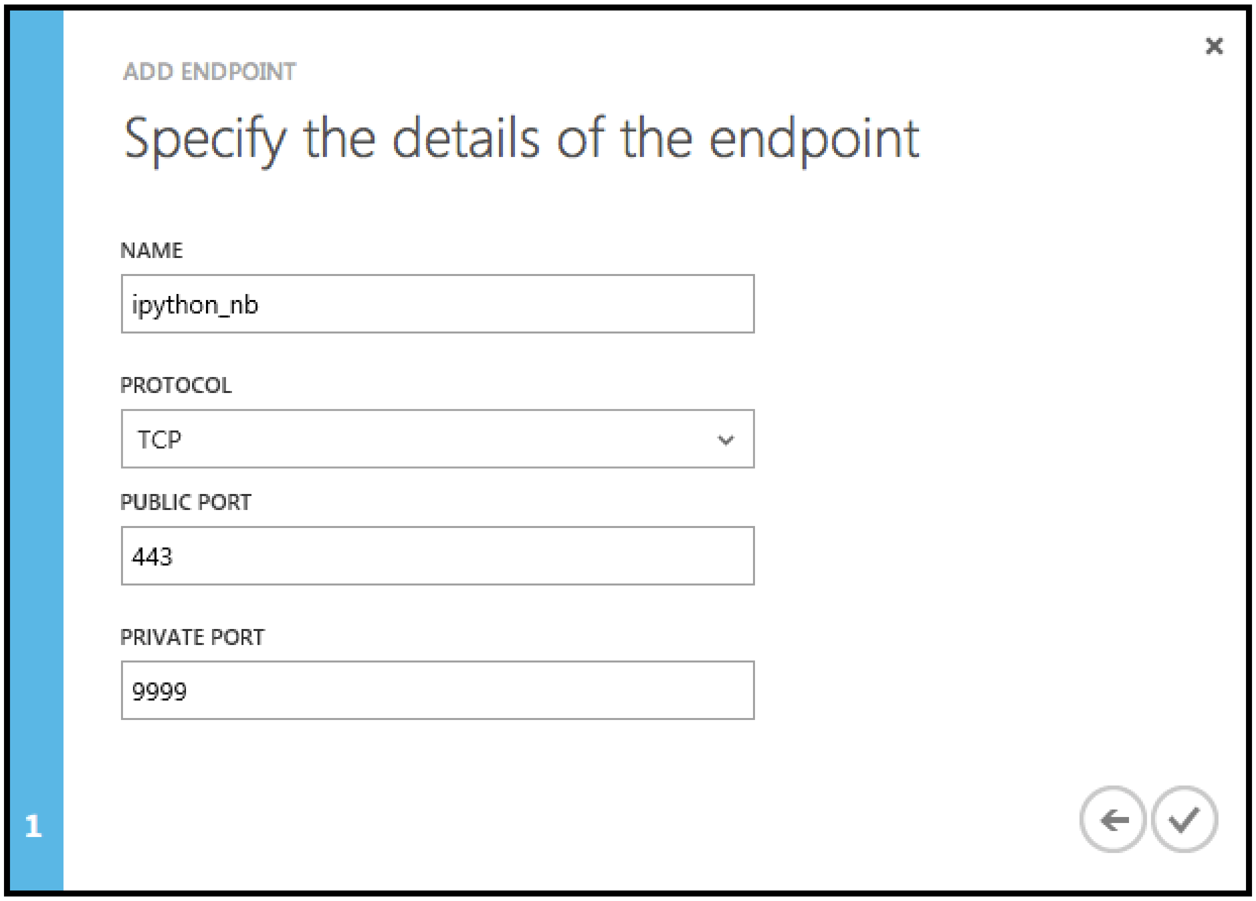
The first thing to do is to create a new VM. For the steps, see Create a Virtual Machine Running Linux. Create an **Ubuntu 13.04** VM and name it **ipython-nb**.

Create an Endpoint for the IPython Notebook

In this example, IPython will run its notebook server on port 9999. To make this port publicly available, you must create an endpoint in the Microsoft Azure Management Portal. This endpoint opens up a port in the Microsoft Azure firewall and maps the public port (HTTPS, 443) to the private port on the VM (9999). Here are the steps to create an endpoint.

1. Go to the Management Portal. In the left-hand column, select **Virtual Machines**.
2. Select the **ipython_nb** VM.
3. On the dashboard, select **Endpoints**.
4. Select **Add Endpoint.**
5. In **NAME**, type **ipython_nb**.
6. In **PROTOCOL**, specify **TCP**.
7. In **PUBLIC PORT**, type **443**.
8. In **PRIVATE PORT**, type **9999**.
9. Select the **check mark**.

The following screen shot shows the completed dialog.



Specify the details of the endpoint

NAME
ipython_nb

PROTOCOL
TCP

PUBLIC PORT
443

PRIVATE PORT
9999

Figure 4

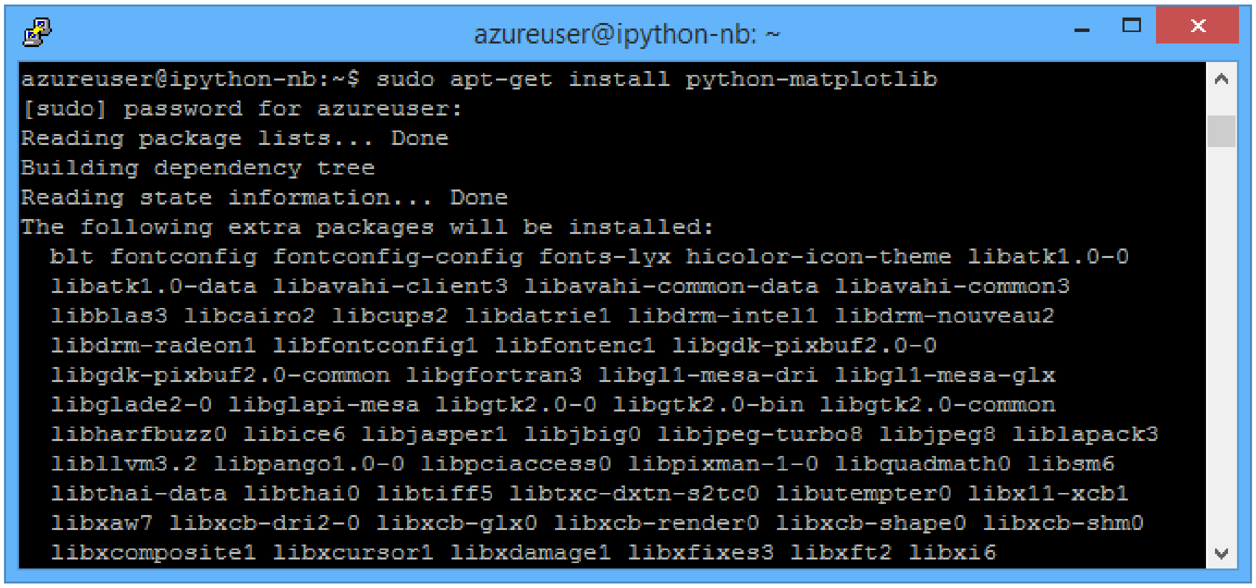
Install the Required Software on the VM

To run the IPython Notebook on the VM, you must first install IPython, its dependencies and any packages that you want to use.

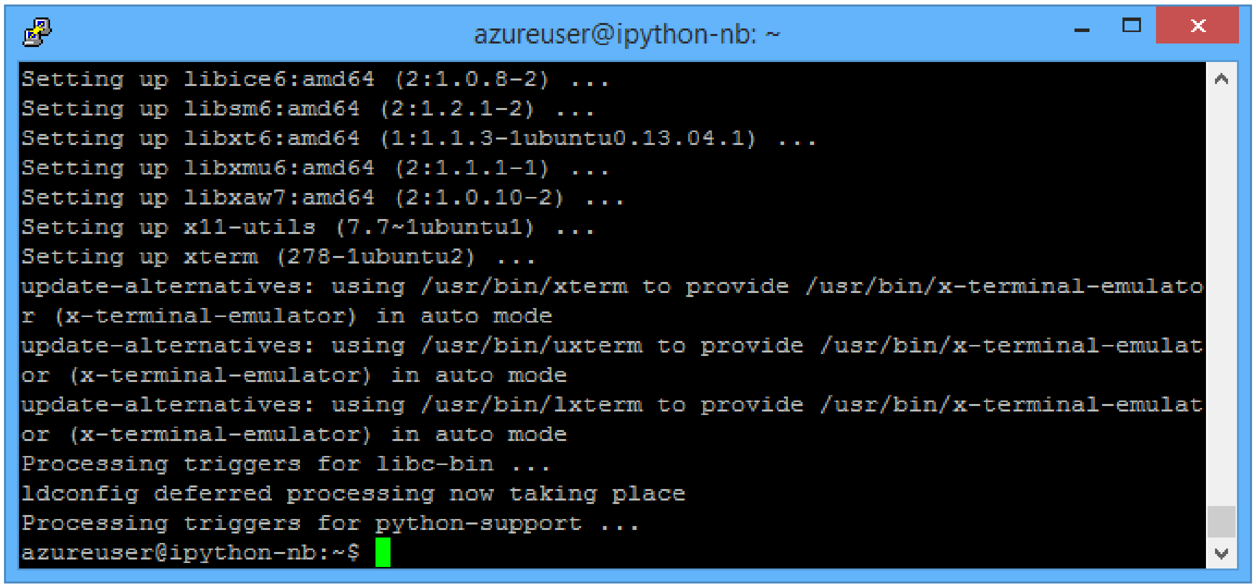
To install the dependencies, SSH into the VM. The following shell commands install Matplotlib, Tornado, and other IPython dependencies into an Ubuntu VM.

```
sudo apt-get install python-matplotlib

sudo apt-get install python-tornado

sudo apt-get install ipython

sudo apt-get install ipython-notebook
```

For example, the following screen shot shows the output for the `sudo apt-get install python-matplotlib command.`



```
azureuser@ipython-nb:~$ sudo apt-get install python-matplotlib
[sudo] password for azureuser:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  blt fontconfig fontconfig-config fonts-lyx hicolor-icon-theme libatk1.0-0
  libatk1.0-data libavahi-client3 libavahi-common-data libavahi-common3
  libblas3 libcairo2 libcups2 libdatrie1 libdrm-intel1 libdrm-nouveau2
  libdrm-radeon1 libfontconfig1 libfontenc1 libgdk-pixbuf2.0-0
  libgdk-pixbuf2.0-common libgfortran3 libgl1-mesa-dri libgl1-mesa-glx
  libglade2-0 libglapi-mesa libgtk2.0-0 libgtk2.0-bin libgtk2.0-common
  libharfbuzz0 libice6 libjasper1 libjbig0 libjpeg-turbo8 libjpeg8 liblapack3
  libllvm3.2 libpango1.0-0 libpciaccess0 libpixman-1-0 libquadmath0 libsm6
  libthai-data libthai0 libtiff5 libtxc-dxtn-s2tc0 libutempter0 libx11-xcb1
  libxaw7 libxcb-dri2-0 libxcb-glx0 libxcb-render0 libxcb-shape0 libxcb-shm0
  libxcomposite1 libxcursor1 libxdamage1 libxfixes3 libxft2 libxi6
```



```
Setting up libice6:amd64 (2:1.0.8-2) ...
Setting up libsm6:amd64 (2:1.2.1-2) ...
Setting up libxt6:amd64 (1:1.1.3-1ubuntu0.13.04.1) ...
Setting up libxmu6:amd64 (2:1.1.1-1) ...
Setting up libxaw7:amd64 (2:1.0.10-2) ...
Setting up x11-utils (7.7~1ubuntu1) ...
Setting up xterm (278-1ubuntu2) ...
update-alternatives: using /usr/bin/xterm to provide /usr/bin/x-terminal-emulato
r (x-terminal-emulator) in auto mode
update-alternatives: using /usr/bin/uxterm to provide /usr/bin/x-terminal-emulat
or (x-terminal-emulator) in auto mode
update-alternatives: using /usr/bin/lxterm to provide /usr/bin/x-terminal-emulat
or (x-terminal-emulator) in auto mode
Processing triggers for libc-bin ...
ldconfig deferred processing now taking place
Processing triggers for python-support ...
azureuser@ipython-nb:~$
```

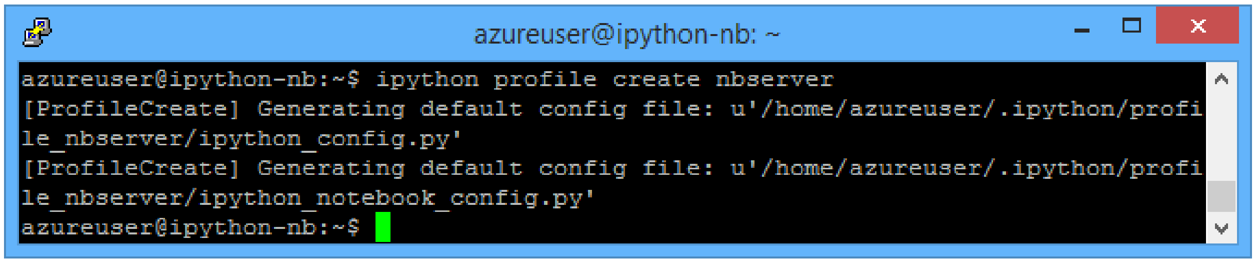
Figure 5

Configure the IPython Notebook

You must configure the IPython Notebook. First, use the following shell command to create a configuration profile.

```
ipython profile create nbserver
```

Here is the output.



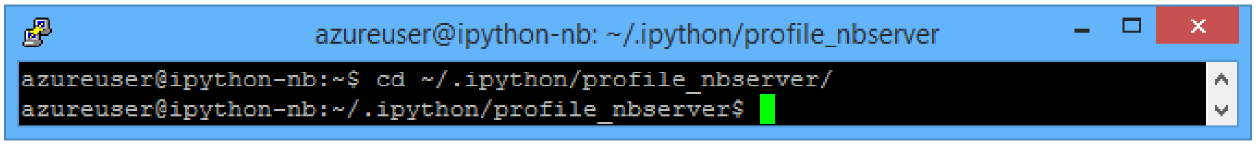
```
azureuser@ipython-nb:~$ ipython profile create nbserver
[ProfileCreate] Generating default config file: u'/home/azureuser/.ipython/profi
le_nbserver/ipython_config.py'
[ProfileCreate] Generating default config file: u'/home/azureuser/.ipython/profi
le_nbserver/ipython_notebook_config.py'
azureuser@ipython-nb:~$
```

Figure 6

Go to the profile directory to create an SSL certificate and to edit the profiles configuration file. Here is the shell command.

```
cd ~/.ipython/profile_nbserver/
```

Here is the output.



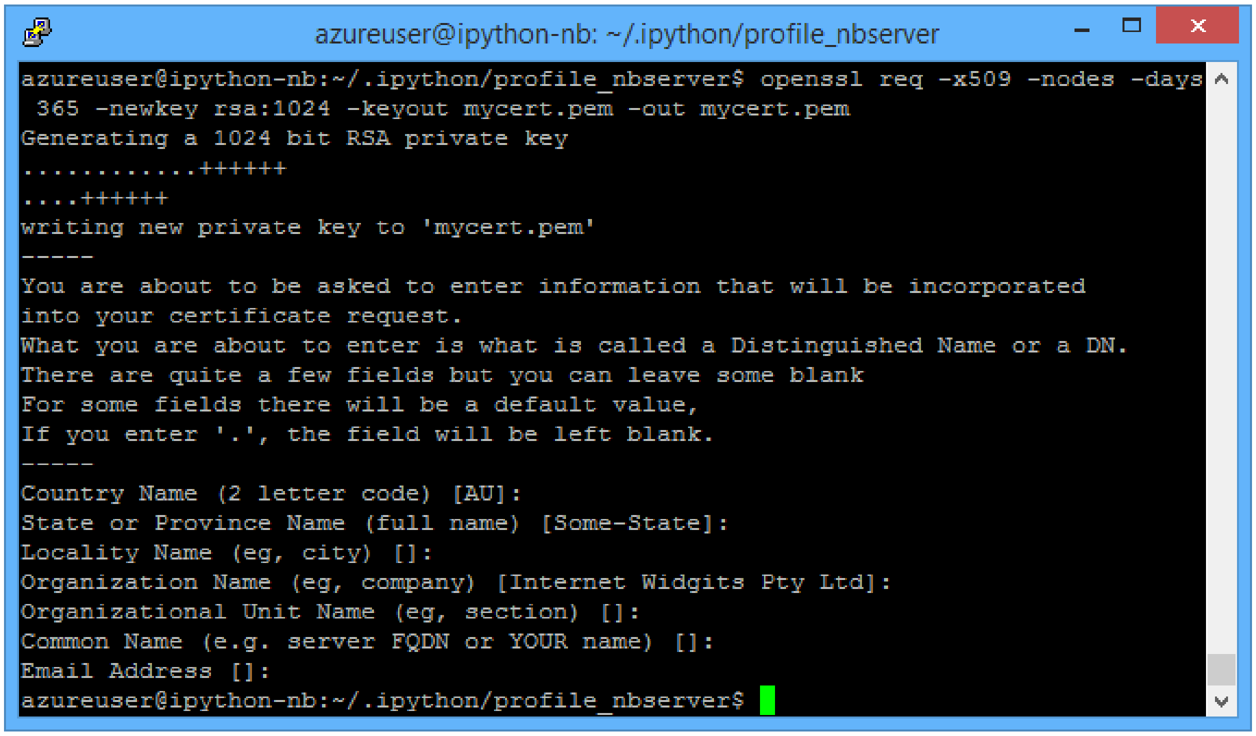
```
azureuser@ipython-nb:~$ cd ~/.ipython/profile_nbserver/
azureuser@ipython-nb:~/.ipython/profile_nbserver$
```

Figure 7

Create the SSL certificate using the following shell command.

```
openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mycert.pem -out
mycert.pem
```

Here is the output.



```
azureuser@ipython-nb:~/.ipython/profile_nbserver$ openssl req -x509 -nodes -days
 365 -newkey rsa:1024 -keyout mycert.pem -out mycert.pem
Generating a 1024 bit RSA private key
............++++++
....++++++
writing new private key to 'mycert.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]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
azureuser@ipython-nb:~/.ipython/profile_nbserver$
```

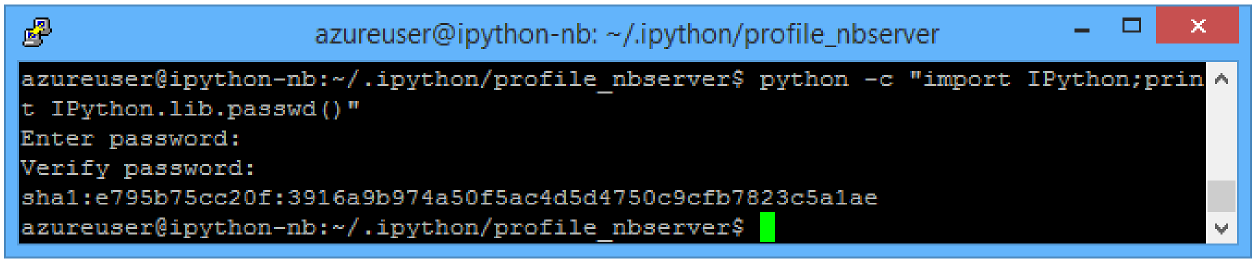
Figure 8

In addition to using a certificate, you must also provide a password to protect your notebook from unauthorized use. IPython uses encrypted passwords in its configuration file, so you'll need to encrypt your password first. IPython provides a utility for this. Here is the shell command.

```
python -c "import IPython;print IPython.lib.passwd()"
```

You will be prompted for a password and confirmation. The output shows you the encrypted password. Here is an example.

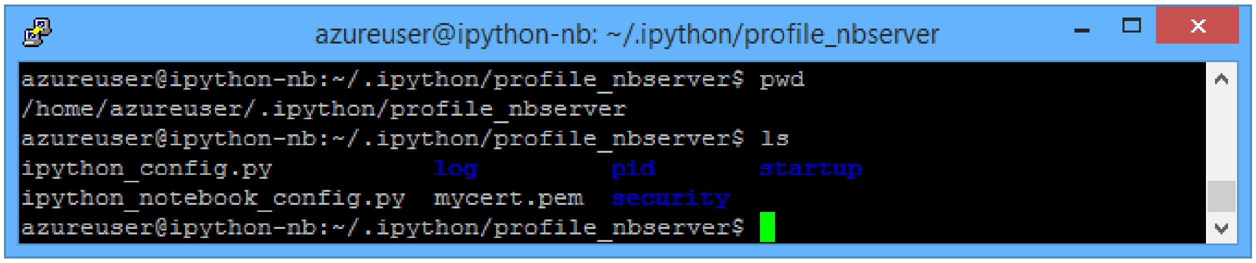
```
Enter password:

Verify password:

sha1:b86e933199ad:a02e9592e59723da722…
```



```
azureuser@ipython-nb:~/.ipython/profile_nbserver$ python -c "import IPython;prin
t IPython.lib.passwd()"
Enter password:
Verify password:
sha1:e795b75cc20f:3916a9b974a50f5ac4d5d4750c9cfb7823c5a1ae
azureuser@ipython-nb:~/.ipython/profile_nbserver$
```

Figure 9

Here’s the contents of the current directory.



```
azureuser@ipython-nb:~/.ipython/profile_nbserver$ pwd
/home/azureuser/.ipython/profile_nbserver
azureuser@ipython-nb:~/.ipython/profile_nbserver$ ls
ipython_config.py           log         pid        startup
ipython_notebook_config.py  mycert.pem  security
azureuser@ipython-nb:~/.ipython/profile_nbserver$
```

Figure 10

Next, edit the profile's configuration file, which is the `ipython_notebook_config.py` file in the profile directory you are currently in. This file has a number of fields and by default all are commented out. You can open this file with any text editor you like. The file should contain at least the following content once you've finished editing it.

```
c = get_config()

# This starts plotting support always with matplotlib
c.IPKernelApp.pylab = 'inline'

# You must give the path to the certificate file.

# If using a Linux VM (Ubuntu):
c.NotebookApp.certfile = u'/home/azureuser/.ipython/profile_nbserver/mycert.pem'

# Create your own password as indicated above
c.NotebookApp.password = u'sha1:b86e933199ad:a02e9592e5 etc... '

# Network and browser details. We use a fixed port (9999) so it matches
# our Microsoft Azure setup, where we've allowed traffic on that port

c.NotebookApp.ip = '*'
c.NotebookApp.port = 9999
c.NotebookApp.open_browser = False
```

Run an IPython Notebook

You're now ready to run a notebook. Navigate to the directory where you want to store your notebooks. For example, you can use the following shell commands.

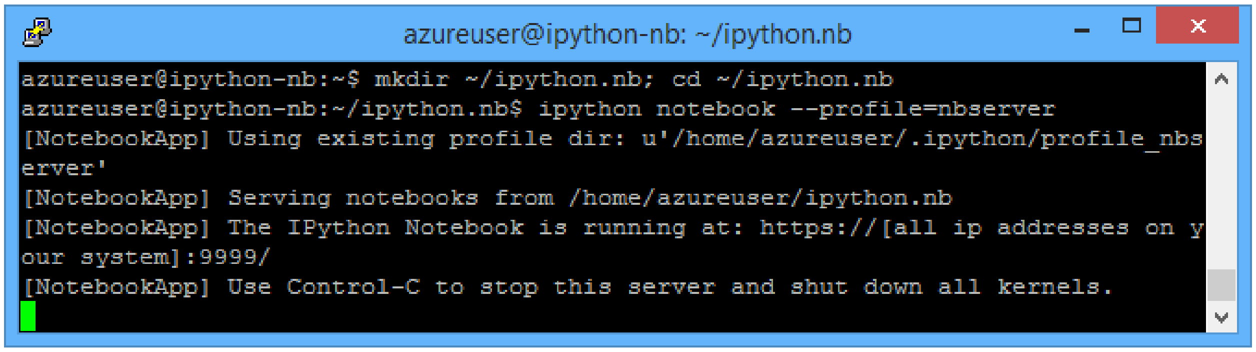
```
mkdir ~/ipython.nb
```

```
cd ~/ipython.nb
```

Next, start the IPython Notebook server with the following shell command.

```
ipython notebook --profile=nbserver
```

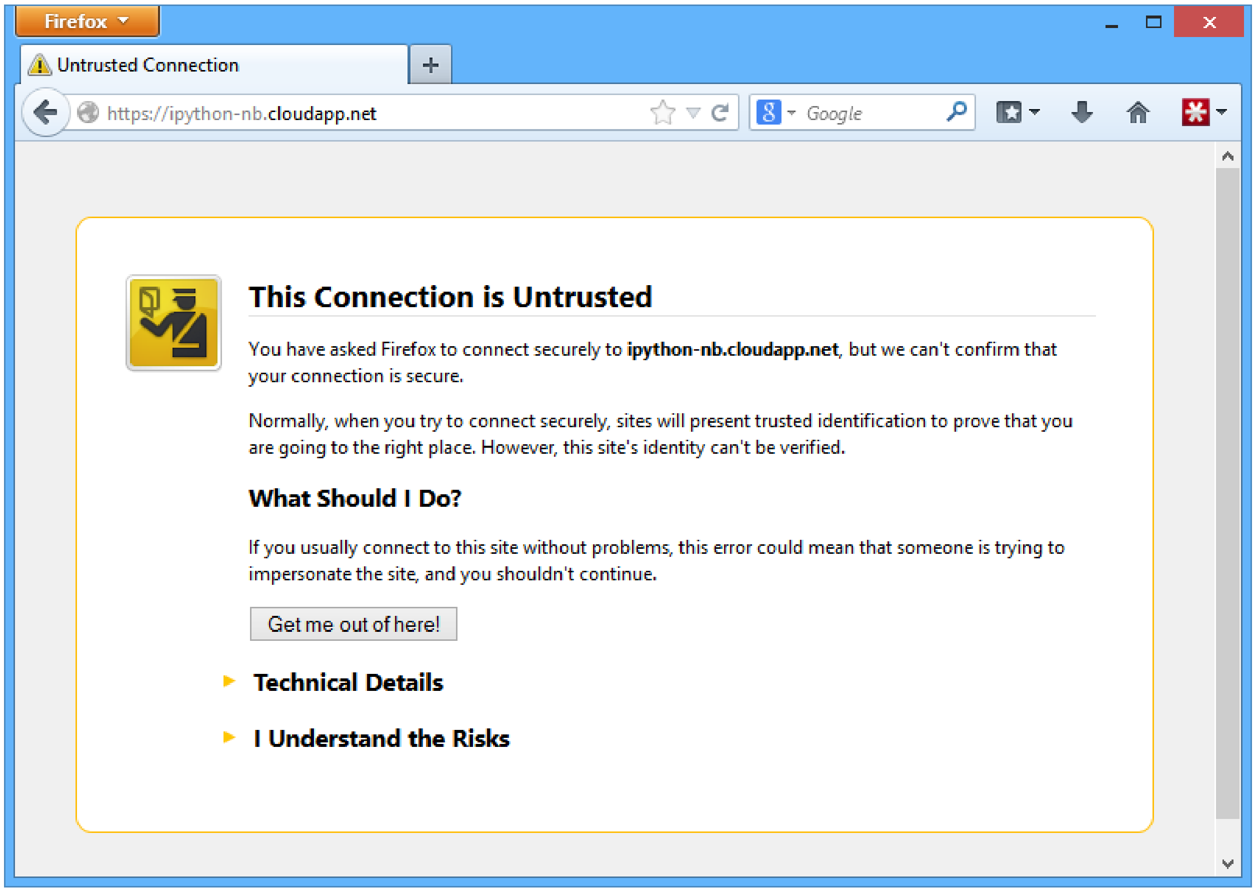
Here is the output.



```
azureuser@ipython-nb:~$ mkdir ~/ipython.nb; cd ~/ipython.nb
azureuser@ipython-nb:~/ipython.nb$ ipython notebook --profile=nbserver
[NotebookApp] Using existing profile dir: u'/home/azureuser/.ipython/profile_nbs
erver'
[NotebookApp] Serving notebooks from /home/azureuser/ipython.nb
[NotebookApp] The IPython Notebook is running at: https://[all ip addresses on y
our system]:9999/
[NotebookApp] Use Control-C to stop this server and shut down all kernels.
```

Figure 11

You should now be able to access an IPython Notebook at the address https://[*Your Chosen Name Here*].cloudapp.net.



This Connection is Untrusted

You have asked Firefox to connect securely to **ipython-nb.cloudapp.net**, but we can't confirm that your connection is secure.

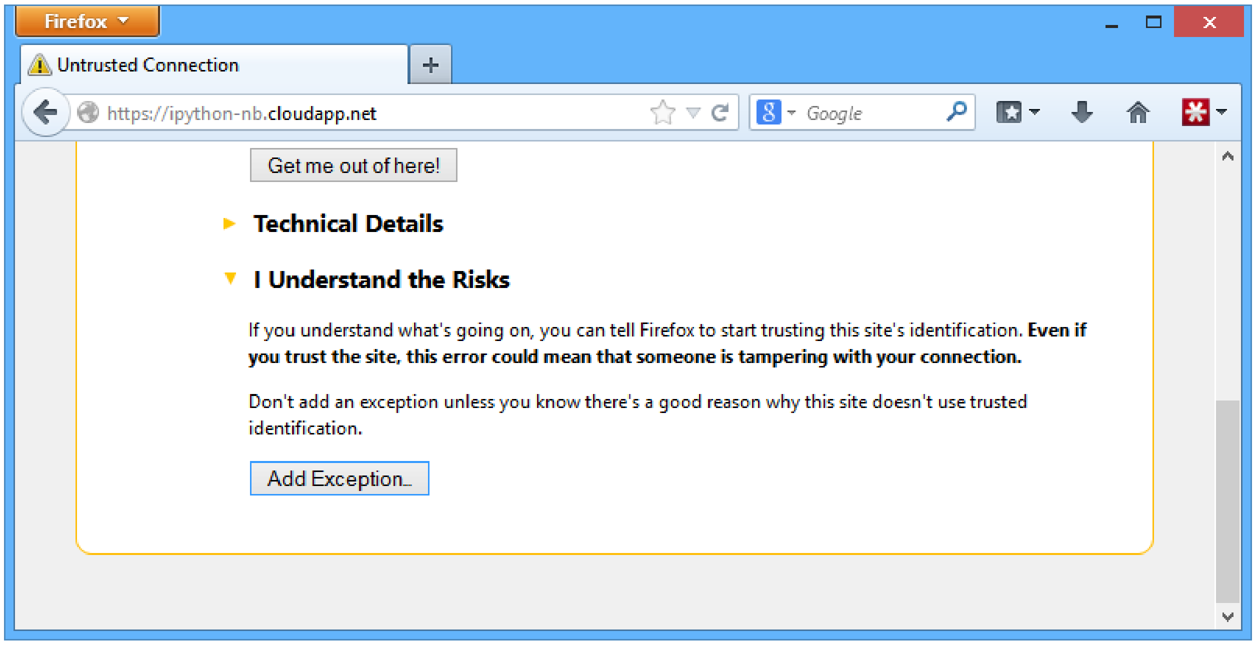
Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

Figure 12

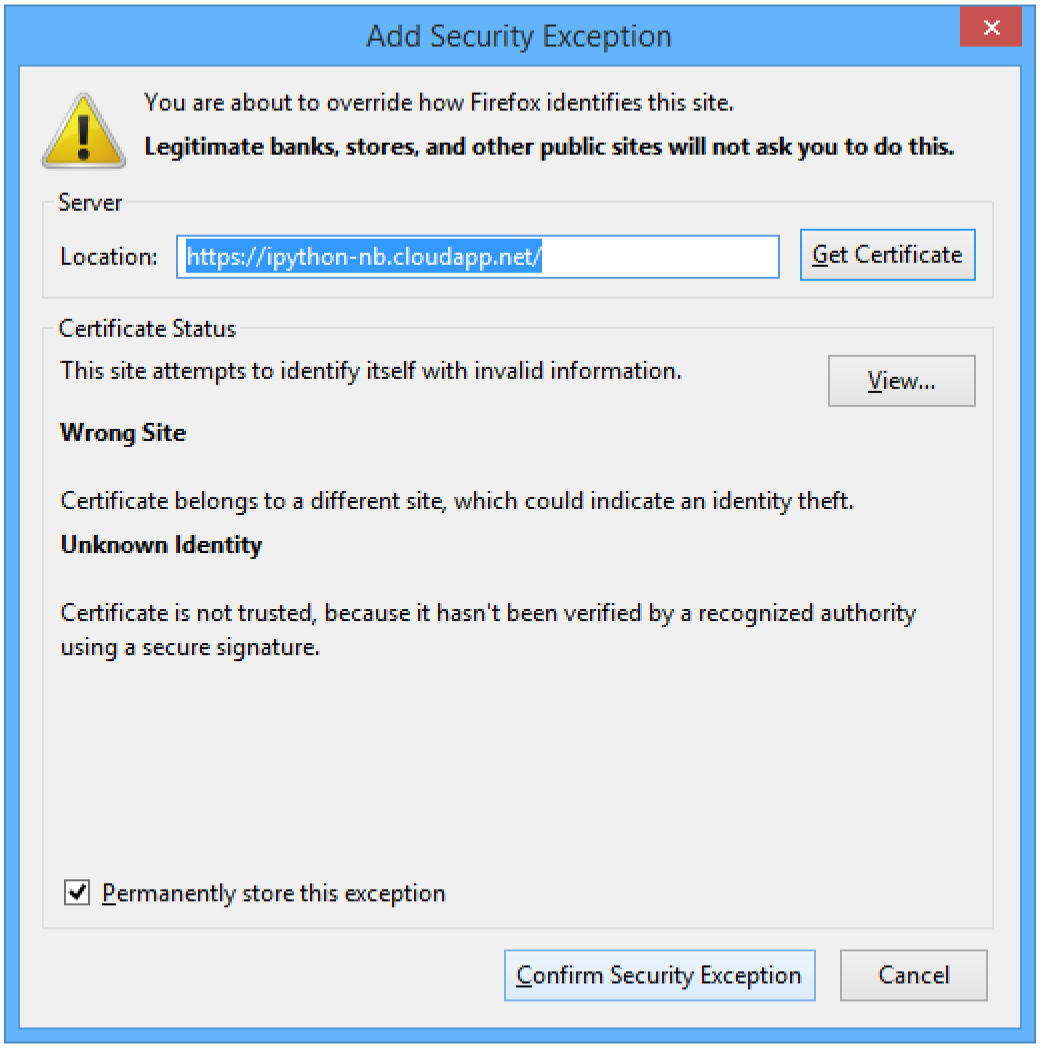
EXPAND “I Understand the Risks” and proceed to add an exception so that we won’t get further complaints about this being a self-signed certificate.



If you understand what's going on, you can tell Firefox to start trusting this site's identification. **Even if you trust the site, this error could mean that someone is tampering with your connection.**

Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.

Figure 13



You are about to override how Firefox identifies this site.

Legitimate banks, stores, and other public sites will not ask you to do this.

Server

Location: https://ipython-nb.cloudapp.net/

Certificate Status

This site attempts to identify itself with invalid information.

Wrong Site

Certificate belongs to a different site, which could indicate an identity theft.

Unknown Identity

Certificate is not trusted, because it hasn't been verified by a recognized authority using a secure signature.

[✓] Permanently store this exception

Figure 14

When you first access your notebook, the logon page asks for your password. The following screen shot shows an example of the logon page.

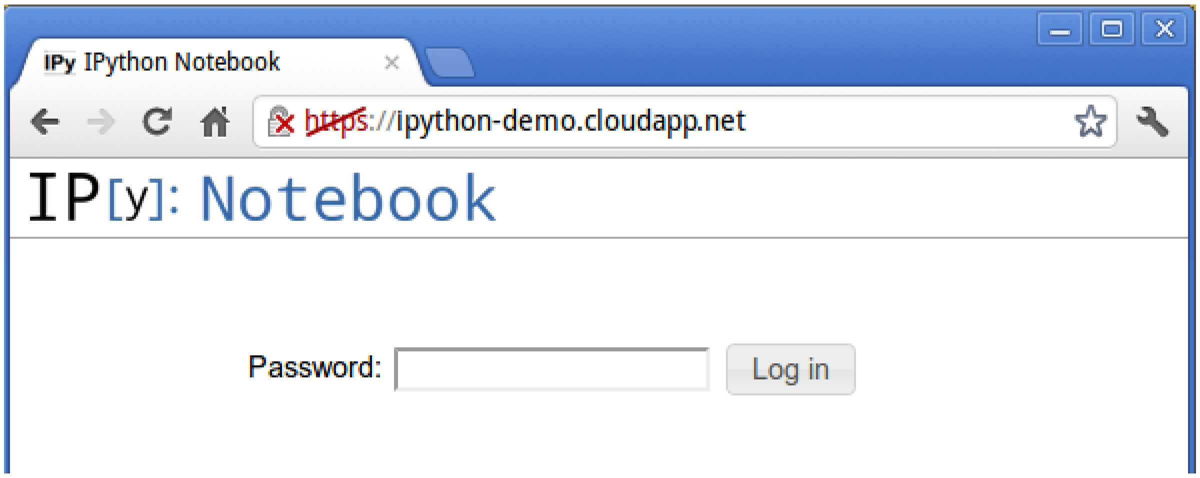


Figure 15

After you log on, you will see the IPython Notebook Dashboard, which is the control center for all notebook operations. From this page you can perform tasks such as create new notebooks and open existing ones. The following screen shot is an example of the dashboard.

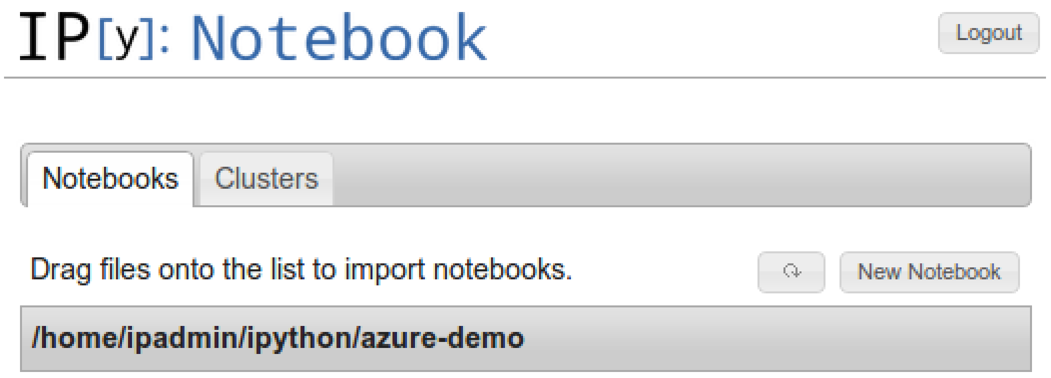

IP[y]: Notebook
Logout
Notebooks
Clusters
Drag files onto the list to import notebooks.
New Notebook
/home/ipadmin/ipython/azure-demo

Figure 16

You now have your IP Notebook set up and are ready to use it.

Using the Elastacloud VM

Instead of creating your own IPython Notebook, you may want to use the Elastacloud Azure Data Analysis VM that is tailored for users who want to perform data analysis on Microsoft Azure. The VM is available on VM Depot, which is a community-driven catalog of VM images that can easily be deployed to Microsoft Azure. The Azure Data Analysis VM includes useful packages such as R, Open MPI, Spark 0.8, Shark 0.7 and Storm and Kafka.

Using Python to Create VMs

Microsoft Azure has a service management API that allows you to programmatically perform many of the same tasks that you can do through the Management Portal. By using the Microsoft Azure SDK for Python you can, for example, manage your cloud services, storage accounts, and VMs. To give you a sense of how to use Python with the service management API, this section shows you how to connect to the service management endpoint and how to create a VM. For complete information, see How to Use Service Management from Python.

Connecting to the Service Management Endpoint

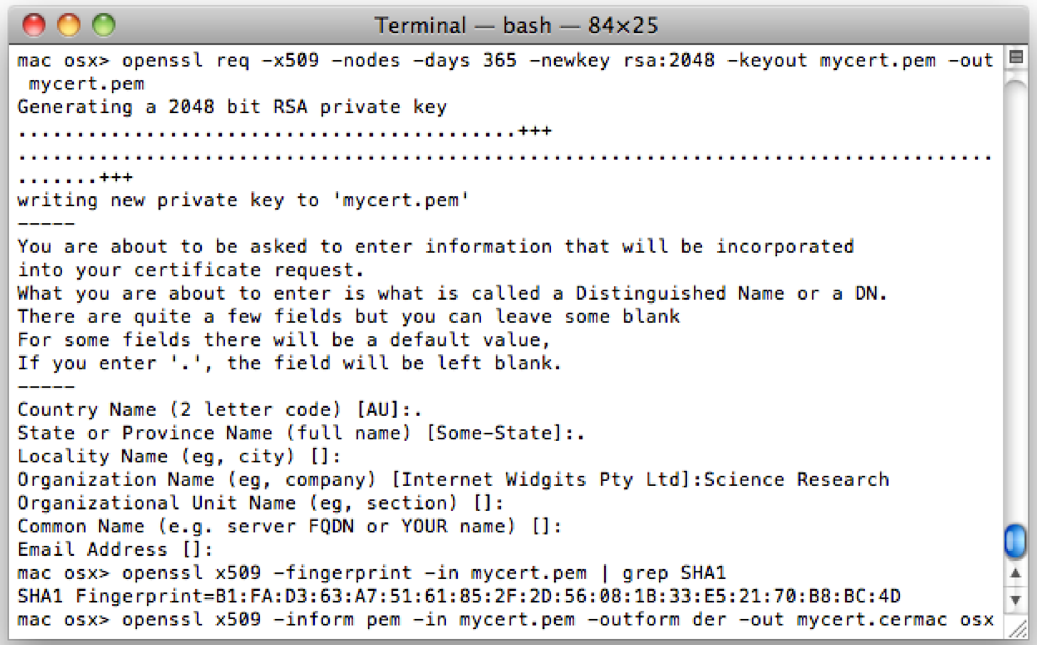
To connect to the service management endpoint, you need your Microsoft Azure subscription ID and the path to a valid management certificate. You can obtain your subscription ID through the management portal, and you can create management certificates in a number of ways. This example uses OpenSSL.

You need to create two certificates, one for the server (a .cer file) and one for the client (a .pem file). Here is the shell command to create the .pem file. The certificate's common name is displayed by the Microsoft Azure Portal as the name of the certificate. You can use any name here that will help you identify the certificate. The common name can either be specified with the parameter *-subj "/CN=My Cert Name"* or interactively (along with other information, if desired), as shown in the screen shot.

```
openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mycert.pem -out
mycert.pem -subj "/CN=My Cert Name"
```

Here is the shell command to create the .cer file.

```
openssl x509 -inform pem -in mycert.pem -outform der -out mycert.cer
```

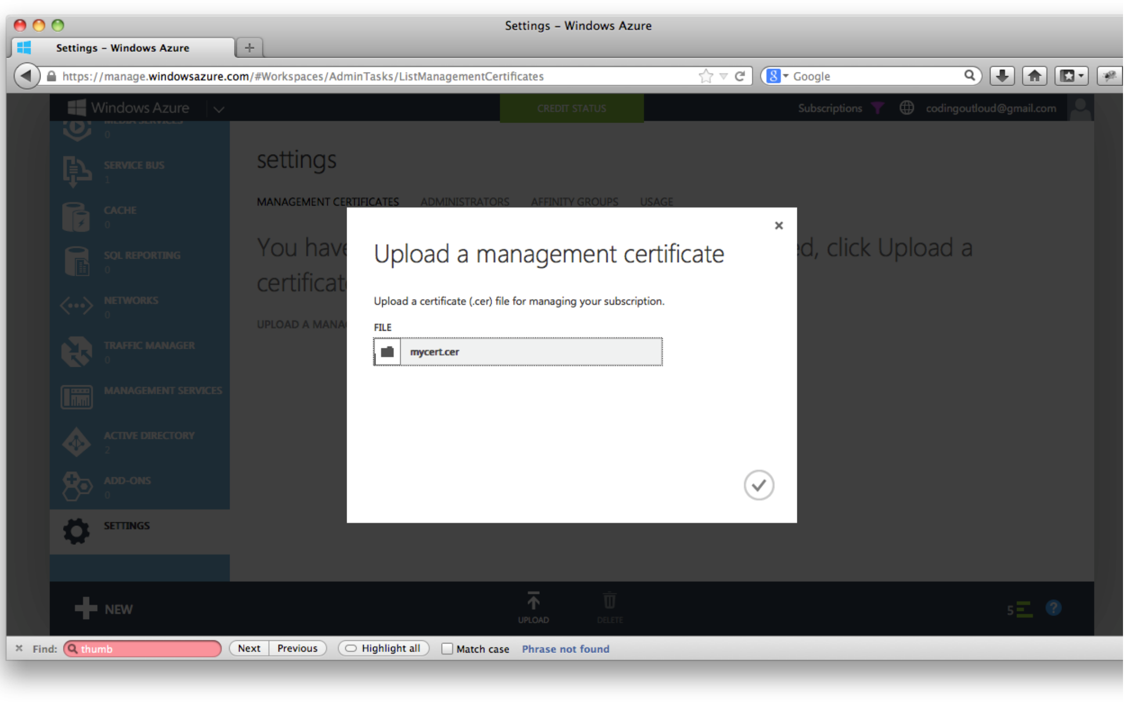


```
mac osx> openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout mycert.pem -out
 mycert.pem
Generating a 2048 bit RSA private key
...........................................+++
....................................................................................
.......+++
writing new private key to 'mycert.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]:.
State or Province Name (full name) [Some-State]:.
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Science Research
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
mac osx> openssl x509 -fingerprint -in mycert.pem | grep SHA1
SHA1 Fingerprint=B1:FA:D3:63:A7:51:61:85:2F:2D:56:08:1B:33:E5:21:70:B8:BC:4D
mac osx> openssl x509 -inform pem -in mycert.pem -outform der -out mycert.cermac osx
```

Figure 17

Upload the .cer File

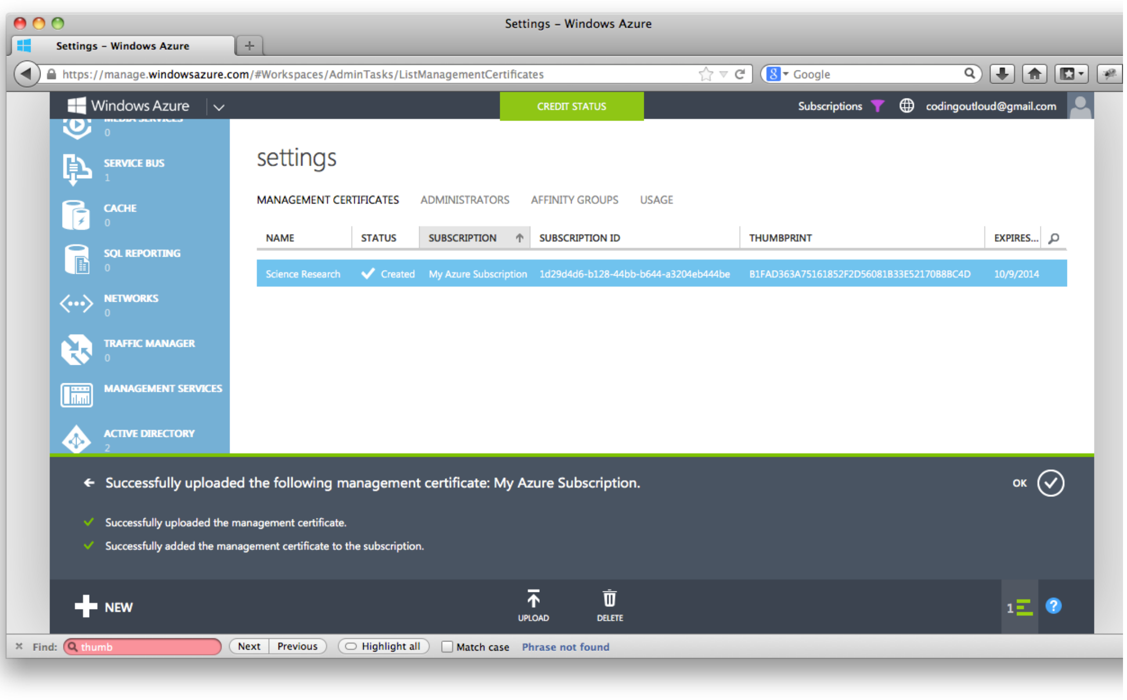
You need to upload the .cer file to Microsoft Azure by selecting **Upload** on the **Settings** tab of the Management Portal. Make a note of where you saved the .cer file. The following screen shots show you an example.


Upload a management certificate

Upload a certificate (.cer) file for managing your subscription.

FILE

mycert.cer


settings

MANAGEMENT CERTIFICATES ADMINISTRATORS AFFINITY GROUPS USAGE

NAME	STATUS	SUBSCRIPTION	SUBSCRIPTION ID	THUMBPRINT	EXPIRES...
Science Research	✓ Created	My Azure Subscription	1d29d4d6-b128-44bb-b644-a3204eb444be	B1FAD363A75161852F2D56081B33E52170B8BC4D	10/9/2014

Successfully uploaded the following management certificate: My Azure Subscription.

✓ Successfully uploaded the management certificate.

✓ Successfully added the management certificate to the subscription.

Figure 18

After you have obtained your subscription ID, created a certificate, and uploaded the .cer file to Microsoft Azure, you can connect to the Microsoft Azure service management endpoint by passing the subscription id and the certificate location to **ServiceManagementService**. The location can be a certificate store path on Microsoft Azure or a local path to a .pem file. Here is the Python code.

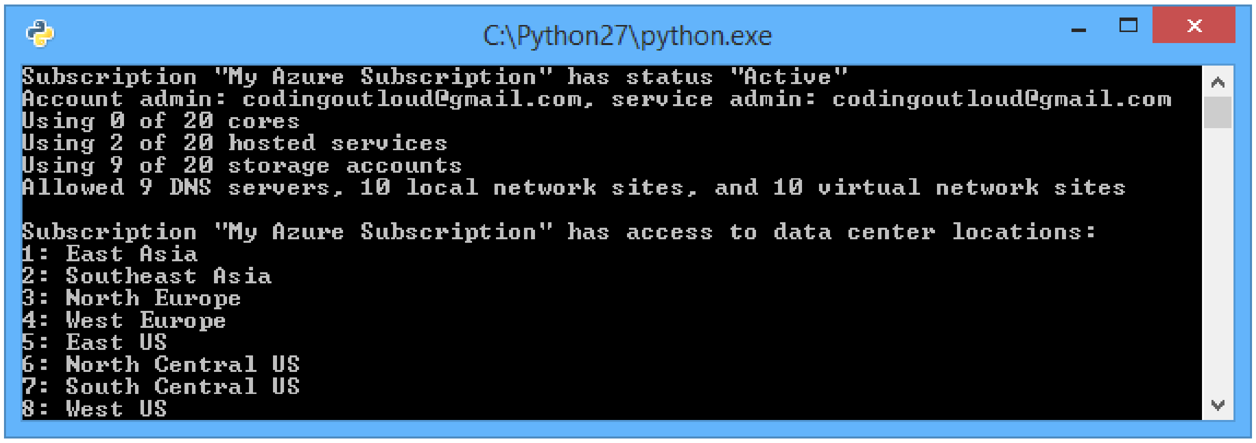
```
from azure.servicemanagement import *

subscription_id = '<your_subscription_id>' # e.g., '1d29 …4fe'
# If running on Windows, note that for historical reasons, this
# value is different. Rather than point to a .pem on Windows, it points
# to a location in the certificate store, e.g., 'CURRENT_USER\\my\\LabX'.
# On non-Windows platforms, it is a path to a .pem file, e.g., './foo.pem'.
certificate_path = './mycert.pem'   # Linux, Mac
certificate_path = 'CURRENT_USER\\my\\LabX' # Windows only

sms = ServiceManagementService(subscription_id, certificate_path)
```

You are now ready to use the service management API.

Interrogating Subscription Information

You can use the service management API to programmatically create and release resources (such as Cloud Services, Virtual Networks, and VMs) as well as manipulate and examine your cloud and subscription environment. The following code shows how the service management API can be used to display information about a subscription, including its quotas, and current usage. This code uses the **sms** variable defined above.

```
sub_meta = sms.get_subscription();
print('Subscription "%s" has status "%s"' % (sub_meta.subscription_name,
       sub_meta.subscription_status))
print('Account admin: %s, service admin: %s' % (sub_meta.account_admin_live_email_id,
       sub_meta.service_admin_live_email_id))
print('Using %s of %s cores' % (sub_meta.current_core_count, sub_meta.max_core_count))
print('Using %s of %s hosted services' % (sub_meta.current_hosted_services,
       sub_meta.max_hosted_services))
print('Using %s of %s storage accounts' % (sub_meta.current_storage_accounts,
       sub_meta.max_storage_accounts))
print('Allowed %s DNS servers, %s local network sites, and %s virtual network sites' %
       (sub_meta.max_dns_servers, sub_meta.max_local_network_sites,
sub_meta.max_virtual_network_sites))
print
print('Subscription "%s" has access to data center locations:' %
sub_meta.subscription_name)
result = sms.list_locations()
i = 1
for location in result:
    print('%s: %s' % (i, location.name))
    i = i + 1
```



```
Subscription "My Azure Subscription" has status "Active"
Account admin: codingoutloud@gmail.com, service admin: codingoutloud@gmail.com
Using 0 of 20 cores
Using 2 of 20 hosted services
Using 9 of 20 storage accounts
Allowed 9 DNS servers, 10 local network sites, and 10 virtual network sites

Subscription "My Azure Subscription" has access to data center locations:
1: East Asia
2: Southeast Asia
3: North Europe
4: West Europe
5: East US
6: North Central US
7: South Central US
8: West US
```

Figure 19

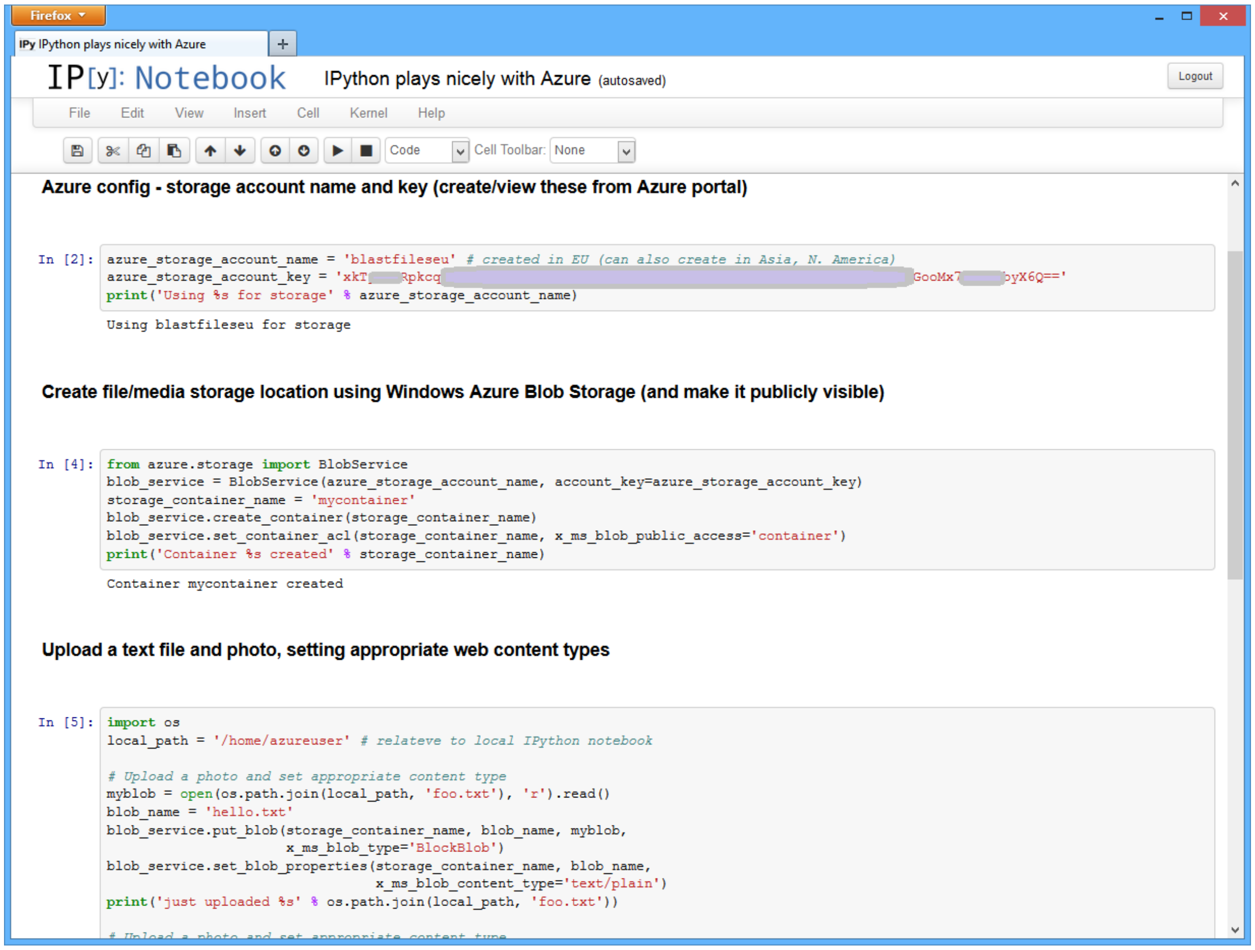
Note: Currently VM Depot does not support automatically cloning an image and copying it to your local account.

More Information

Typically, you use automation to create VMs when you need to create a cluster of VMs. (If you only need a single VM, than using the Microsoft Azure Management Portal is simpler.) For more information about creating Linux VM clusters and using IPython to run simulations, see the hands-on-lab named *Creating a Linux virtual machine cluster and running simulation analysis with IPython Cluster*.

Using Python to Create an Application

For a complete example of how to use Python to create an application on Microsoft Azure, see “Using Microsoft Azure and Python for BLAST.” Here is a short excerpt from the paper that shows you how to use blob storage with Python. Figure 20 shows the code within a Python Notebook.


Firefox
IPy IPython plays nicely with Azure
IP[y]: Notebook
IPython plays nicely with Azure (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Code
Cell Toolbar: None

Azure config - storage account name and key (create/view these from Azure portal)

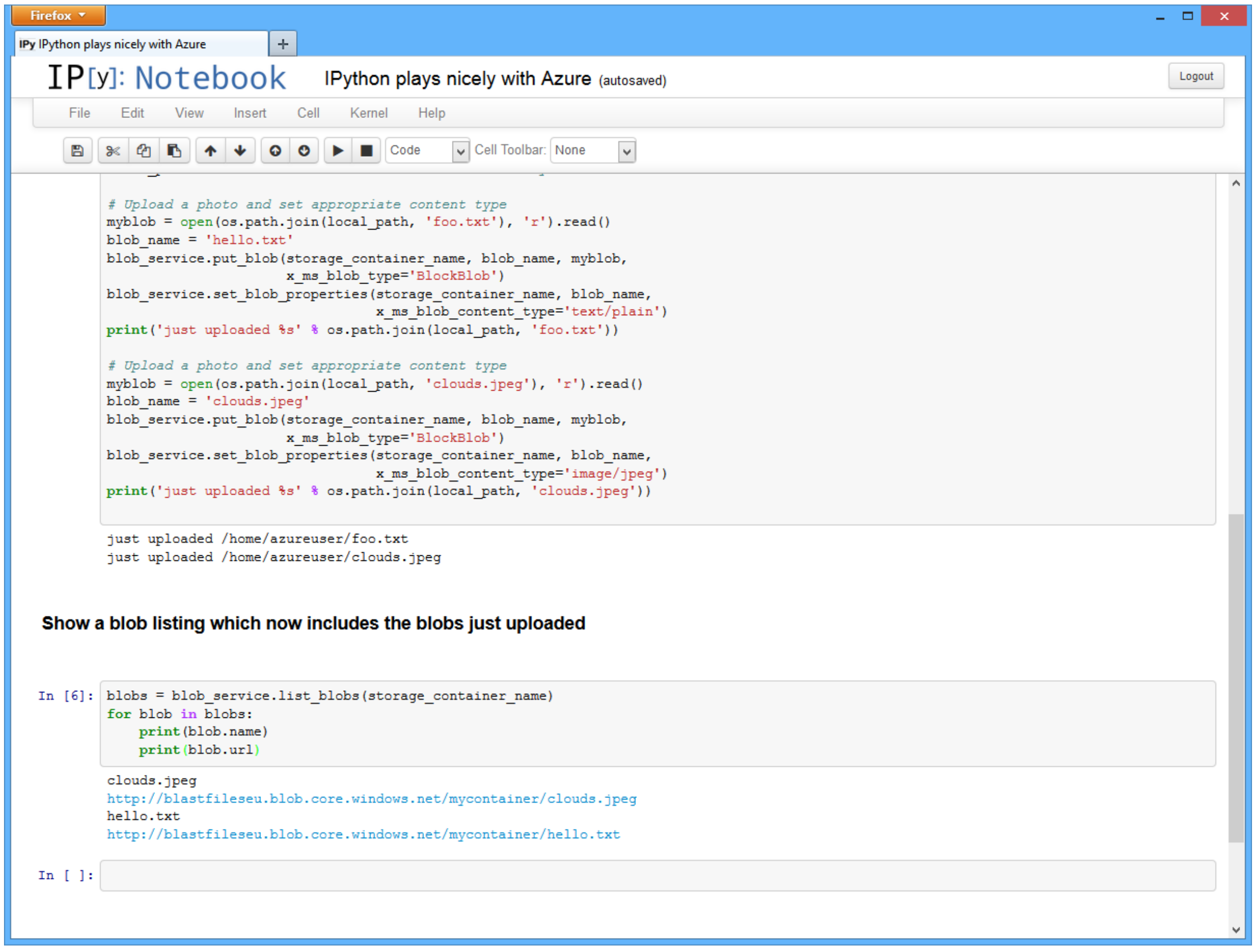
```
In [2]: azure_storage_account_name = 'blastfileseu' # created in EU (can also create in Asia, N. America)
        azure_storage_account_key = 'xkT[redacted]Rpkcq[redacted]GooMx7[redacted]byX6Q=='
        print('Using %s for storage' % azure_storage_account_name)

        Using blastfileseu for storage
```

Create file/media storage location using Windows Azure Blob Storage (and make it publicly visible)

```
In [4]: from azure.storage import BlobService
        blob_service = BlobService(azure_storage_account_name, account_key=azure_storage_account_key)
        storage_container_name = 'mycontainer'
        blob_service.create_container(storage_container_name)
        blob_service.set_container_acl(storage_container_name, x_ms_blob_public_access='container')
        print('Container %s created' % storage_container_name)

        Container mycontainer created
```

Upload a text file and photo, setting appropriate web content types

```
In [5]: import os
        local_path = '/home/azureuser' # relateve to local IPython notebook

        # Upload a photo and set appropriate content type
        myblob = open(os.path.join(local_path, 'foo.txt'), 'r').read()
        blob_name = 'hello.txt'
        blob_service.put_blob(storage_container_name, blob_name, myblob,
                              x_ms_blob_type='BlockBlob')
        blob_service.set_blob_properties(storage_container_name, blob_name,
                                         x_ms_blob_content_type='text/plain')
        print('just uploaded %s' % os.path.join(local_path, 'foo.txt'))

        # Upload a photo and set appropriate content type
```


Firefox
IPython plays nicely with Azure
IP[y]: Notebook
IPython plays nicely with Azure (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Code
Cell Toolbar: None

```
# Upload a photo and set appropriate content type
myblob = open(os.path.join(local_path, 'foo.txt'), 'r').read()
blob_name = 'hello.txt'
blob_service.put_blob(storage_container_name, blob_name, myblob,
                      x_ms_blob_type='BlockBlob')
blob_service.set_blob_properties(storage_container_name, blob_name,
                                 x_ms_blob_content_type='text/plain')
print('just uploaded %s' % os.path.join(local_path, 'foo.txt'))

# Upload a photo and set appropriate content type
myblob = open(os.path.join(local_path, 'clouds.jpeg'), 'r').read()
blob_name = 'clouds.jpeg'
blob_service.put_blob(storage_container_name, blob_name, myblob,
                      x_ms_blob_type='BlockBlob')
blob_service.set_blob_properties(storage_container_name, blob_name,
                                 x_ms_blob_content_type='image/jpeg')
print('just uploaded %s' % os.path.join(local_path, 'clouds.jpeg'))
```

```
just uploaded /home/azureuser/foo.txt
just uploaded /home/azureuser/clouds.jpeg
```

Show a blob listing which now includes the blobs just uploaded

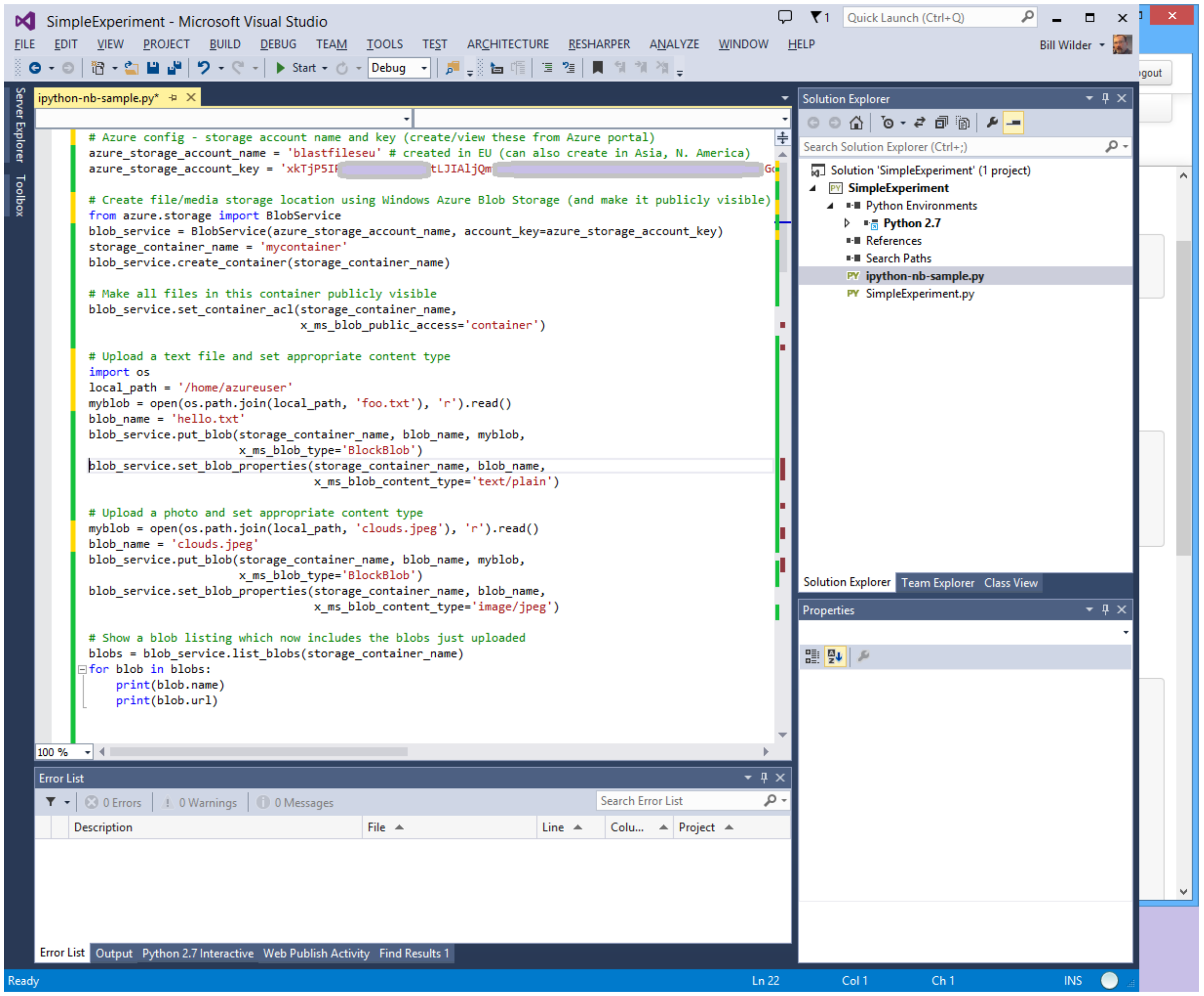
```
In [6]: blobs = blob_service.list_blobs(storage_container_name)
        for blob in blobs:
            print(blob.name)
            print(blob.url)
```

```
clouds.jpeg
http://blastfileseu.blob.core.windows.net/mycontainer/clouds.jpeg
hello.txt
http://blastfileseu.blob.core.windows.net/mycontainer/hello.txt
```

```
In [ ]:
```

Figure 20

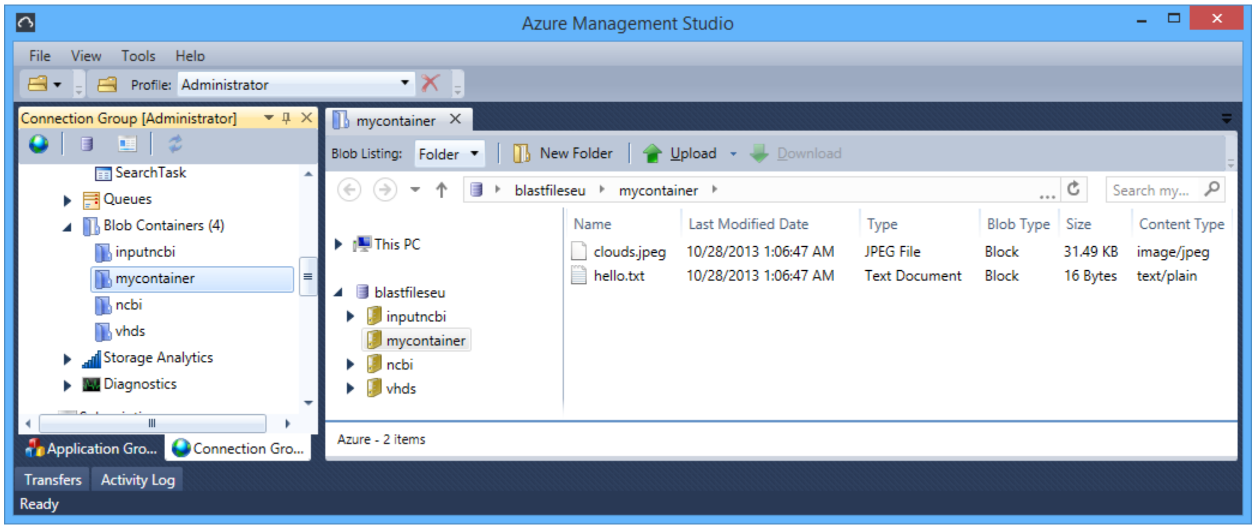
The next screen shot, Figure 21, shows the code within PTVS.



```
# Azure config - storage account name and key (create/view these from Azure portal)
azure_storage_account_name = 'blastfileseu' # created in EU (can also create in Asia, N. America)
azure_storage_account_key = 'xkTjP5IF[redacted]tLJIAljQm[redacted]Gc

# Create file/media storage location using Windows Azure Blob Storage (and make it publicly visible)
from azure.storage import BlobService
blob_service = BlobService(azure_storage_account_name, account_key=azure_storage_account_key)
storage_container_name = 'mycontainer'
blob_service.create_container(storage_container_name)

# Make all files in this container publicly visible
blob_service.set_container_acl(storage_container_name,
                               x_ms_blob_public_access='container')

# Upload a text file and set appropriate content type
import os
local_path = '/home/azureuser'
myblob = open(os.path.join(local_path, 'foo.txt'), 'r').read()
blob_name = 'hello.txt'
blob_service.put_blob(storage_container_name, blob_name, myblob,
                      x_ms_blob_type='BlockBlob')
blob_service.set_blob_properties(storage_container_name, blob_name,
                                 x_ms_blob_content_type='text/plain')

# Upload a photo and set appropriate content type
myblob = open(os.path.join(local_path, 'clouds.jpeg'), 'r').read()
blob_name = 'clouds.jpeg'
blob_service.put_blob(storage_container_name, blob_name, myblob,
                      x_ms_blob_type='BlockBlob')
blob_service.set_blob_properties(storage_container_name, blob_name,
                                 x_ms_blob_content_type='image/jpeg')

# Show a blob listing which now includes the blobs just uploaded
blobs = blob_service.list_blobs(storage_container_name)
for blob in blobs:
    print(blob.name)
    print(blob.url)
```

Figure 21

The following screen shot shows the results of uploading files to a blob container. There are two files in the blob storage container.



Name	Last Modified Date	Type	Blob Type	Size	Content Type
clouds.jpeg	10/28/2013 1:06:47 AM	JPEG File	Block	31.49 KB	image/jpeg
hello.txt	10/28/2013 1:06:47 AM	Text Document	Block	16 Bytes	text/plain

Figure 22

The following Python code example shows how to use blob storage.

```python
from azure.storage import *

# Substitute your own account_name and account account_key
blob_service = BlobService(account_name='myacct', account_key='0eiY2nZG…8M3q4P9s1r7A==')
storage_container_name = 'mycontainer'
blob_service.create_container(storage_container_name)

# Make all files in this container (like a directory or folder) publicly visible
blob_service.set_container_acl(storage_container_name,
                               x_ms_blob_public_access='container')

# Upload a text file and set appropriate content type
myblob = open(r'foo.txt', 'r').read()
blob_name = 'hello.txt'
blob_service.put_blob(storage_container_name, blob_name, myblob,
                      x_ms_blob_type='BlockBlob')
blob_service.set_blob_properties(storage_container_name, blob_name,
                                 x_ms_blob_content_type='text/plain')

# Upload a photo and set appropriate content type
myblob = open(r'clouds.jpeg', 'r').read()
blob_name = 'clouds.jpeg'
blob_service.put_blob(storage_container_name, blob_name, myblob,
                      x_ms_blob_type='BlockBlob')
blob_service.set_blob_properties(storage_container_name, blob_name,
                                 x_ms_blob_content_type='image/jpeg')

# Show a blob listing which now includes the blobs just uploaded
blobs = blob_service.list_blobs(storage_container_name)
for blob in blobs:
    print(blob.name)
    print(blob.url)
```

Summary of Websites

In order to use Python with Microsoft Azure, you must install the Python Azure SDK. (http://azure.microsoft.com/en-us/documentation/articles/python-how-to-install/)

For more information about using Python with Microsoft Azure, go to the Python Developer Center. (http://azure.microsoft.com/en-us/develop/python/)

For more information about IPython, see The IPython project website. (http://ipython.org/index.html)

For more information about the IPython Notebook interactive computing environment, see the IPython Notebook page. (http://ipython.org/notebook.html)

To install PTVS, go to the CodePlex Python for Visual Studio page. (https://pytools.codeplex.com/releases)

To download a version of Python, go to the Python download page. (http://www.python.org/download/)

To learn how to use IPython Notebooks on Microsoft Azure, see the IPython Notebook on Microsoft Azure documentation. (http://azure.microsoft.com/en-us/documentation/articles/virtual-machines-python-ipython-notebook/)

To learn how to perform remote debugging for a Python application, go to Cross-platform Remote Debugging. (http://pytools.codeplex.com/wikipage?title=Remote%20Debugging%20for%20Windows%2C%20Linux%20and%20OS%20X)

To learn how to use the service management API from Python, see How to use Service Management from Python. (http://azure.microsoft.com/en-us/documentation/articles/cloud-services-python-how-to-use-service-management/)