

Infrastructure as code on Azure with Puppet & Chef

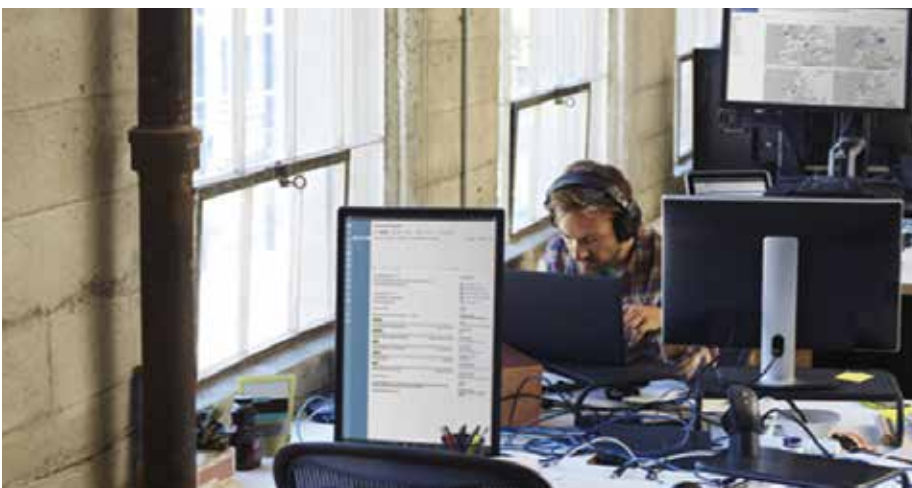


Infrastructure as code on Azure with Puppet & Chef

The concept of infrastructure as code, or programmable infrastructure, plays a significant part in making DevOps possible and is the first step in bringing the development and operations disciplines together within an organization.

As part of the development process, operations works with development from the start of the project – instead of getting involved when the project is nearing deployment – to understand the requirements the application needs to run. The two work together for the duration of the development cycle. These requirements are used to create the automation files that will be used to provision the infrastructure during code deployment through automation tools.

By storing the infrastructure as code in the same versioning system as the code, a record of the changes be tracked, and changes needed to the environment become a matter of executing the automation script. Beyond the increased speed of deployment, infrastructure as code can help prevent configuration drift in an environment. If a change is made to an environment that is not captured in the code, it will be negated at the time of the next deployment.





Microsoft Azure and infrastructure as code

Microsoft Azure is built to be the target of infrastructure as code and includes tools that help facilitate the adoption of the discipline. Every service in Azure is accessible through automation tools via the [Azure Command Line Interface](#) or [Azure PowerShell](#). These connections let the creation of Infrastructure as a Service (IaaS) or Platform as a Service (PaaS) assets be scripted and run for any platform. Beyond the creation of Azure assets, tools like [Azure Automation](#) and [Desired State Configuration](#) allow for deeper integration with services once they are deployed to configure the environment.

Azure is a first class platform for Linux and open source technology, and there is a great story to tell about [support for open source on Azure](#). Nearly one in three Azure virtual machines run Linux, and there is an open source project for [Desired State Configuration for Linux](#).

Azure Resource Manager

While the creation of environments can certainly be programmed through PowerShell and the Command Line Interface, the components of these environments are usually seen as one entity. [Azure Resource Manager](#) allows the deployment and management of these environments as a single entity. Azure Resource Manager [templates](#) are JSON files that define the resources to be deployed together through a declarative syntax. These templates can then be deployed via code. To get started, check out the [Azure Quickstart Templates on GitHub](#).

Chef and Puppet

We know that many organizations have made investments in on-premises hardware, in multiple platforms, and in automation resources like [Chef](#) and [Puppet](#). Azure supports all of these scenarios, and lets customers extend investments they have already made into the cloud.

Both Chef and Puppet are supported through virtual machine extensions, allowing VMs that are created to support automation. This assumes that a Chef or Puppet environment is already set up. If you would like to set up a new environment, images for Chef and Puppet are available for download and deployment on the [Azure Marketplace](#).

Let's have a closer outlook at these two sets of tools.

Automating Azure virtual machine deployment with Chef

Chef is a great tool for delivering automation and desired state configurations.

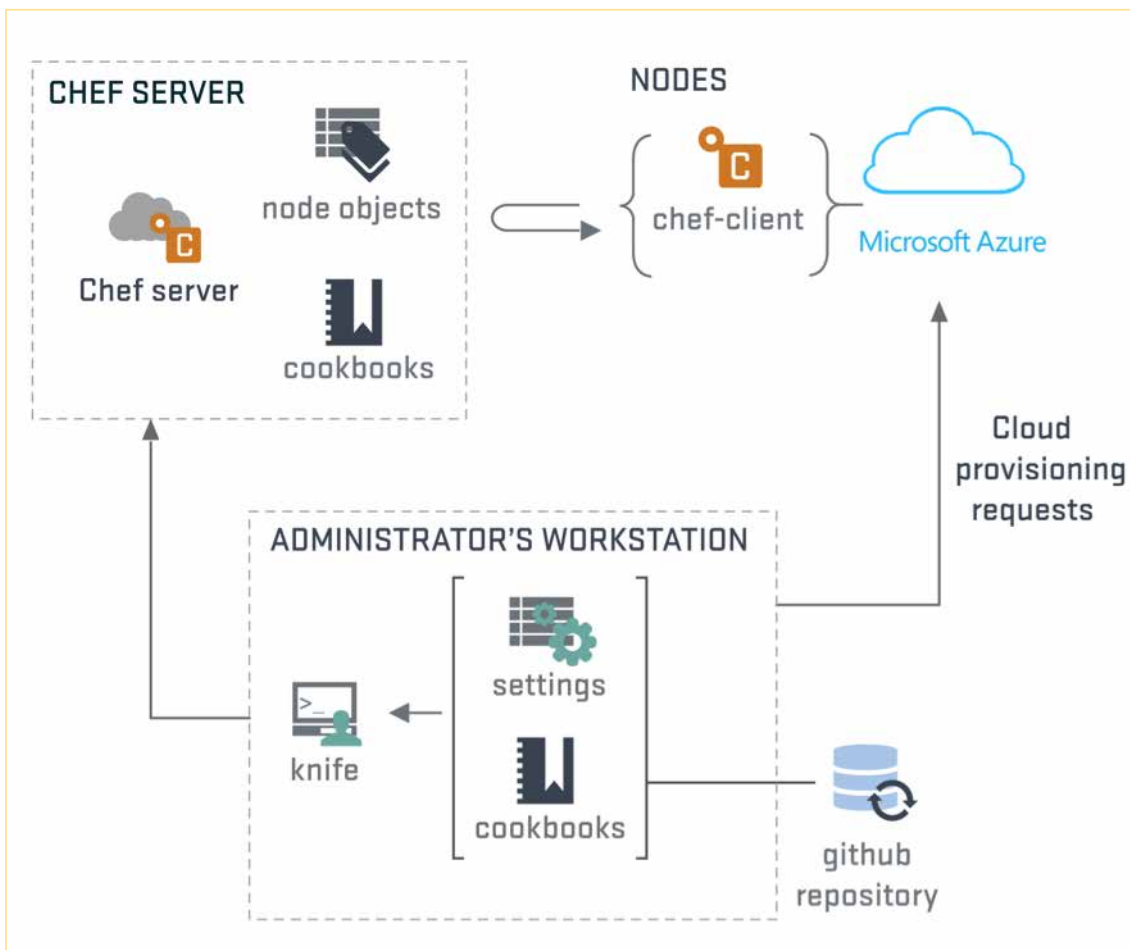
With our latest cloud-api release, Chef provides seamless integration with Azure, giving you the ability to provision and deploy configuration states through a single command.

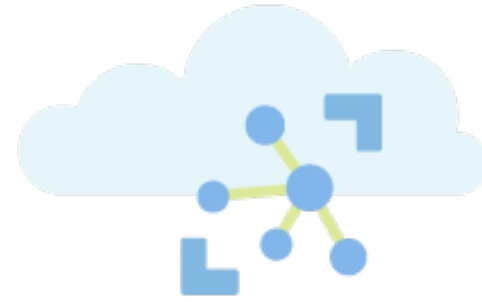
In this article, I'll show you how to set up your Chef environment to provision Azure virtual machines and walk you through creating a policy or "CookBook" and then deploying this cookbook to an Azure virtual machine.

Chef basics

Before you begin, I suggest you review the basic concepts of Chef. There is great material [here](#) and I recommend you have a quick read before you attempt this walkthrough. I will however recap the basics before we get started.

The following diagram depicts the high-level Chef architecture.





Chef has three main architectural components: Chef Server, Chef Client (node), and Chef Workstation.

The Chef Server is our management point and there are two options for the Chef Server: a hosted solution or an on-premises solution. We will be using a hosted solution.

The Chef Client (node) is the agent that sits on the servers you are managing.

The Chef Workstation is our admin workstation where we create our policies and execute our management commands. We run the knife command from the Chef Workstation to manage our infrastructure.

There is also the concept of "Cookbooks" and "Recipes". These are effectively the policies we define and apply to our servers.

Preparing the workstation

First, let's prep the workstation. I'm using a standard Windows workstation. We need to create a directory to store our config files and cookbooks.

First create a directory called C:\chef.

Then create a second directory called c:\chef\cookbooks.

We now need to download our Azure settings file so Chef can communicate with our Azure subscription.

Download your publish settings from [here](#).

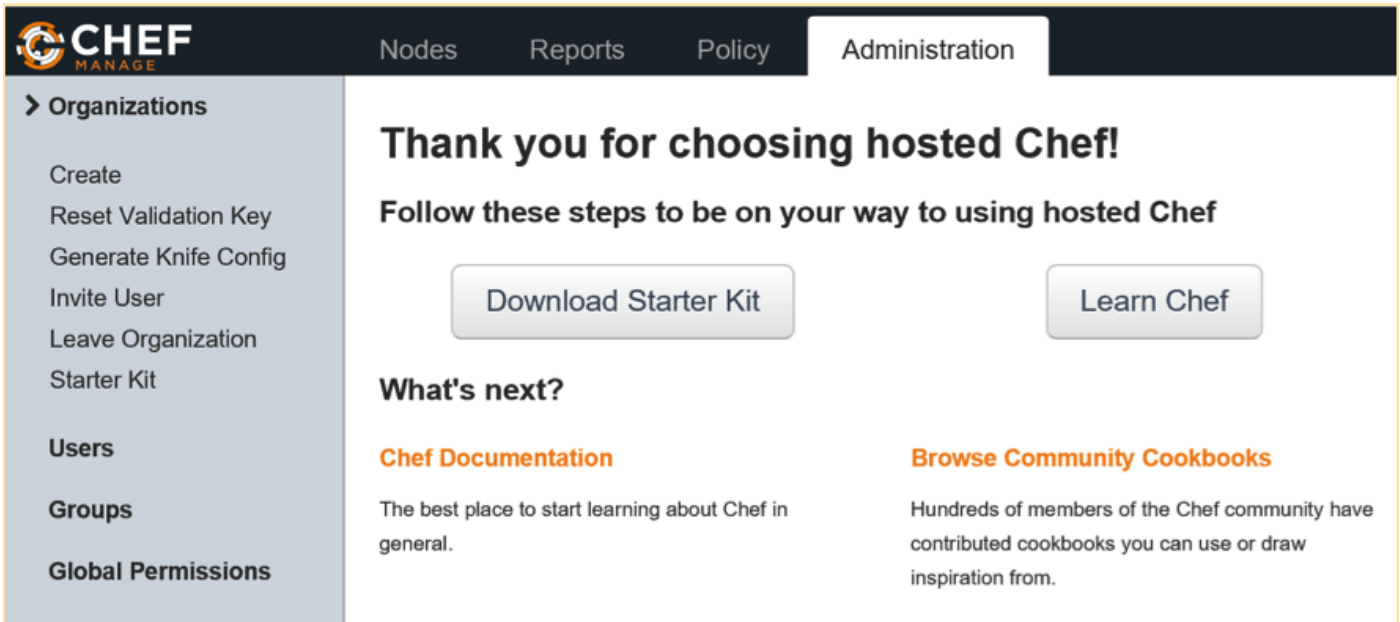
Save the publish settings file in C:\chef.

Creating a managed Chef account

Sign up for a hosted Chef account [here](#).

During the signup process, you will be asked to create a new organization.

A screenshot of a web form titled "Create Organization". The form has a dark header with a close button (X). Below the header, there are two input fields. The first is labeled "Full Name (example: Chef, Inc.)" and contains the text "Azure". The second is labeled "Short Name (example: chef)" and is empty. Below the second field, there is a red error message that says "Short name is required". At the bottom of the form, there are two buttons: "Cancel" and "Create Organization".



Once your organization is created, download the starter kit.

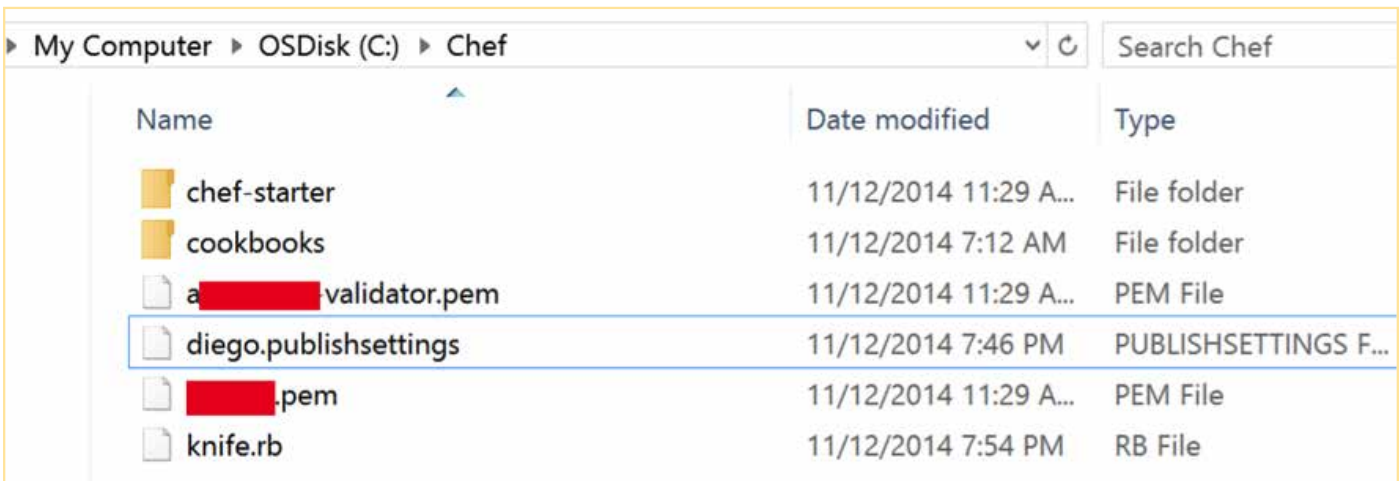
Note

If you receive a prompt warning you that your keys will be reset, it's ok to proceed as we have no existing infrastructure configured as yet.

This starter kit zip file contains your organization config files and keys.

Configuring the Chef workstation

Extract the content of the chef-starter.zip to C:\chef.
 Copy all files under chef-starter\chef-repo.chef to your c:\chef directory.
 Your directory should now look something like the following example.





You should now have four files including the Azure publishing file in the root of c:\chef.

The PEM files contain your organization and admin private keys for communication while the knife.rb file contains your knife configuration. We will need to edit the knife.rb file.

Open the file in your editor of choice and modify the "cookbook_path" by removing the ../ from the path so it appears as shown next.

```
cookbook_path [«#{current_dir}/cookbooks»]
```

Also add the following line reflecting the name of your Azure publish settings file.

```
knife[:azure_publish_settings_file] = «yourfilename.publishsettings»
```

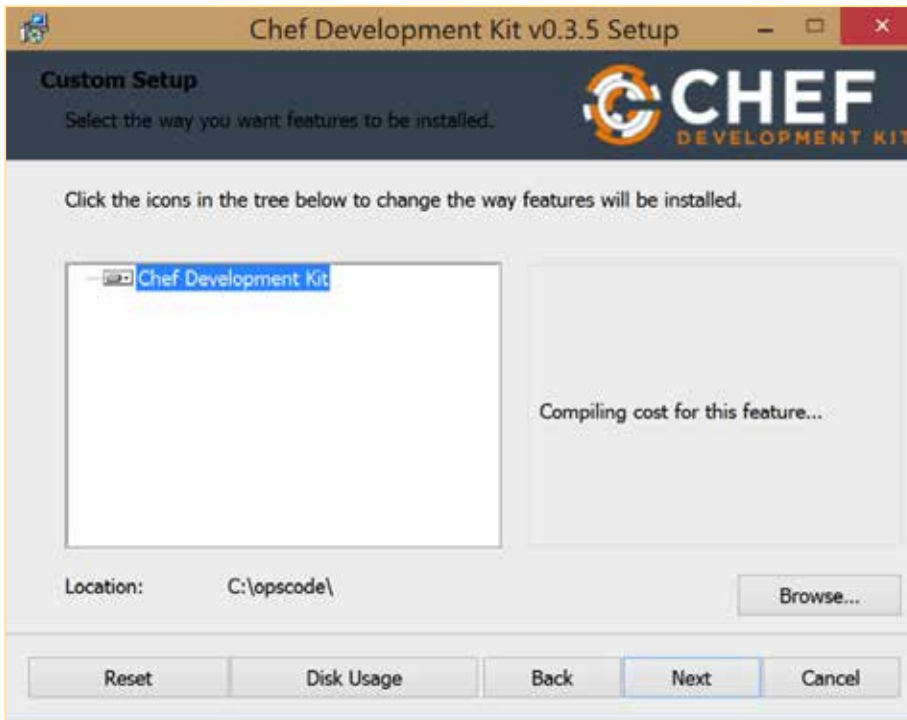
```
current_dir = File.dirname(__FILE__)
log_level      :info
log_location   STDOUT
node_name      [REDACTED]
client_key     "#{current_dir}/[REDACTED].pem"
validation_client_name "a-[REDACTED]-validator"
validation_key  "#{current_dir}/a-[REDACTED]-validator.pem"
chef_server_url "https://api.opscode.com/organizations/a-[REDACTED]"
cache_type     'BasicFile'
cache_options( :path => "#{ENV['HOME']}/.chef/checksums" )
cookbook_path  ["#{current_dir}/cookbooks"]
knife[:azure_publish_settings_file] = "[REDACTED].publishsettings"
```

Your knife.rb file should now look similar to the following example.

These lines will ensure that Knife references the cookbooks directory under c:\chef\cookbooks, and also uses our Azure Publish Settings file during Azure operations.

Installing the Chef Development Kit

Next [download and install](#) the ChefDK (Chef Development Kit) to set up your Chef Workstation.



Install in the default location of `c:\opscode`. This install will take around 10 minutes.

Confirm your PATH variable contains entries for `C:\opscode\chefdk\bin`; `C:\opscode\chefdk\embedded\bin`; `c:\users\yourusername\chefdk\gem\ruby\2.0.0\bin`

If they are not there, make sure you add these paths!

NOTE THE ORDER OF THE PATH IS IMPORTANT! If your opscode paths are not in the correct order you will have issues. Reboot your workstation before you continue.

Next, we will install the Knife Azure extension. This provides Knife with the "Azure Plugin".

Run the following command.

```
chef gem install knife-azure --pre
```

Note

The `--pre` argument ensures you are receiving the latest RC version of the Knife Azure Plugin which provides access to the latest set of APIs.

It's likely that a number of dependencies will also be installed at the same time.

```

C:\Chef>chef gen install knife-azure --pre

Temporarily enhancing PATH to include DevKit...
Building native extensions. This could take a while...
Successfully installed eventmachine-1.0.4
Successfully installed en-winrm-0.6.0
Successfully installed winrm-s-0.2.2
Successfully installed knife-windows-0.8.2
Fetching: knife-azure-1.4.0.rc.0.gem (100%)
Successfully installed knife-azure-1.4.0.rc.0
Parsing documentation for eventmachine-1.0.4
Installing ri documentation for eventmachine-1.0.4
Parsing documentation for en-winrm-0.6.0
Installing ri documentation for en-winrm-0.6.0
Parsing documentation for winrm-s-0.2.2
Installing ri documentation for winrm-s-0.2.2
Parsing documentation for knife-windows-0.8.2
Installing ri documentation for knife-windows-0.8.2
Parsing documentation for knife-azure-1.4.0.rc.0
Installing ri documentation for knife-azure-1.4.0.rc.0
Done installing documentation for eventmachine, en-winrm, winrm-s, knife-windows, knife-azure after 16 seconds
5 gems installed
  
```

To ensure everything is configured correctly, run the following command.

```
knife azure image list
```

If everything is configured correctly, you will see a list of available Azure images scroll through.

Congratulations. The workstation is set up!

Creating a Cookbook

A Cookbook is used by Chef to define a set of commands that you wish to execute on your managed client. Creating a Cookbook is straightforward and we use the chef generate cookbook command to generate our Cookbook template. I will be calling my Cookbook web server as I would like a policy that automatically deploys IIS.

Under your C:\Chef directory run the following command.

```
chef generate cookbook webserver
```

This will generate a set of files under the directory C:\Chef\cookbooks\webserver. We now need to define the set of commands we would like our Chef client to execute on our managed virtual machine.

The commands are stored in the file default.rb. In this file, I'll be defining a set of commands that installs IIS, starts IIS and copies a template file to the wwwroot folder.



Modify the `C:\chef\cookbooks\webserver\recipes\default.rb` file and add the following lines.

```
powershell_script 'Install IIS' do
  action :run
  code 'add-windowsfeature Web-Server'
end

service 'w3svc' do
  action [ :enable, :start ]
end

template 'c:\inetpub\wwwroot\Default.htm' do
  source 'Default.htm.erb'
  rights :read, 'Everyone'
end
```

Save the file once you are done.

Creating a template

As we mentioned previously, we need to generate a template file which will be used as our `default.html` page.

Run the following command to generate the template.

```
chef generate template webserver Default.htm
```

Now navigate to the `C:\chef\cookbooks\webserver\templates\default\Default.htm.erb` file. Edit the file by adding some simple "Hello World" HTML code, and then save the file.

Upload the Cookbook to the Chef Server

In this step, we are taking a copy of the Cookbook that we have created on our local machine and uploading it to the Chef Hosted Server. Once uploaded, the Cookbook will appear under the Policy tab.+

Nodes		Reports	Policy	Administration
Showing All Cookbooks				
Cookbook	Current Version			
webservers	0.1.2			

```
knife cookbook upload webservers
```

Deploy a virtual machine with Knife Azure

We will now deploy an Azure virtual machine and apply the "Webservers" Cookbook which will install our IIS web service and default web page. In order to do this, use the knife azure server create command. An example of the command appears next.

```
knife azure server create --azure-dns-name 'diegotest01' --azure-vm-name 'testserver01' --azure-vm-size 'Small' --azure-storage-account 'portalvhdsxxxx' --bootstrap-protocol 'cloud-api' --azure-source-image 'a699494373c04fc0bc8f2bb1389d6106__Windows-Server-2012-Datacenter-201411.01-en.us-127GB.vhd' --azure-service-location 'Southeast Asia' --winrm-user azureuser --winrm-password 'myPassword123' --tcp-endpoints 80,3389 --r 'recipe[webservers]'
```

The parameters are self-explanatory. Substitute your particular variables and run.

Note

Through the the command line, I'm also automating my endpoint network filter rules by using the `--tcp-endpoints` parameter. I've opened up ports 80 and 3389 to provide access to my web page and RDP session.

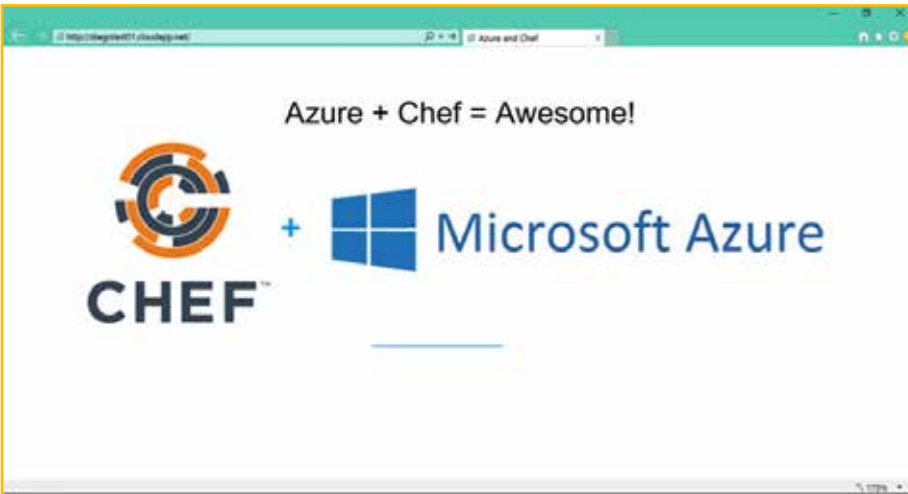
Once you run the command, go to the Azure portal and you will see your machine begin to provision. The command prompt appears next.





```
cmd /c:chef-knife azure vm create --azure-dns-name 'diagtest01' --azure-vm-name 'testserver01' --azure-vm-size 'Small' --azure-rt
...
Waiting for virtual machine to reach status 'provisioning'.....on state 'provisioning' reached after 2.79 minutes.
Waiting for virtual machine to reach status 'ready'.....on state 'ready' reached after 2.23 minutes.
VM Name: diagtest01.cloudapp.net
VM Size: Small
Azure Source Image: a699494372c04f0bc8f23a138966106__Windows-Server-2012-Datacenter-201411-01-en-us-127GB.vhd
Azure Service Location: Southeast Asia
Public IP Address: 104.43.9.88
Private IP Address: 100.72.52.27
WinRM Port: 5985
TCP Ports: [{"Name": "Icmpnet_3389", "testserver01", "Pip": "104.43.9.88", "PublicPort": "3389", "LocalPort": "3389"}, {"Name": "Icmp
...
Waiting for Resource Extension to reach status 'urgent provisioning'....Resource extension state 'urgent provisioning' reached aft
...
Waiting for Resource Extension to reach status 'provisioning'.....Resource extension state 'provisioning' reached after
...
Waiting for Resource Extension to reach status 'ready'.....Resource extension state 'ready' reached
...
VM Name: diagtest01.cloudapp.net
VM Size: Small
Azure Source Image: a699494372c04f0bc8f23a138966106__Windows-Server-2012-Datacenter-201411-01-en-us-127GB.vhd
Azure Service Location: Southeast Asia
Public IP Address: 104.43.9.88
Private IP Address: 100.72.52.27
WinRM Port: 5985
TCP Ports: [{"Name": "Icmpnet_3389", "testserver01", "Pip": "104.43.9.88", "PublicPort": "3389", "LocalPort": "3389"}, {"Name": "Icmp
...
Environment: default
Hostid: ("res:ip:localhost")]
```

Once the deployment is complete, we should be able to connect to the web service over port 80 as we had opened the port when we provisioned the virtual machine with the Knife Azure command. As this virtual machine is the only virtual machine in my cloud service, I'll connect it with the cloud service url.



Don't forget we can also connect through an RDP session from the Azure classic portal via port 3389.

Deploying Puppet Enterprise in Microsoft Azure

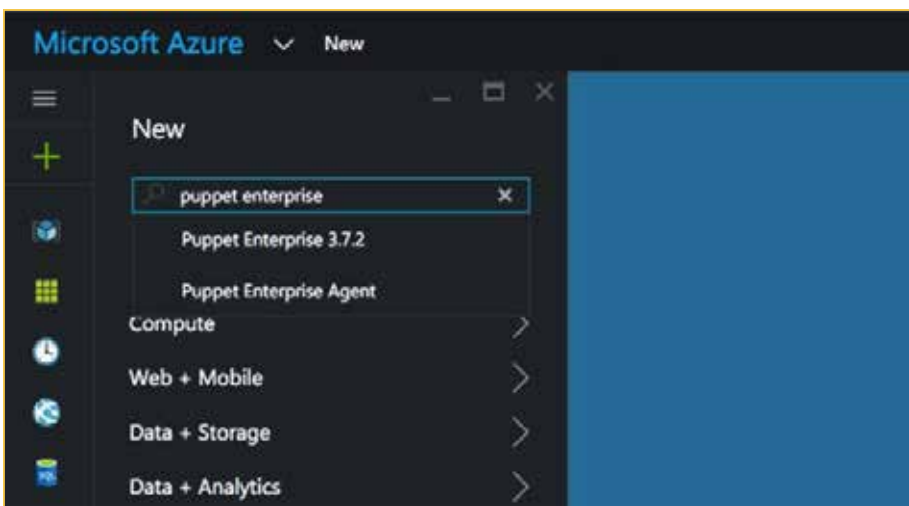
Puppet Enterprise is now offered in the Microsoft Azure Marketplace, enabling you to quickly spin up VMs with the latest version of PE for evaluation. This section will help you use Azure to provision a puppet master and a puppet agent.

Install a Puppet Master

Before you get started, you should have created an [Azure account](#).

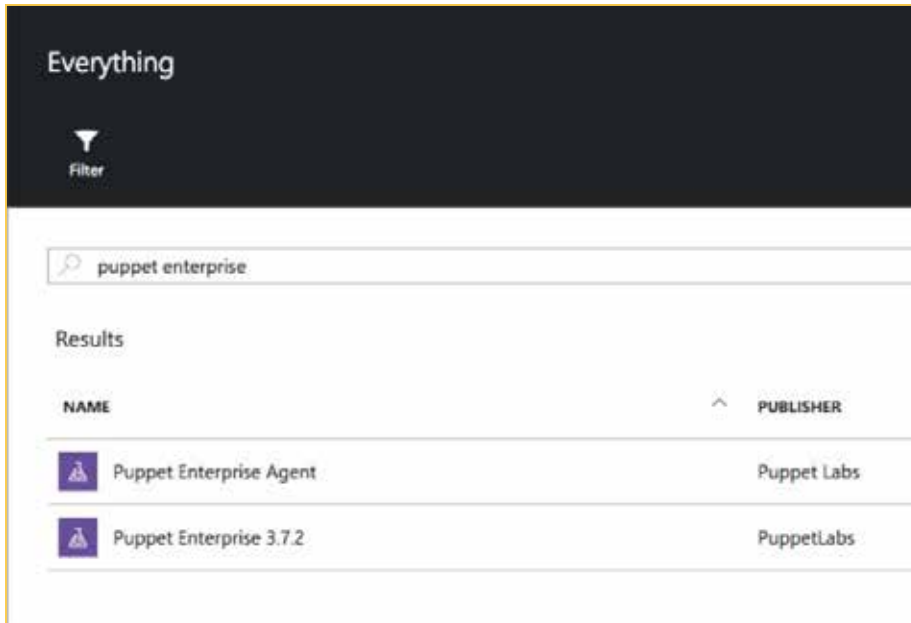
Step 1. Open the Azure portal.

In the [Azure portal](#), click NEW. Type "puppet enterprise" in the search box and select Puppet Enterprise 3.7.2.

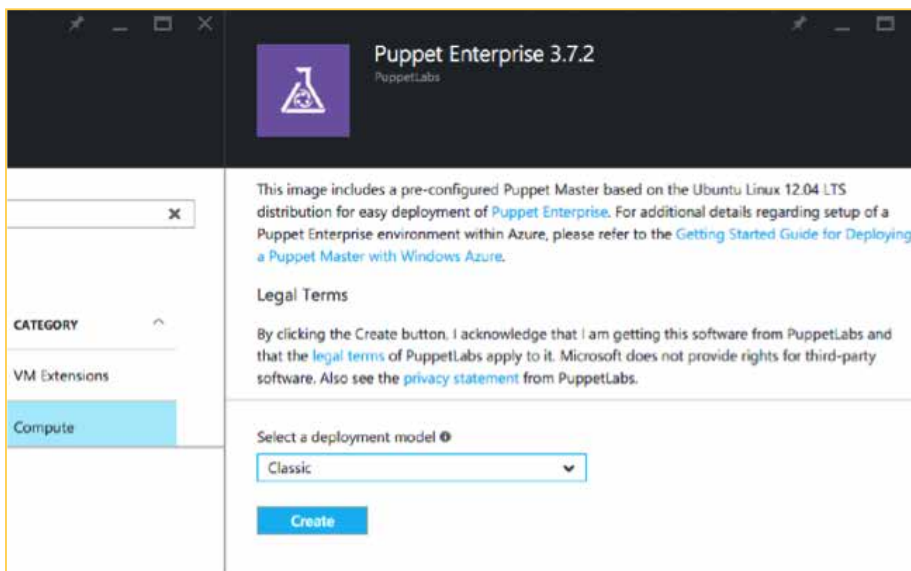


Step 2. Choose the PE image.

Select Puppet Enterprise 3.7.2



Step 3. Select the Classic Deployment Model Under Select a Deployment Model, select Classic.



HOST NAME: Can be any name of 3-15 characters, consisting of letters, numbers and hyphens. Letters must be lower-case; the SSL certificate won't be created if your name contains upper-case letters.

USER NAME: Provide a user name. This is basically your administrative account for accessing the machine. User name should not be "puppet", "pe-admin", or "pe-puppet".

AUTHENTICATION TYPE: You can choose to upload your SSH key, or to provide a password. For these steps, we use a password: Click PROVIDE A PASSWORD and type one in.

If you want to use an SSH key, follow the steps to get a required X509 certificate described under [SSH Key Generation](#).

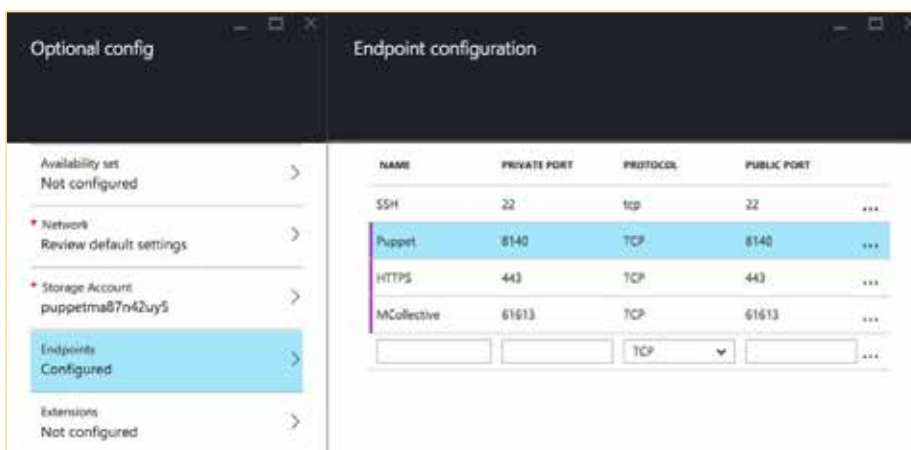
PRICING TIER: Ensure that at least Standard_D2_v2 (2 core, 7 GB memory) is selected. You can bump up the size, but it's not necessary and will increase the cost. A smaller size will not provide enough power and the VM will underperform.

For documentation on port usage, see Firewall Configuration in [System Requirements and Preinstallation](#).

OPTIONAL CONFIGURATION: In the ENDPOINTS section, configure your VM's ports. The SSH port is pre-set, because you can only run a puppet master on a Linux image.

Add the following mandatory ports. The first one can be selected from the drop-down list. Puppet and MCollective are the additional ports, and must be added manually.

HTTPS, to open port 443 for the PE console.
 Puppet, to open port 8140 for puppet agents.
 MCollective, to open port 61613 for MCollective.



RESOURCE GROUP: Select an existing resource group or create a new one.

SUBSCRIPTION: Select the desired subscription.

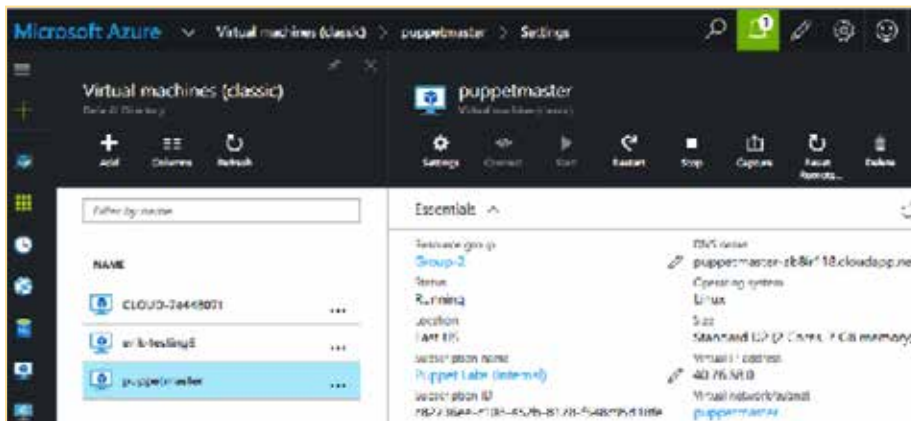
LOCATION: Choose your region. By default, the region is prefilled with the region on your account

After you have configured your location, click Create.

The provisioning process begins. The PE install takes around 15 minutes. From the left nav, click the icon for Virtual machines (classic). Click the VM you just created. Take note of the fully qualified domain name (FQDN) under DNS name.

For example: `puppetmaster-w8msh322.cloudapp.net`

A note about DNS settings: Azure provides an internal and an external DNS name for your VM. For these steps, we use the external DNS name. This way, you can connect to puppet agents outside the Azure network, a common scenario. Doing so, however, entails opening two additional ports in the ENDPOINTS area.

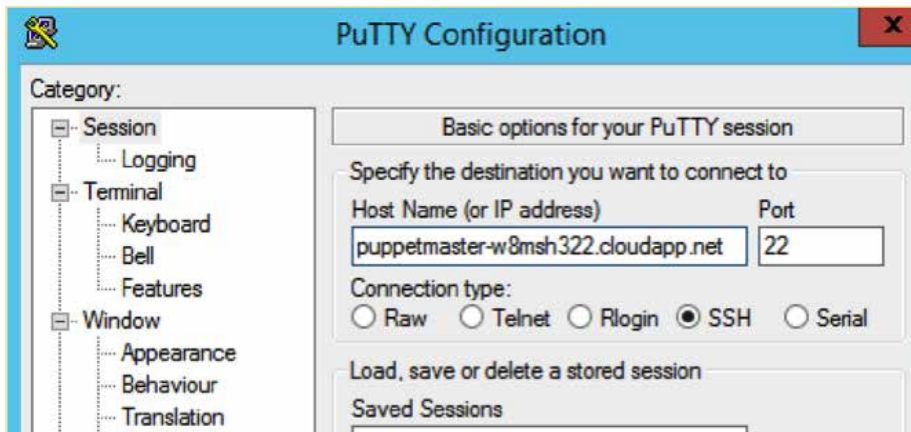


SSH to the PE Virtual Machine

In your CLI or other ssh client, run the command `ssh <username>@<public DNS name>` with the DNS name you found above.

For example: `ssh azureuser@pe-demo.cloudapp.net`

Note: If you don't have an SSH client, download PuTTY for free [here](#).



The first time you SSH to the puppet master you will be asked to accept the remote host's public key. The message says that the authenticity of the host cannot be established, and asks if you're sure you want to connect anyway.

Type **"yes"**. Then, type in the password you created when you set up the PE VM.

After 15 minutes has elapsed, move on to the next step.

Log Into the PE Console

Before you create the puppet agent VM, you need the password that's generated when the puppet master is provisioned. This is used to access the PE console, a web GUI, that enables you to manage node requests, assign classes to nodes, trigger puppet runs, and much more. For more information, see the [PE console section of the online documentation](#).

In this section, you get the password, and then you access the console.

Step 1. Get the console credentials.

Run `sudo watch tail /etc/puppetlabs/installer/database_info.install` Locate the setting, `q_puppet_enterpriseconsole_auth_password`, which has the password appended to it.

It looks similar to this: `q_puppet_enterpriseconsole_auth_password=Foryurcit0`
If the password does not appear, the console is not yet ready.

```
Every 2.0s: tail /etc/puppetlabs/installer/database_info.install
_q_puppet_enterpriseconsole_database_name=console
q_puppet_enterpriseconsole_database_password=wriaFrav3
q_puppet_enterpriseconsole_database_user=console
q_puppetdb_database_name=pe-puppetdb
q_puppetdb_database_password=Grapen0v6
q_puppetdb_database_user=pe-puppetdb
q_rbac_database_name=pe-rbac
q_rbac_database_password=19C0nZKRjfJtFUxm3okZ
q_rbac_database_user=pe-rbac
q_puppet_enterpriseconsole_auth_password=Foryurcit0
```

Copy the password for use in step 3.
The username for the console is admin.

Step 2. Open the PE console.

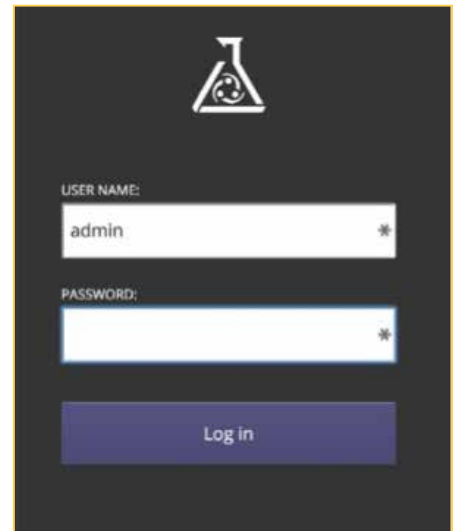
In a browser window, type "`https://<public DNS name>`".
For example: "`https://pe-demo.cloudapp.net`". Ensure you're using "https", not "http".
This is a good way to find out when your PE VM has been created. You'll know it's ready when you see the login for the console (see image).

Step 3. Log into the PE console.

In the PE console login, type in your user name and password to log into the console. The first time you connect to the console, you get an SSL security warning. It's safe to proceed. For more information, see the [the PE documentation](#).

Because you haven't set up any agents yet, there's not much information here.
Click Node and you'll see the puppet master node listed, along with its private DNS name.
After you add an agent in the next section, you'll return to the console.

Note: If you forget to open a port when you're setting up the puppet master, you can do it afterwards: on the Azure dashboard, click the arrow on the puppet master VM, click ENDPOINTS, click Add, choose standalone endpoint, and select or create the port you want.



Note: If your puppet master did not install successfully, check out the log that's located here:

`/var/log/upstart/puppetinit.log`



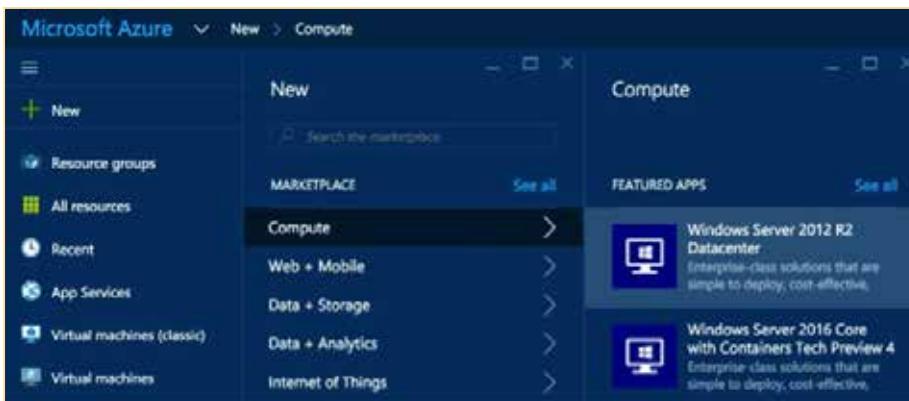
Install a Puppet Agent

These steps show how to bootstrap a puppet agent with an Azure VM at provision time. The steps are similar to configuring the PE VM.

Step 1. Launch the VM workflow.

In the Azure portal, click NEW and then click

COMPUTE -> Windows Server 2012 R2 Datacenter.



Step 2. Select the Classic Deployment Model

Under **Select a Deployment Model**, select **Classic**.



Step 3. Begin configuring the VM.

Provide the following information and then click the arrow to go to the next page.

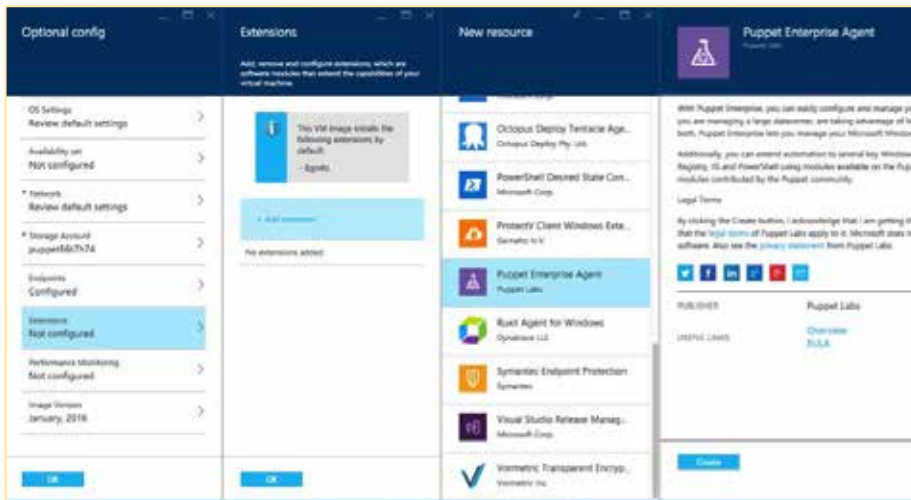
HOST NAME: Can be any name of 3-15 characters, consisting of letters, numbers and hyphens.

USER NAME: Can be anything you want. This is basically your administrative account for accessing the machine.

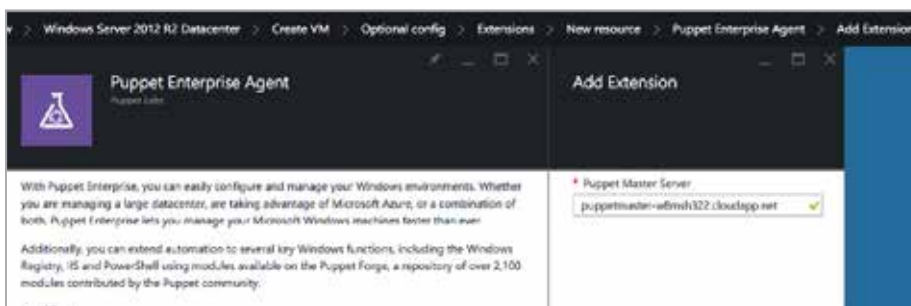
PASSWORD: Choose a password.

PRICING TIER: Choose whichever size will suit this VM's anticipated workload.

OPTIONAL CONFIGURATION: In the EXTENSIONS section, select "Add Extension". Select the Puppet Enterprise Agent extension. Click "Create".



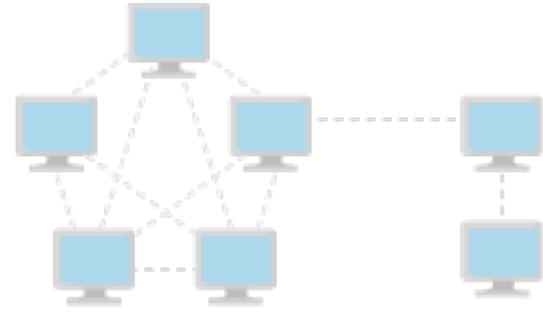
Step 4: Configure the extension to use the Puppet Master Insert the FQDN of the Puppet Master. Then, click OK.



Hint: From the left nav, click Virtual Machines (classic).
Click the Puppet Master VM. Find the fully qualified domain name (FQDN) under DNS name.

Step 5: Continue configuring the Virtual Machine

RESOURCE GROUP: Select the resource group you previously created.



SUBSCRIPTION: Select the desired subscription.

LOCATION: Select the desired location.

After you have configured your location, click Create. Once the VM is provisioned, the puppet agent is installed and it registers with the puppet master you designated. This process can also take several minutes.

When it's ready, the agent sends a certificate request to the master.

Approve the Agent Node Request

In this section, you view the puppet agent in the PE console, and accept the agent's Certificate Signing Request (CSR).

Step 1. Open the PE console as you did previously.

Step 2. Locate and approve the agent request.

The screenshot shows the Puppet Enterprise console interface. At the top, it displays 'puppetenterprise console' and '1 Node Request 1 of 10 Licenses admin Resource'. Below the navigation bar, there is a section titled 'Pending node requests: 1'. This section contains three buttons: 'Accept All' (blue), 'Reject All' (red), and a grey button that says 'There are no accepted/rejected requests to clear'. Below these buttons is a table with the following data:

Name	Fingerprint	Action
puppetagent-0.puppetagent-0.g6.int...	73:20:96:51:41:D3:10:D5...	Accept Reject

At the top of the console window, click node requests to open the node requests view.

There should only be one request in the list. Click Accept.

If your agent isn't listed in the node requests page, see the following section.

Establish a Remote Connection Using RDP

The following steps describe what to do if your puppet agent isn't connecting to your puppet master.

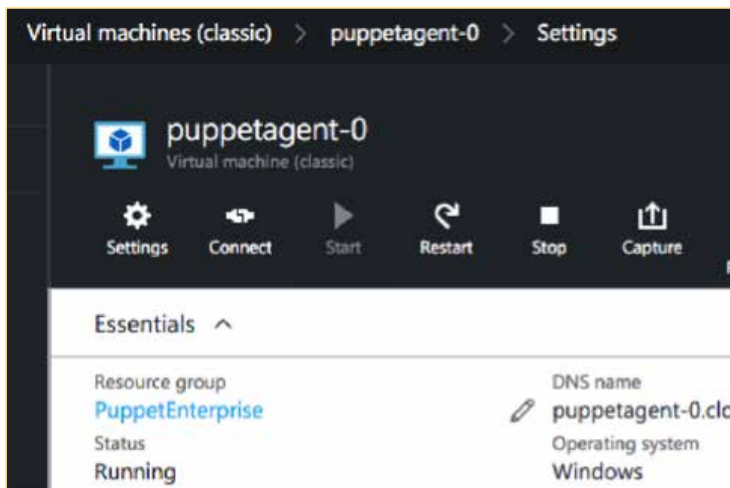
Step 1. Navigate to the VM dashboard.

In Azure, from the left nav, click Virtual Machines (classic). Click the agent VM.

Step 2. Download the RDP file.

Download the RDP file by clicking "Connect" from the tool bar at the top of the dashboard.

Double click on the RDP file to connect to the VM. Login is "hostname/username". For more information, see the [Azure RDP documentation](#).



Step 3. Run Puppet agent.

Search cmd, right-click Start Command Prompt with Puppet and click Run as administrator. In PowerShell, type puppet agent --configprint server.

Is your puppet agent trying to connect to the correct puppet master node?



Now that you've completed this Getting Started Guide, check out the [Puppet Enterprise documentation](#) to learn more about the automation and configuration capabilities of PE.

Hope this information has been helpful for you! Go and start your infrastructure as code journey with Azure today!

Infrastructure as code on Azure with Puppet & Chef