Blu Documentation

Release 0.4.1

Anooshiravan Ahmadi

May 20, 2016

Contents

1 About **3**

2 Blu PowerShell Service **5**
- 2.1 About PowerShell Service 5
- 2.2 Using blu_script Chef resource 6
- 2.3 Marshalling and Type conversion 7
- 2.4 Guards and interpreters 9
- 2.5 Notifiers 9

3 BluStation **11**
- 3.1 About BluStation 11
- 3.2 Import BluStation to PowerShell 11
- 3.3 Connect to Chef API 11
- 3.4 Using Chef API 12
- 3.5 API Graphical User Interface 13

4 .Net Reference **17**
- 4.1 BluAPI 17
- 4.2 BluLang 17
- 4.3 BluIpc 17
- 4.4 BluService 17
- 4.5 BluShell 17

The quickest way to get started with Blu PowerShell service is to get a copy of BluService.exe and BluShell.exe and save it into C:\Blu folder. Then register the Windows service by:

```
sc create "Blu Powershell Runspace Service" binpath= C:\Blu\BluService.exe
```

Then go to C:Blu folder and execute a simple PowerShell command using BluShell.exe:

```
cd C:\Blu
BluShell.exe -Command 'Get-Command;'
```

This PowerShell command will show you a list of available commands, and is running in BluShell seamlessly as inside a PowerShell console. Now try:

```
BluShell.exe -Command "$a = Get-Command;"
```

Then close the powershell session. Normally this is going to garbage collect all the objects and variables that are defined in a PowerShell runspace. To illustrate how BluService changes this behaviour, start a command prompt again (or a PowerShell session) and execute:

```
cd C:\Blu
BluShell.exe -Command "$a"
```

As you can see, the variable $a still returns a valid list of all available commands. This new PowerShell behavior is extremely useful for PowerShell automation and specially in Chef cookbooks. It happens because Blu PowerShell service does not dispose the variables in current the scope unless you dispose them manually. You can also look at Windows event viewer for event id 271, source BluService to monitor what is running under the hood.

To know more about many other futures of Blu framework, please continue reading:

CHAPTER 1

About

From the ground up **Blu** is designed to be **Chef** and **Terraform** compatible. In Terraform it leverages `file` and `remote_exec` provisioner only. It communicates to Chef server as the configuration management system.

Note: If you are not using Chef as the configuration management system for your infrastructure, then this tool set is probably not useful for you.

Chef, which is extensively supported by DevOps community, is one the most methodological ways of configuration management of today's infrastructures. The value of those methods and concepts like resources, providers, data bags, organizations, etc. are not limited to any specific OS. They are as valuable in Windows OS as they are in any other operating system.

Blu is a new way to leverage Chef methods in Windows using PowerShell. To use Blu you need a Chef server installed and properly configured in your infrastructure. Blu is a set of Chef client-side tools, following the methods and concepts, accepted by the Chef community.

BluService.exe and **BluShell.exe** are server and client executables of Blu PowerShell service. BluService which is running as a Windows service is a PowerShell runspace and pipeline which contains PowerShell objects and pipe data during a chef run. One of the major problems of the current powershell_script and dsc resources in chef client is that each PowerShell block is being executed as isolated script lines of code and one piece of code has no access to the previous objects of powershell block. It means each instance of powershell_script is "blind" to what happened before. This is not the way that PowerShell is designed for and such a limitation causes many problems specially when switching (zigzaggin) between PowerShell and Ruby in the course of a cookbook. PowerShell is aware of Windows object model and CmdLet data output is a System.Object. If we execute isolated powershell_scripts without system state awareness and then try to interpret the restul (as Object) in the form of ShellOut, we simply cause a hell of trouble shooting when something goes wrong with the script block and also lose state awareness each time we scape to Ruby. PowerShell service is meant to address this issue: by creating a constant runspace and a single pipeline for each PS block in the main runspace. By this way the powershell_script block is aware of what happened before him (unless we dispose the runspace and garbage collect the objects), so a PowerShell cookbook can be written very structurally where all variables and objects just needs to be defined one time at the first block or the beginning of the cookbook code. You can find more details of this concept in the PowerShell service documnetation.

BluStation.dll, the PowerShell CmdLets library of Blu and can be used in conjunction with Ruby chef-client, or as an additional tool-set for the existing cookbooks, or as an Ruby-less chef-client by itself. It is not meant to "replace" chef-client nor to limit your choices by forcing you to choose one syntax to another. It is written to extend your choices, to query the Chef server and execute Chef methods and/or run cookbooks natively on Windows. **The key concept is 100% compatibility:** Any conflicting behavior of BluStation with the latest version of Chef Client or Chef Server is considered as a problem.

Minimum Complexity: No installation, configuration, or even updating path variables are required. To use Blu, you just add the blu_toolkit cookbook to the runlist of the node and the installation is no more than file copies. The

reason behind this portable design is an educated guess: When your are provisioning a machine, the least amount of processing should be required to enable the machine to communicate with the configuration management server. When the file is copied, the rest are just the most familiar PowerShell methods and CmdLets, supporting all PowerShell technologies including DSC.

DevOps: Blu is a young project and is under heavy development. The component shared in GitHub are meant to be production ready, but it doens't mean the solutions and ideas that are chosen to achive specific goals are on their ideal average. You need to be a real DevOps when using Blu: Don't wait for someone else to save your day; **share your ideas, join the club, change the code as you desire, request a pull and make it work!**

CHAPTER 2

Blu PowerShell Service

2.1 About PowerShell Service

Powershell_script resource in a normal Chef run is limited to a single PowerShell runspace. When the resource execution is completed, all variables and object are garbage collected and are not accessible anymore. This has major affects on the way we write cookbooks using PowerShell: a PowerShell configuration script cannot be structured in multiple resources and/or files. For extensive PowerShell scripting, this ends up to an unreadable and long scripts in a single powershell_script resource and/or using template for config scripts. Also Ruby runtime is 32-bit which limits powershell_script to only 32-bit binaries and CmdLets, e.g. Microsoft Exchange Server and SharePoint.

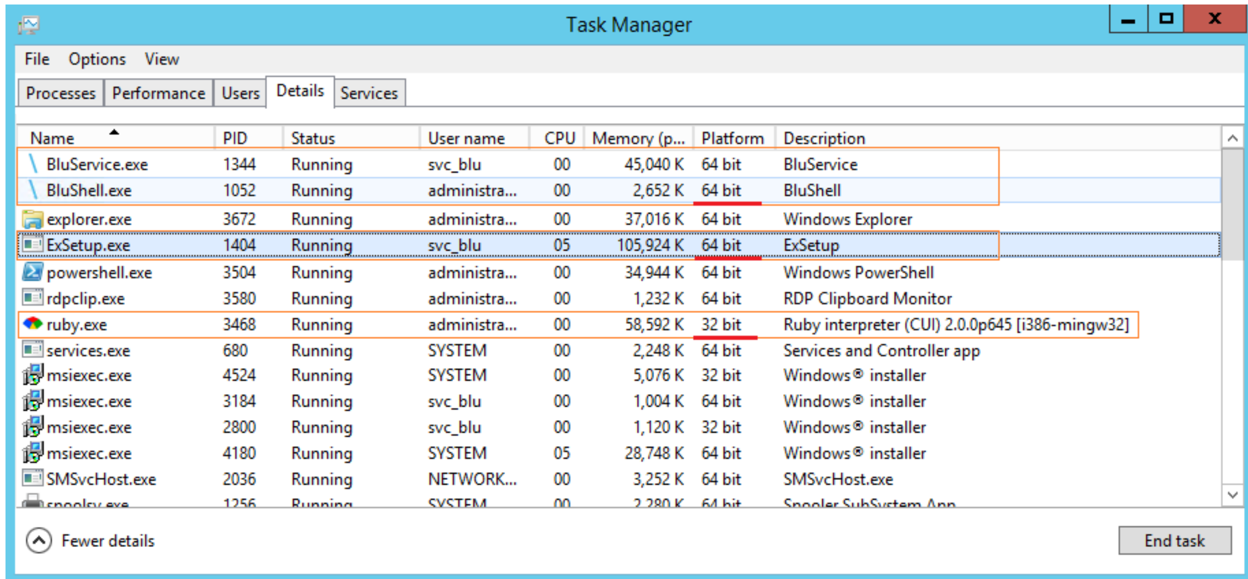
Note: In Windows, It is not possible to run a 64-bit process as a child of a 32-bit process. The only method to communicate between 32-bit and 64-bit processes is IPC (interprocess communication) This method is used in Blu PowerShell service to escape the 32-bit limitation of Ruby.exe

Blu PowerShell service (a part of the blu cookbook) is meant to address both issues. This Windows service which is installed automatically when you add blu cookbook to the node run_list, invokes a PowerShell runspace which is valid during the life cycle of the Windows service. The objects are only garbage collected when it is specifically requested in the recipe. The following code disposes Blu PowerShell runspace:

```
# Dispose Blu PowerShell Runspace
blu_script 'DisposeRunspace' do
    code 'DisposeRunspace'
action :run
end
```

Warning: If a cookbook doesn't dispose the runspace in the default recipe, all PowerShell variables and objects from the previous chef run are still valid. This might cause unexpected results. Therefore it is a good practice to dispose PowerShell runspace at the begin and end of a cookbook code.

The following screenshot illustrates how Blu PowerShell Service escapes the 32-bit Ruby/Chef boundary during Microsoft Exchange 2016 setup and runs ExSetup.exe in a 64-bit address space:



2.2 Using blu_script Chef resource

Chef defines a resource as follows:

A resource is a statement of configuration policy that:

- Describes the desired state for a configuration item
- Declares the steps needed to bring that item to the desired state
- Specifies a resource type—such as package, template, or service
- Lists additional details (also known as resource properties), as necessary
- Are grouped into recipes, which describe working configurations
- Where a resource represents a piece of the system (and its desired state), a provider defines the steps that are needed to bring that piece of the system from its current state into the desired state.

The blu_script resource inherits all behavior of a Chef resource and is defines in the blu cookbook in the script.rb file:

```
actions :run
actions :define
default_action :run
attribute :code, kind_of: String, required: true
```

The provider that supports this resource is also defined in the blu cookbook as follow:

```
action :run do
new_resource.updated_by_last_action(true)
execute "blu_script" do
    cwd node['blu']['root']
    command "#{node['blu']['root']}\\BluShell.exe -Command \"#{new_resource.code}\""
    only_if { ::Win32::Service.exists?('BluService') }
end
end

action :define do
new_resource.updated_by_last_action(true)
```

```
execute "blu_script" do
    cwd node['blu']['root']
    command "#{node['blu']['root']}\\BluShell.exe -Define \"#{new_resource.code}\""
    only_if { ::Win32::Service.exists?('BluService') }
end
end
```

You can use **blu_script** resource in recipe like the **powershell_script** resource. In the following example we load Active Directory management snap-in by the **define** action:

```
# Load AD module
blu_script 'Load AD module' do
code <<-EOF
    If (!(Get-module ActiveDirectory)) { Import-Module ActiveDirectory }
    If (!(Get-module ServerManager)) { Import-Module ServerManager }
EOF
action :define
end
```

Note: From this point on, your blu_script resources has access to ServerManager and ActiveDirectory snap-ins everywhere in recipes. These snap-ins remain valid until you dispose the runspace as specified above.

A good practice is to define variables and snap-in by **action :define** and run PowerShell converge scripts by **action :run** so that PowerShell code is more readable and also you can take advantage of other mechanisms of **define** action like type conversion. **Marshalling** between Ruby and PowerShell is covered later in this document.

Warning: If you don't check the loaded snap-in before loading them, by **If (!(Get-module <name>))** and also do not dispose PowerShell runspace, in the next Chef run you get an error that the required snap-in is already loaded.

2.3 Marshalling and Type conversion

Currently there are 4 new data types are added to the blu namespace, namely blu_true, blu_false, blu_nil and blu_array:

2.3.1 Booleans (blu_true / blu_false):

You can define blu_true and blu_false in a node attribute, example:

```
default['myapp']['attribute1'] = 'blu_true'
default['myapp']['attribute2'] = 'blu_false'
```

Define them in the blu_script resource:

```
# Boolean attributes in PowerShell variables
blu_script 'boolean attributes in powershell variables' do
code <<-EOF
    $Attribute1  = '#{node['myapp']['attribute1']}'
    $Attribute2  = '#{node['myapp']['attribute2']}'
EOF
action :define
end
```

And use them in the blu_script:

```
# Boolean attributes in PowerShell
blu_script 'boolean attributes in powershell' do
code <<-EOF
    if ($Attribute1)
    {
        # Do some work
    }

    if (!$Attribute2)
    {
        # Do some other works
    }
EOF
action :run
end
```

When the resource action is **define**, Blu PowerShell service marshals these attributes from string to PowerShell specific boolean types of **$True** and **$False**.

Note: Such a type conversion does not happen when the resource action is ":run". We assume that all variables that need to be converted are defined in the blu_scripts with ":define" action.

2.3.2 Null (blu_nil):

When the resource action **:define**; an attribute of string 'blu_nil' is converted to PowerShell **$Null**, example:

```
default['myapp']['attribute3'] = 'blu_nil'
```

Define $Null in blu_script:

```
# Null in PowerShell
blu_script 'null in powershell variable' do
code <<-EOF
    $Attribute3  = '#{node['myapp']['attribute3']}'
EOF
action :define
end
```

Use the variable in blu_script:

```
blu_script 'null in powershell' do
code <<-EOF
    if ($Attribute3 -eq $Null)
    {
        # Do some work
    }
EOF
action :run
end
```

2.3.3 Array (blu_array@):

When the resource action **:define**; an attribute of string 'blu_array' is converted to PowerShell array, the syntax of a blu_array definition is:

```
my_array = "blu_array@('<string1>', '<string2>', '<string3>')"
```

Example:

```
default['myapp']['attribute4'] = "blu_array@('pizza', 'ravioli', 'macaroni')"
```

Define $array in blu_script:

```
# Define array in PowerShell
blu_script 'array in powershell variables' do
code <<-EOF
    $Foods  = '#{node['myapp']['attribute4']}'
EOF
action :run
end
```

Do something with it:

```
# Array in PowerShell
blu_script 'array in powershell' do
code <<-EOF
    foreach ($ItalianFood in $Foods) {
        # buon appetito
    }
EOF
action :run
end
```

2.4 Guards and interpreters

Because blu_script is a Chef LWRP, all the syntax and rules of a resource guard and interpreters are valid, example:

```
default['myapp']['guard'] = 'down'
```

```
# Guard in blu_script
blu_script 'guard example' do
code <<-EOF
    if ($Attribute3 -eq $Null)
    {
        # Do some work
    }
EOF
action :run
only_if { node['myapp']['guard'] == 'down' }
end
```

2.5 Notifiers

You can use blu_script notifiers like other resources in Chef, for example:

```
# Reboot handler
reboot 'if_pending' do
action :nothing
only_if { reboot_pending? }
end
```

```
# Notifier in blu_script
blu_script 'notifier example' do
code <<-EOF
    if ($Attribute3 -eq $Null)
    {
        # Do some work
    }
EOF
action :run
notifies :reboot_now, 'reboot[if_pending]'
end
```

CHAPTER 3

BluStation

3.1 About BluStation

BluStation.dll is currently an experimental library for Blu project which contains:

- A fully functional Chef API layer (BluApi.dll) written in C#
- A Chef DSL language transpiler to convert Ruby syntaxt to PowerShell (BluLang.dll)
- A GUI that can be started as a PowerShell CmdLet to interact wit BluApi and BluLang libraries in a graphical mode

Note: Unlike Blu PowerShell Service which is meant for Chef Production use in a Windows environment; BluStation libraries are not production ready and are meant to provide building blocks for future projects. Therefore it is provided AS IS with no guranatee that it would be useful for your specific use case.

3.2 Import BluStation to PowerShell

Copy BluStation.sll into C:\Blu folder and then import the module using:

```
cd C:\Blu
Import-Module .\BluStation.dll
```

To check if BluStation is loaded correctly, run:

```
Get-Module
```

3.3 Connect to Chef API

To connect to Chef API use Connect-Chef CmdLet:

Parameters:

```
Connect-Chef
    -Org <Chef Organization Uri>(Required)
    -Client <ClientName>(Required)
    -Node <NodeName>(Optional)
```

```
    -Validator <ValidatorName>(Optional)
    -KeyPath <drive:\path\to\validation_key.pem>(Optional)
    -Key <RSA key as string>(Optional)
    (either -Key or -Key file should be used and validated)
```

This CmdLet saves the configuration prvided by the above parameters to Windows registry in the below path:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Blu\Config
```

Example:

```
Connect-Chef -Org https://chef.mycompany.com/organizations/myorganisation -Client Computer01 -KeyPath
```

If all the configuration is ok, you will recieve a message as:

```
14-03-2016 16:11:54 | OK    | Using Organization Name: myorganisation and Uri: https://chef.mycompany
14-03-2016 16:11:54 | OK    | Using Client Name: Computer01
14-03-2016 16:11:54 | OK    | Using Node Name: Computer01
14-03-2016 16:11:54 | OK    | Using RSA Key : c:\chef\client.pem
14-03-2016 16:11:54 | API   | Method : GET
14-03-2016 16:11:54 | API   | Client : Computer01
14-03-2016 16:11:54 | API   | organizationUri : https://chef.mycompany.com/organizations/myorganisat
14-03-2016 16:11:54 | API   | resource : clients/Computer01
14-03-2016 16:11:54 | API   | getUri : https://chef.mycompany.com/organizations/myorganisation/client
14-03-2016 16:11:54 | API   | XOpsProtocol(Computer01, https://chef.mycompany.com/organizations/myorg



 __          __    __
 \ \        / /_  / /_  __
  \ \      / __ \/ / / / /
   \ \    / /_/ / / /_/ /
    \_\  /_.___/_/\__,_/

14-03-2016 16:11:54 | OK    | Connected!
14-03-2016 16:11:54 | INFO  | Client Name: Computer01 / Organization: myorganisation
14-03-2016 16:11:54 | API   | Method : GET
14-03-2016 16:11:54 | API   | Client : Computer01
14-03-2016 16:11:54 | API   | organizationUri : https://chef.mycompany.com/organizations/myorganisat
14-03-2016 16:11:54 | API   | resource : nodes/Computer01
14-03-2016 16:11:54 | API   | getUri : https://chef.mycompany.com/organizations/myorganisation/nodes/
14-03-2016 16:11:54 | API   | XOpsProtocol(Computer01, https://chef.mycompany.com/organizations/myorg
14-03-2016 16:11:55 | OK    | Recieved Node json for: Computer01
```

Note: You can take a look at “HKEY_LOCAL_MACHINE\SOFTWARE\Blu\Config” and check if all the configuration are correct. You can also delete this key to let Connect-Chef CmdLet recreate it. This key is the only place where ChefAPI library is looking for connection settings. Files like client.rb and other Chef system files are not used by .Net Chef API client.

3.4 Using Chef API

Access to Chef API from PowerShell namespace is provided by **Use-ChefApi** CmdLet:

Parameters:

```
Use-ChefAPI
    -Select <Chef endpoint>(Optional)
    -Path <Api Path>(Optional)
    -Node <NodeName>(Optional)
    -Format <data format>(optional) [json, dictionary]
```

In this example we request the number of CPUs for Computer01 from Ohai data:

```
Use-ChefAPI -Select nodes/SBPLT240 -Path automatic/cpu/total
```

> **Warning:** Blu Chef API is a functional API client. That means API calls like “delete” or “post” are working as they meant to be. If you delete an object by Blu Chef API, it really gets deleted from the Chef server, so you need to use Blu Chef API with the same cautions as other Chef API clients.

3.5 API Graphical User Interface

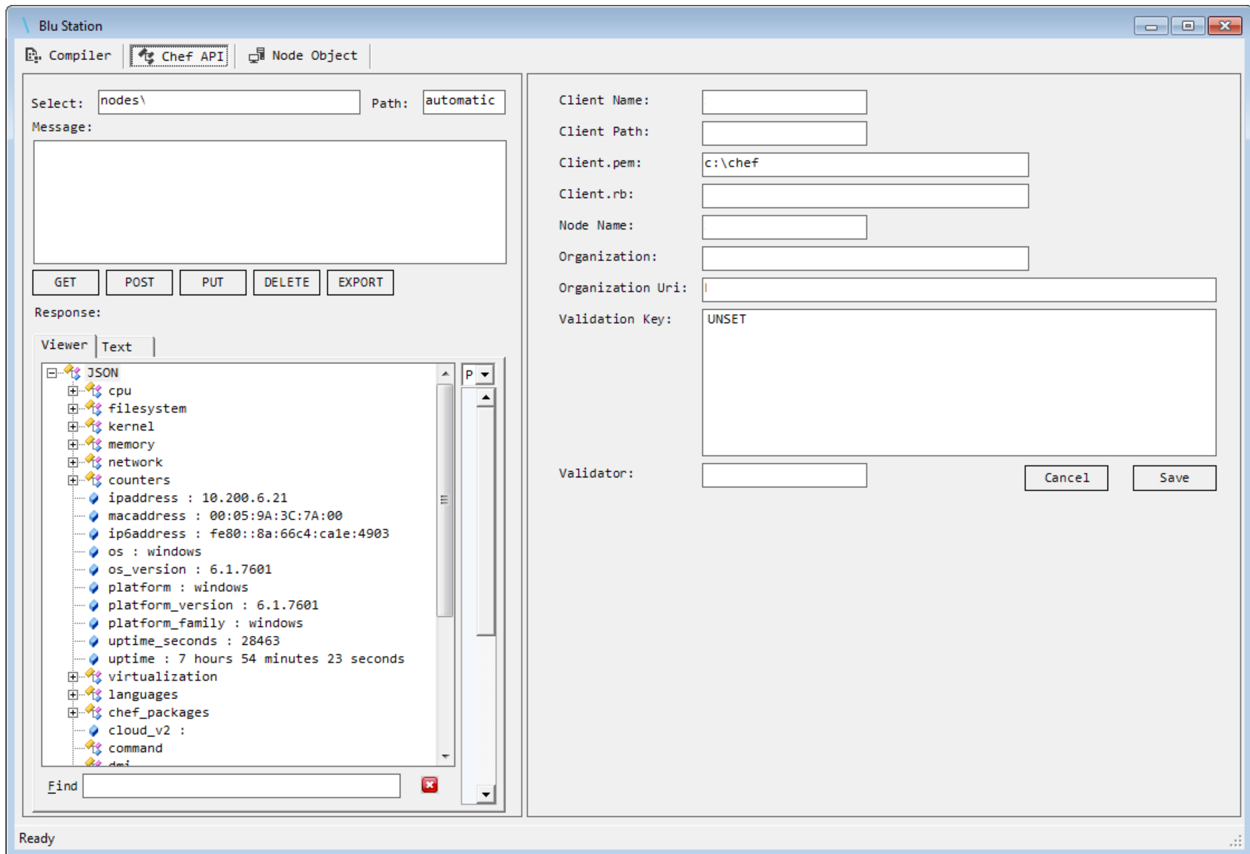
To start the Chef API graphical user interface, use Show-UI CmdLet without any parameters:

```
Show-UI
```

This UI provides a configurable GUI to Chef API (all configuration are saved and loaded from the same registry key) and an interface to experimental DSL transpiler.

3.5.1 Screenshots:

Chef API interface:



Transpiler interface:

CHAPTER 4

.Net Reference

4.1 BluAPI

4.2 BluLang

4.3 BluIpc

4.4 BluService

4.5 BluShell