Customizing with WebSphere CloudBurst, Part 3: Using script packages for customizing above and beyond patterns

Skill Level: Intermediate

Brian Stelzer (bsstelze@us.ibm.com)
Staff Software Engineer
IBM

Xin Peng Liu (xinpengl@cn.ibm.com)
Software Engineer
IBM

04 Nov 2009

Because every user scenario is unique, the IBM® WebSphere® CloudBurst™ Appliance has built-in features to help you configure and customize your IBM WebSphere Application Server environments. Part 3 of this series describes how to customize and enhance your deployed WebSphere Application Server environments using script packages.

Introduction

The IBM WebSphere CloudBurst Appliance is a new IBM hardware appliance that facilitates the creation and management of a private cloud environment. WebSphere CloudBurst patterns, which represent topology definitions for repeatable deployments that can be shared, play a key role in the creation of the private cloud. WebSphere CloudBurst contains several preloaded WebSphere Application Server patterns based on industry best practices. Not only does WebSphere CloudBurst provide these patterns to help you instantly build up virtual systems with different topologies, but it also enables you to customize your cloud to suit your business requirements.

A range of customization techniques are possible. You can:

- Customize the preloaded virtual images and patterns.
- Configure the deployed WebSphere Application Server environment directly using the administrative console.
- Define your own scripts to configure the deployed environment.

These and other techniques were highlighted in Part 1 of this series. This article focuses on script packages and covers:

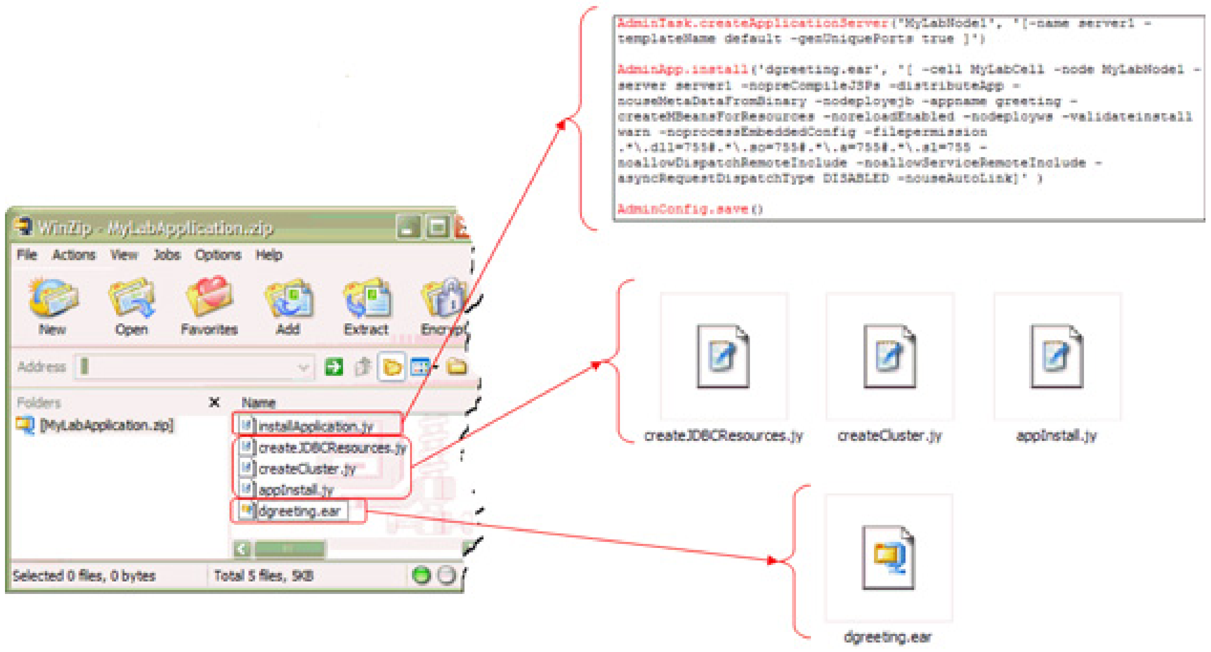
- The format and composition of a typical script package.
- How to upload a script package and manage it in the appliance.
- How to associate a script package with a pattern.
- How to transfer and run a script package in a virtual system.

Script packages

A script package is designed to customize the behavior of parts within a pattern. The intention of using script packages is to further enable you to customize your WebSphere Application Server environment beyond the customization provisions that are standard with WebSphere CloudBurst. A typical scenario might be to install a WebSphere Application Server application into a server or cluster environment rendered by WebSphere CloudBurst, as described in Part 1.

A script package is an archive (.zip) file containing artifacts that you want to be executed, and artifacts that you want to be executed upon. Script packages can be as simple as a single script file or as complex as containing an entirely new product. The content of a script package is not defined or restricted by WebSphere CloudBurst. The scripts included in a script package define the requirements for the script package. Figure 1 shows the content of a sample script package.

Figure 1. Typical content of a sample script package



```
AdminTask.createApplicationServer('MyLabNode1', '[-name server1 -
templateName default -genUniquePorts true ]')

AdminApp.install('dgreeting.ear', '[ -cell MyLabCell -node MyLabNode1 -
server server1 -nopreCompileJSPs -distributeApp -
nouseMetaDataFromBinary -nodeployejb -appname greeting -
createMBeansForResources -noreloadEnabled -nodeployws -validateinstall
warn -noprocessEmbeddedConfig -filepermission
.*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755 -
noallowDispatchRemoteInclude -noallowServiceRemoteInclude -
asyncRequestDispatchType DISABLED -nouseAutoLink]' )

AdminConfig.save()
```

As shown here, the sample script package is formed as a file called MyLabApplication.zip and is composed of two types of files:

- **J2EE™ application EAR file**, which represents an application to be installed.
- **Executable Jython script files** (with the file extension .jy), which contain the logic to perform the application installation to a single WebSphere Application Server instance or to a cluster of WebSphere Application Server instances.

Patterns are made up of virtual image parts, and each virtual image part gets its own virtual machine to run in when it is deployed. A script package is associated with a virtual image part. Upon deployment, the virtual image part becomes a virtual machine, and the script package associated with the virtual image part gets executed on that virtual machine. Associating a script package with a virtual image part is as easy as dragging and dropping the script package onto the virtual image part.

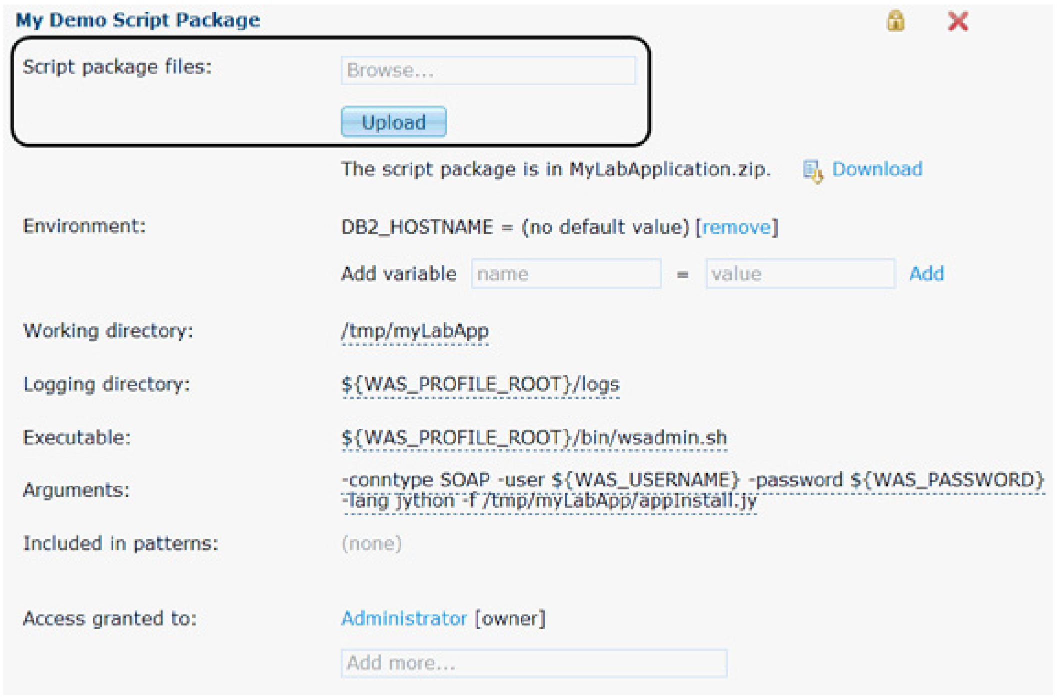
During the deployment process, WebSphere CloudBurst will first create and configure a virtual machine for each virtual image part. Once the topology is completely configured and running, WebSphere CloudBurst will SSH over as root context to the virtual machine for which the script package is targeted. WebSphere CloudBurst will then proceed to unzip the archive and execute the scripts defined within.

An end-to-end script package scenario

This section describes the steps involved in creating and deploying a script package into the cloud.

After you create an archive (.zip) file containing your executable script and supporting artifacts, you can create a script package in WebSphere CloudBurst:

1. Navigate to **Catalog => Script Packages**. To create a script package, click on the green plus icon. Figure 2 shows a script package named **My Demo Script Package**.

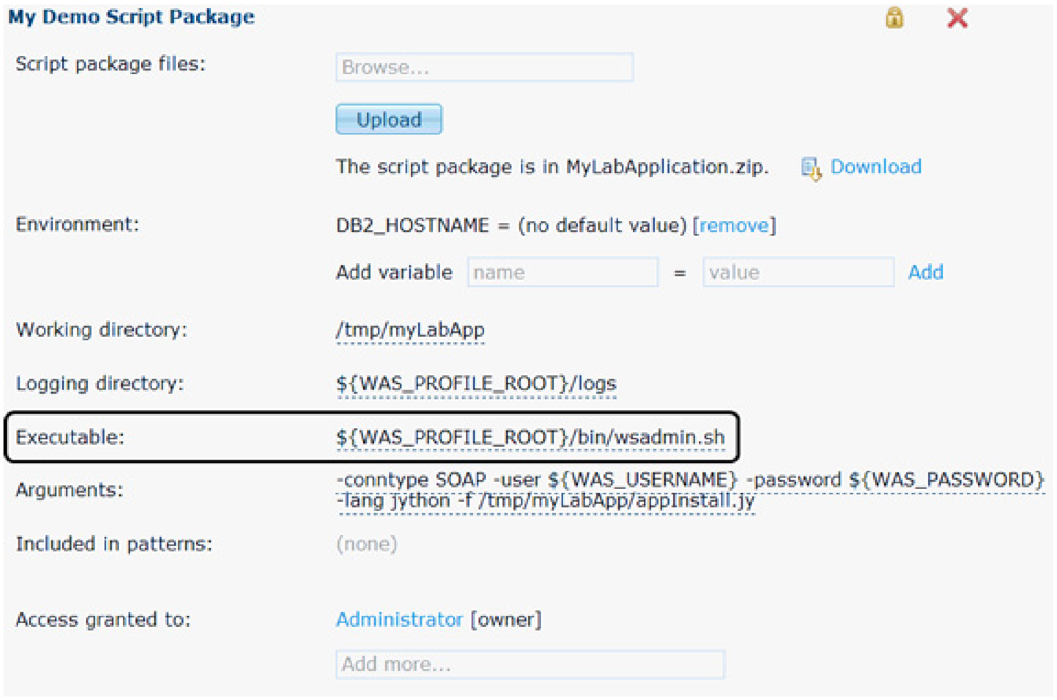
2. Upload your archive file to WebSphere CloudBurst. Click the **Browse...** button, choose your archive from your local file system, and then click **Upload**. The amount of time this operation takes depends on the size of your archive and the speed of your network connection. When this step completes, you will receive a successful confirmation message below the **Upload** button.
Figure 2. Upload .zip or .tar archive

My Demo Script Package

Script package files:	Browse... Upload
	The script package is in MyLabApplication.zip. Download
Environment:	DB2_HOSTNAME = (no default value) [remove]
	Add variable name = value Add
Working directory:	/tmp/myLabApp
Logging directory:	${WAS_PROFILE_ROOT}/logs
Executable:	${WAS_PROFILE_ROOT}/bin/wsadmin.sh
Arguments:	-conntype SOAP -user ${WAS_USERNAME} -password ${WAS_PASSWORD} -lang jython -f /tmp/myLabApp/appInstall.jy
Included in patterns:	(none)
Access granted to:	Administrator [owner]
	Add more...

3. Define the executable that you would like WebSphere CloudBurst to invoke. Because the intention of this example is to install an application, point your **Executable** field to the wsadmin.sh command in Figure 3. (The executable is not required to be wsadmin.sh. The executable can be any executable residing in the target environment or within the archive that

you uploaded.)
There are no artificial restrictions placed on script packages. Your script package can install applications, configure the WebSphere Application Server, tune the operating system, install entirely new products, and so on. Although there are no restrictions, there are best practices for employing their use, which are described in Part 1.

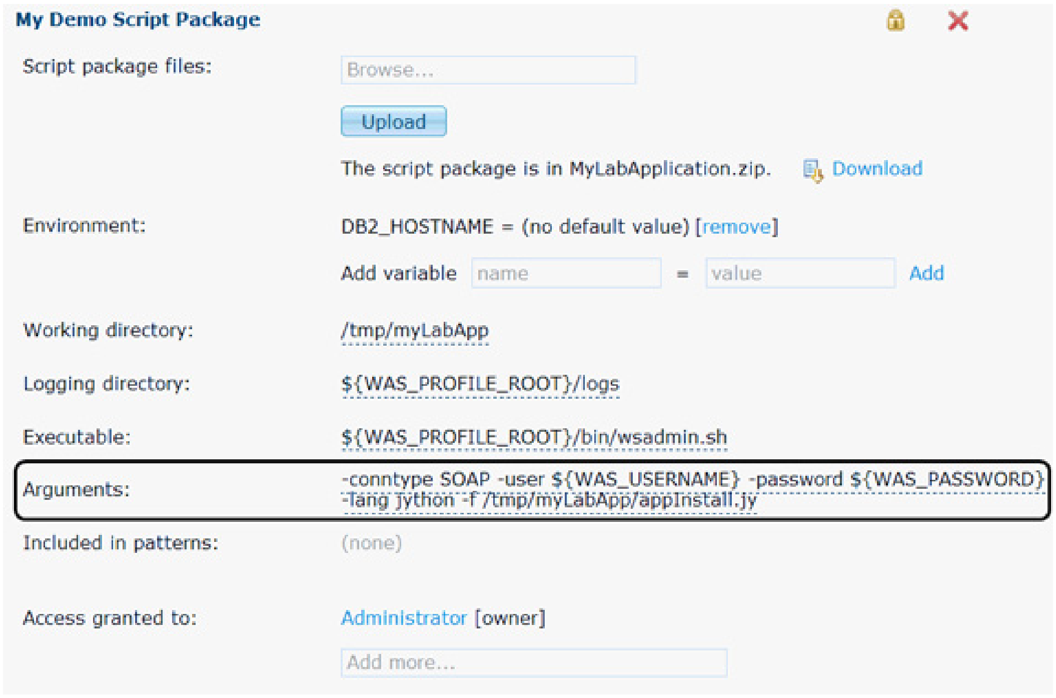
Figure 3. Define executable to run



My Demo Script Package

Script package files:	Browse...
	Upload
	The script package is in MyLabApplication.zip. Download
Environment:	DB2_HOSTNAME = (no default value) [remove]
	Add variable name = value Add
Working directory:	/tmp/myLabApp
Logging directory:	${WAS_PROFILE_ROOT}/logs
Executable:	${WAS_PROFILE_ROOT}/bin/wsadmin.sh
Arguments:	-conntype SOAP -user ${WAS_USERNAME} -password ${WAS_PASSWORD} -lang jython -f /tmp/myLabApp/appInstall.jy
Included in patterns:	(none)
Access granted to:	Administrator [owner]
	Add more...

4. Define your arguments. The **Arguments** field is highlighted in Figure 4. Arguments defined here will be passed to the executable when it is run. In this example, the archive contains two files, an EAR file and a Jython script, **appInstall.jy**. The script file installs the EAR file included in the archive.
Figure 4. Define the arguments to executable

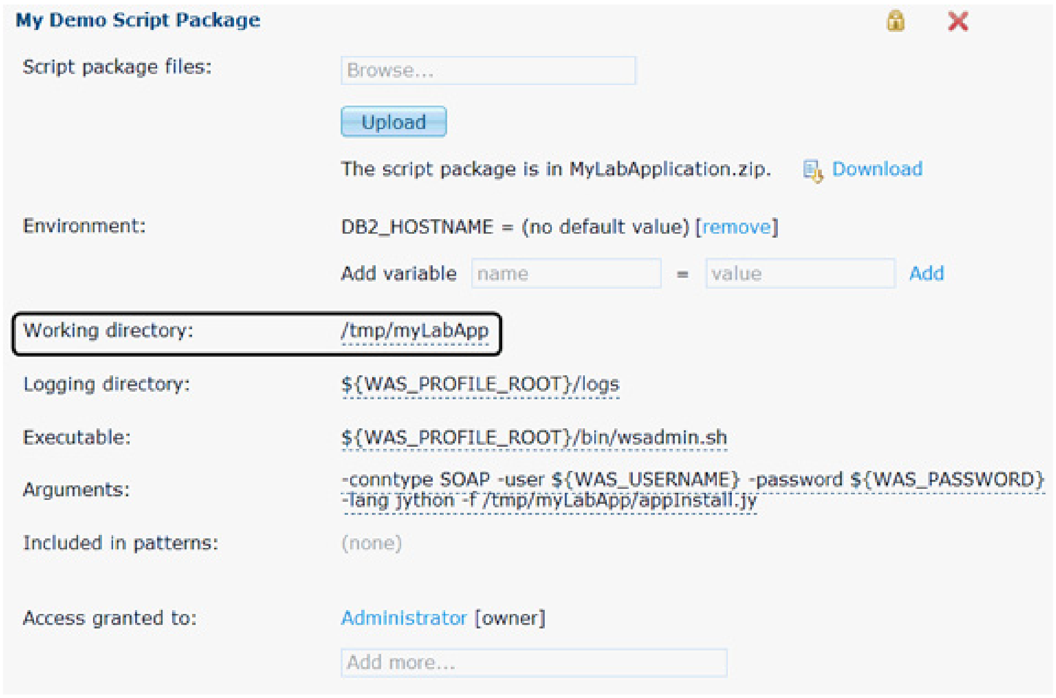


My Demo Script Package

Script package files:	Browse...
	Upload
	The script package is in MyLabApplication.zip. Download
Environment:	DB2_HOSTNAME = (no default value) [remove]
	Add variable name = value Add
Working directory:	/tmp/myLabApp
Logging directory:	${WAS_PROFILE_ROOT}/logs
Executable:	${WAS_PROFILE_ROOT}/bin/wsadmin.sh
Arguments:	-conntype SOAP -user ${WAS_USERNAME} -password ${WAS_PASSWORD} -lang jython -f /tmp/myLabApp/appInstall.jy
Included in patterns:	(none)
Access granted to:	Administrator [owner]
	Add more...

5. Define your working directory. Figure 5 highlights the **Working directory** field. The value of the Working directory field indicates the directory into which the contents of the archive in your script package will be unzipped.

Figure 5. Define working directory



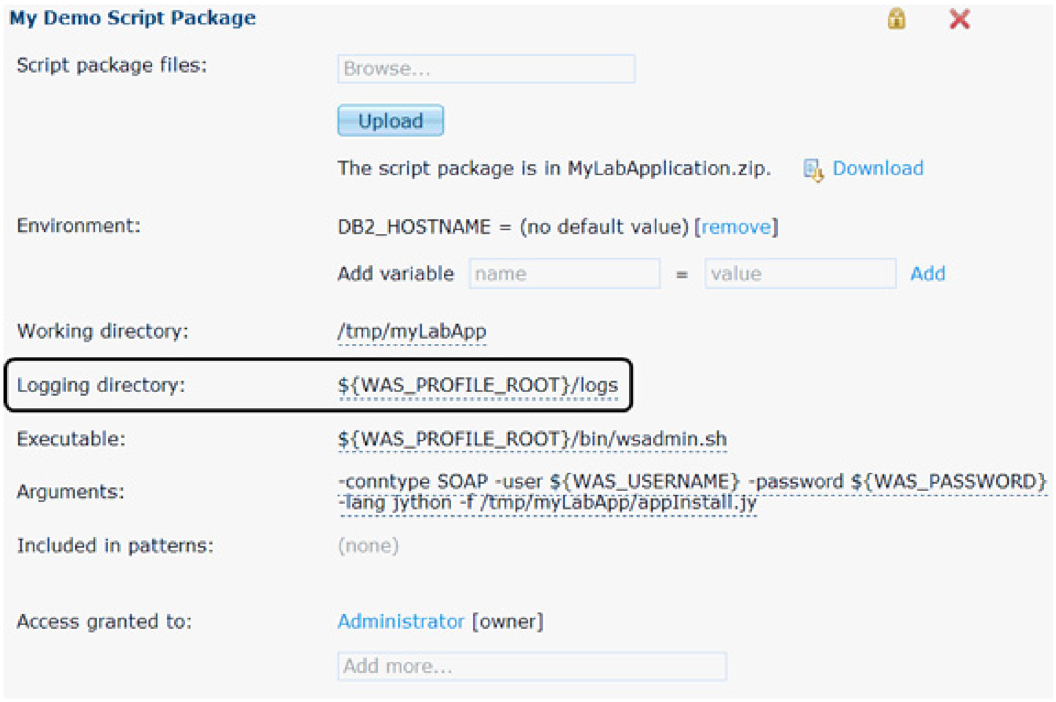
My Demo Script Package

Script package files:	Browse...
	Upload
	The script package is in MyLabApplication.zip. Download
Environment:	DB2_HOSTNAME = (no default value) [remove]
	Add variable name = value Add
Working directory:	/tmp/myLabApp
Logging directory:	${WAS_PROFILE_ROOT}/logs
Executable:	${WAS_PROFILE_ROOT}/bin/wsadmin.sh
Arguments:	-conntype SOAP -user ${WAS_USERNAME} -password ${WAS_PASSWORD} -lang jython -f /tmp/myLabApp/appInstall.jy
Included in patterns:	(none)
Access granted to:	Administrator [owner]
	Add more...

6. The **Logging directory** field shown in Figure 6 tells WebSphere

CloudBurst the directory that is used to store logs produced by the execution of the script package; this is not input to the executable to tell it where to place logs. Because you define the log location here, you will be able to use WebSphere CloudBurst to view these logs (which will be shown later).

Figure 6. Define logging directory



My Demo Script Package

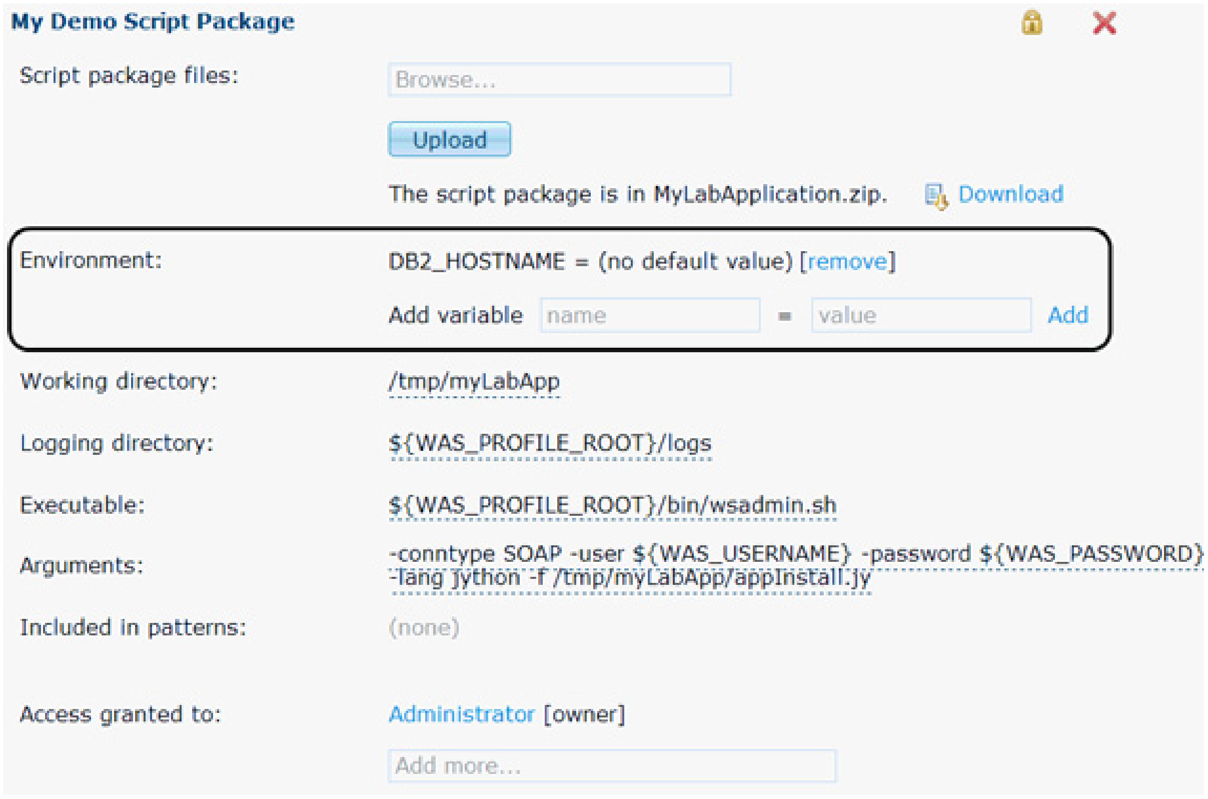
Script package files:	Browse...
	Upload
	The script package is in MyLabApplication.zip. Download
Environment:	DB2_HOSTNAME = (no default value) [remove]
	Add variable name = value Add
Working directory:	/tmp/myLabApp
Logging directory:	${WAS_PROFILE_ROOT}/logs
Executable:	${WAS_PROFILE_ROOT}/bin/wsadmin.sh
Arguments:	-conntype SOAP -user ${WAS_USERNAME} -password ${WAS_PASSWORD} -lang jython -f /tmp/myLabApp/appInstall.jy
Included in patterns:	(none)
Access granted to:	Administrator [owner]
	Add more...

You have likely noticed the use of variables in many of the script package fields throughout this example. There are two types of variables:

- User defined variables are defined in the **Environment** field (Figure 7). These variables are specific to the script package in which they are defined. Once a variable has been defined in the Environment field, it can be used in the scripts defined within the archive or in the definition of the script package itself. You can enter a key and default value or just a key. If you enter the key only (or even if you just enter a value), you will be given the opportunity to enter or override that value at pattern creation or pattern deployment time. This example defines a DB2_HOSTNAME variable with no associated value, so the value will need to be defined before deployment.
- WebSphere CloudBurst defines its own set of variables and makes them available to scripts defined within the archive using the `os.environ['…']` command (that is, `os.environ['CELL_NAME']`). These variables are also available in the definition of the script package. The variables are common to all script packages, but the values are

unique to the context in which the script package is being executed. To help explain this point, take a look at the **Arguments** field in Figure 7. Notice the use of ${WAS_PASSWORD}. At deployment time, you will be required to enter a password for the environment. The password you enter will be used during the execution of the script package. The non-preferred approach would be to hardcode the password, in which case your script package is now tied to deployments using the same password.

Figure 7. Define any environment variables



My Demo Script Package

Script package files:	Browse...
	Upload
	The script package is in MyLabApplication.zip. Download
Environment:	DB2_HOSTNAME = (no default value) [remove]
	Add variable name = value Add
Working directory:	/tmp/myLabApp
Logging directory:	${WAS_PROFILE_ROOT}/logs
Executable:	${WAS_PROFILE_ROOT}/bin/wsadmin.sh
Arguments:	-conntype SOAP -user ${WAS_USERNAME} -password ${WAS_PASSWORD} -lang jython -f /tmp/myLabApp/appInstall.jy
Included in patterns:	(none)
Access granted to:	Administrator [owner]
	Add more...

A subset of the most commonly used variables are shown in Table 1. (A complete list of all variables defined by WebSphere CloudBurst can be found in the WebSphere CloudBurst Information Center.) Keep in mind that you are not permitted to create user defined variables with the same names as WebSphere CloudBurst defined variables.

Table 1. Subset of predefined WebSphere Application Server environment variables

Variable	Description
CELL_NAME	Cell name
NODE_NAME	Node name
WAS_INSTALL_ROOT	Installation root (for example:

	`C:/WebSphere/AppServer`)
WAS_PROFILE_ROOT	Profile root (example: `C:/WebSphere/AppServer/profiles/DefaultProfile`)
PROFILE_NAME	Profile short name (example: `DefaultProfile`)
HOSTNAME	Host name of virtual machine hosting the WebSphere Application Server instance
WAS_USERNAME	User name
WAS_PASSWORD	Password

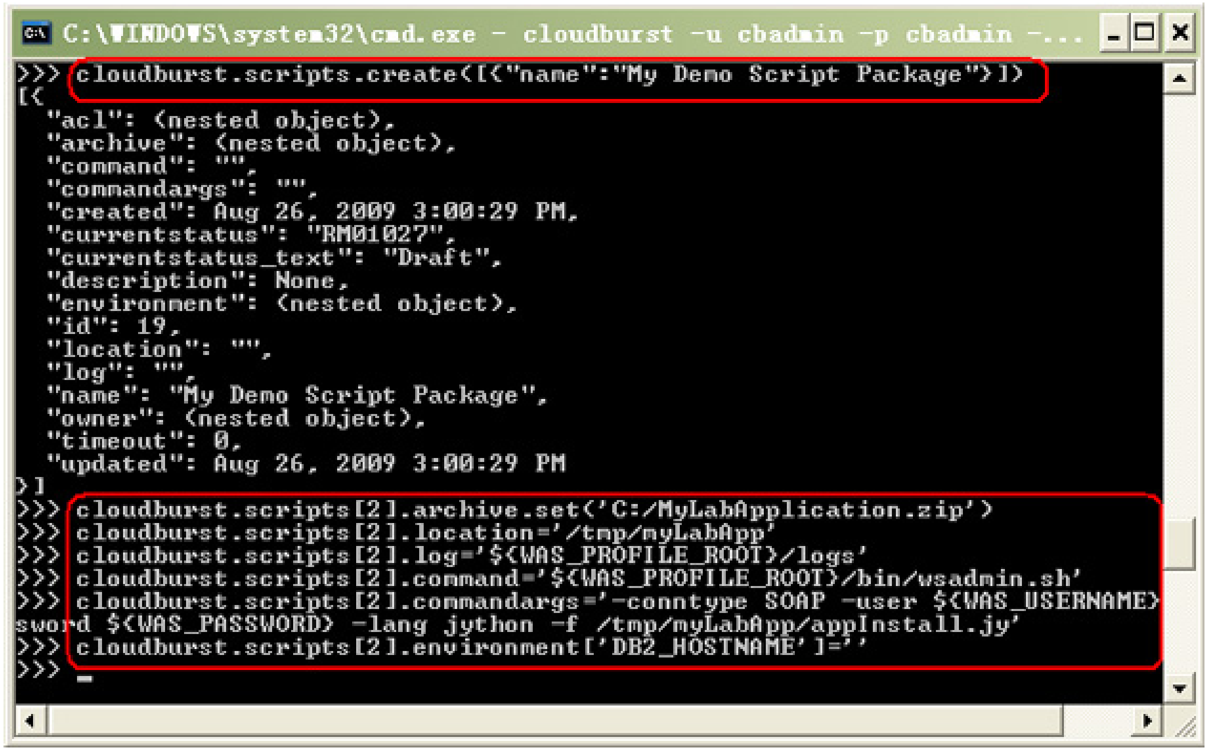
In addition to the Web console, WebSphere CloudBurst provides a command line tool and HTTP REST interfaces to upload script packages, and to manipulate script packages already maintained in the appliance.

There are two script package-related objects defined by the command line tool:

- The scripts object represents the container of script packages currently residing on the appliance, and provides attributes and methods to create, delete, iterate over, list, and search for these packages.
- The script object represents a single script package uploaded to WebSphere CloudBurst, and provides attributes and methods to configure the specific script package.

Figure 8 illustrates how to create the My Demo Script Package script package (defined in the previous figures using the Web console) using the command-line tool.

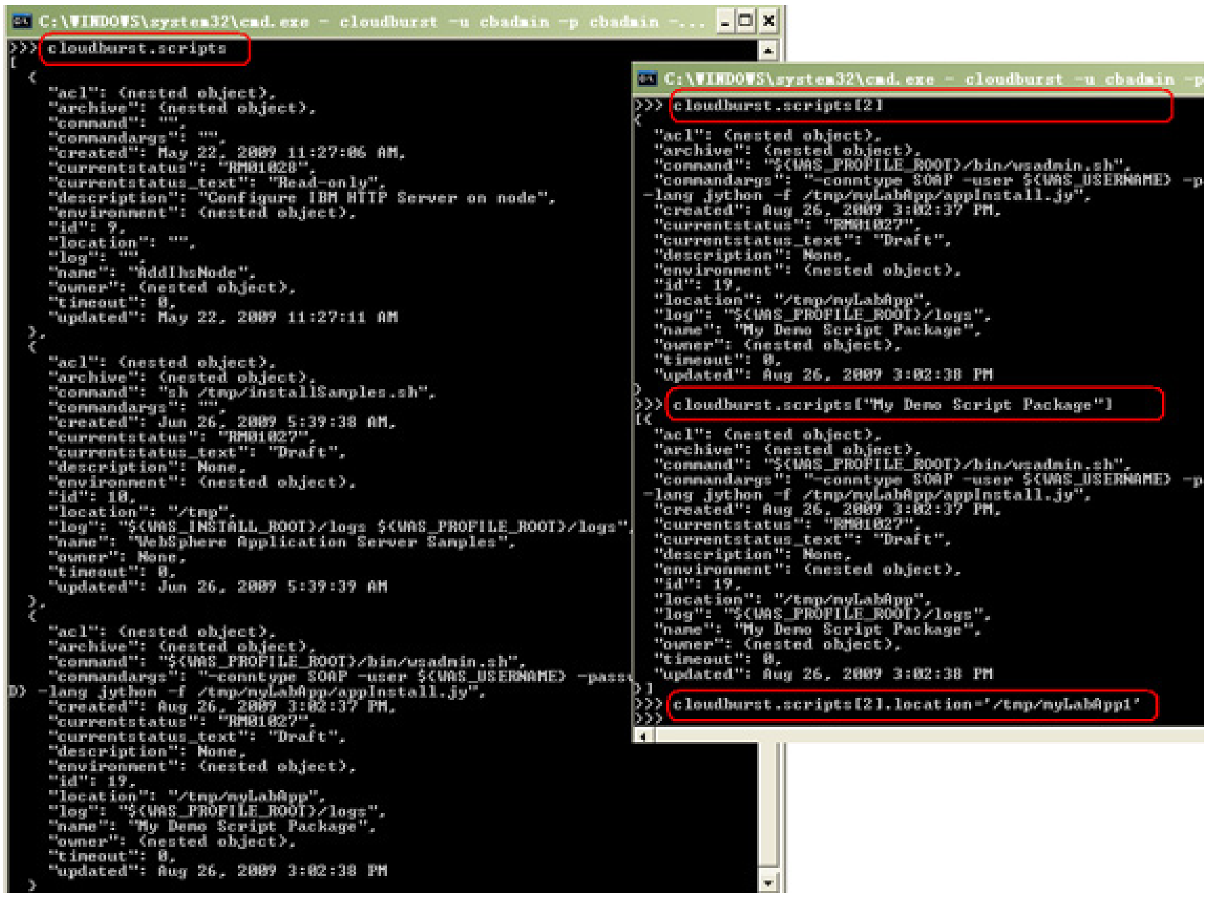
Figure 8. Command line for creating My Demo Script Package


C:\WINDOWS\system32\cmd.exe - cloudburst -u cbadmin -p cbadmin -...

```
>>> cloudburst.scripts.create([{"name":"My Demo Script Package"}])
[{
  "acl": (nested object),
  "archive": (nested object),
  "command": "",
  "commandargs": "",
  "created": Aug 26, 2009 3:00:29 PM,
  "currentstatus": "RM01027",
  "currentstatus_text": "Draft",
  "description": None,
  "environment": (nested object),
  "id": 19,
  "location": "",
  "log": "",
  "name": "My Demo Script Package",
  "owner": (nested object),
  "timeout": 0,
  "updated": Aug 26, 2009 3:00:29 PM
}]
>>> cloudburst.scripts[2].archive.set('C:/MyLabApplication.zip')
>>> cloudburst.scripts[2].location='/tmp/myLabApp'
>>> cloudburst.scripts[2].log='${WAS_PROFILE_ROOT}/logs'
>>> cloudburst.scripts[2].command='${WAS_PROFILE_ROOT}/bin/wsadmin.sh'
>>> cloudburst.scripts[2].commandargs='-conntype SOAP -user ${WAS_USERNAME}
sword ${WAS_PASSWORD} -lang jython -f /tmp/myLabApp/appInstall.jy'
>>> cloudburst.scripts[2].environment['DB2_HOSTNAME']=''
>>> _
```

Figure 9 shows typical commands for listing query and update properties of script packages contained in the appliance.

Figure 9. List, query, and update property to script packages

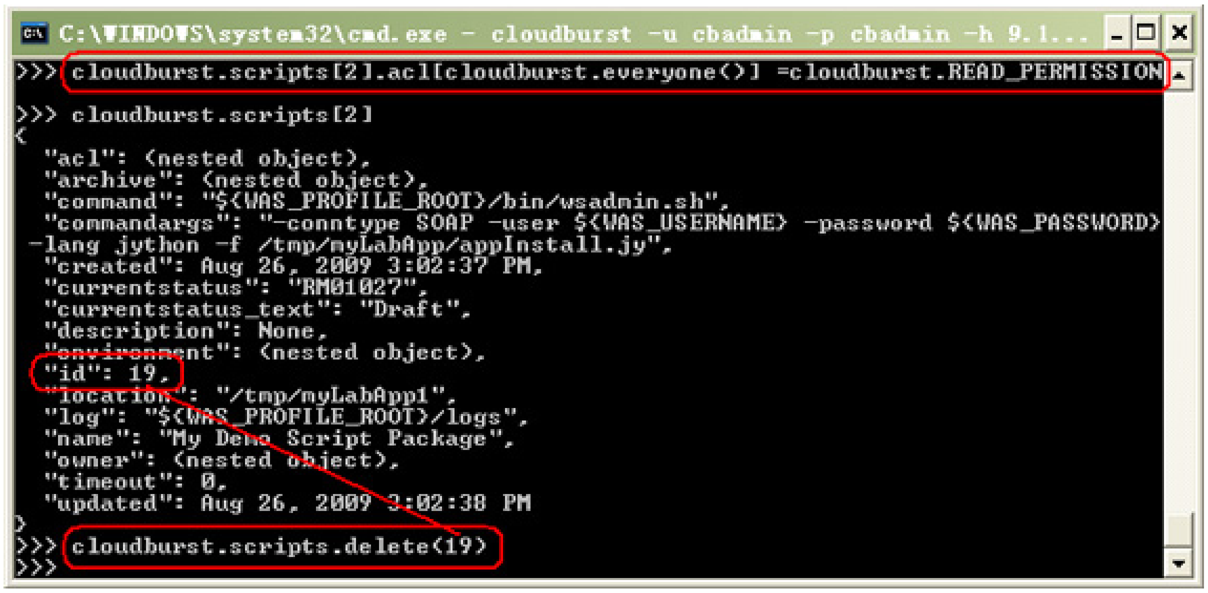

C:\WINDOWS\system32\cmd.exe - cloudburst -u cbadmin -p cbadmin -...
C:\WINDOWS\system32\cmd.exe - cloudburst -u cbadmin -p

```
>>> cloudburst.scripts
[
  {
    "acl": (nested object),
    "archive": (nested object),
    "command": "",
    "commandargs": "",
    "created": May 22, 2009 11:27:06 AM,
    "currentstatus": "RM01028",
    "currentstatus_text": "Read-only",
    "description": "Configure IBM HTTP Server on node",
    "environment": (nested object),
    "id": 9,
    "location": "",
    "log": "",
    "name": "AddIhsNode",
    "owner": (nested object),
    "timeout": 0,
    "updated": May 22, 2009 11:27:11 AM
  },
  {
    "acl": (nested object),
    "archive": (nested object),
    "command": "sh /tmp/installSamples.sh",
    "commandargs": "",
    "created": Jun 26, 2009 5:39:38 AM,
    "currentstatus": "RM01027",
    "currentstatus_text": "Draft",
    "description": None,
    "environment": (nested object),
    "id": 10,
    "location": "/tmp",
    "log": "${WAS_INSTALL_ROOT}/logs ${WAS_PROFILE_ROOT}/logs",
    "name": "WebSphere Application Server Samples",
    "owner": None,
    "timeout": 0,
    "updated": Jun 26, 2009 5:39:39 AM
  },
  {
    "acl": (nested object),
    "archive": (nested object),
    "command": "${WAS_PROFILE_ROOT}/bin/wsadmin.sh",
    "commandargs": "-conntype SOAP -user ${WAS_USERNAME} -pass
D} -lang jython -f /tmp/myLabApp/appInstall.jy",
    "created": Aug 26, 2009 3:02:37 PM,
    "currentstatus": "RM01027",
    "currentstatus_text": "Draft",
    "description": None,
    "environment": (nested object),
    "id": 19,
    "location": "/tmp/myLabApp",
    "log": "${WAS_PROFILE_ROOT}/logs",
    "name": "My Demo Script Package",
    "owner": (nested object),
    "timeout": 0,
    "updated": Aug 26, 2009 3:02:38 PM
  }
```

```
>>> cloudburst.scripts[2]
{
  "acl": (nested object),
  "archive": (nested object),
  "command": "${WAS_PROFILE_ROOT}/bin/wsadmin.sh",
  "commandargs": "-conntype SOAP -user ${WAS_USERNAME} -p
 -lang jython -f /tmp/myLabApp/appInstall.jy",
  "created": Aug 26, 2009 3:02:37 PM,
  "currentstatus": "RM01027",
  "currentstatus_text": "Draft",
  "description": None,
  "environment": (nested object),
  "id": 19,
  "location": "/tmp/myLabApp",
  "log": "${WAS_PROFILE_ROOT}/logs",
  "name": "My Demo Script Package",
  "owner": (nested object),
  "timeout": 0,
  "updated": Aug 26, 2009 3:02:38 PM
}
>>> cloudburst.scripts["My Demo Script Package"]
[{
  "acl": (nested object),
  "archive": (nested object),
  "command": "${WAS_PROFILE_ROOT}/bin/wsadmin.sh",
  "commandargs": "-conntype SOAP -user ${WAS_USERNAME} -p
 -lang jython -f /tmp/myLabApp/appInstall.jy",
  "created": Aug 26, 2009 3:02:37 PM,
  "currentstatus": "RM01027",
  "currentstatus_text": "Draft",
  "description": None,
  "environment": (nested object),
  "id": 19,
  "location": "/tmp/myLabApp",
  "log": "${WAS_PROFILE_ROOT}/logs",
  "name": "My Demo Script Package",
  "owner": (nested object),
  "timeout": 0,
  "updated": Aug 26, 2009 3:02:38 PM
}]
>>> cloudburst.scripts[2].location='/tmp/myLabApp1'
>>>
```

You can also leverage the command line interfaces to change the access control configurations of script packages, and even delete a package from the appliance using its ID, as shown in Figure 10.

Figure 10. Change ACL and delete script package


C:\WINDOWS\system32\cmd.exe - cloudburst -u cbadmin -p cbadmin -h 9.1...

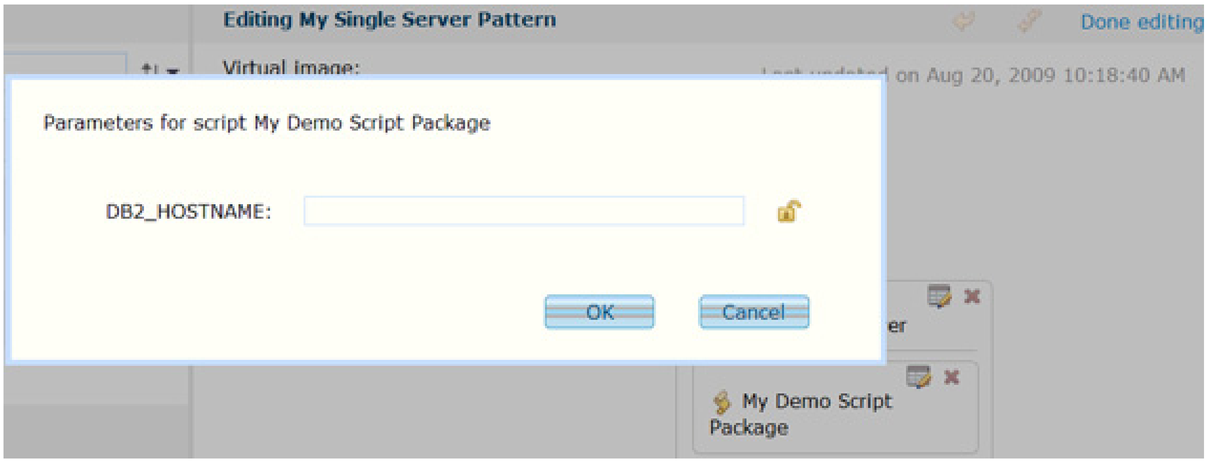
```
>>> cloudburst.scripts[2].acl[cloudburst.everyone()] =cloudburst.READ_PERMISSION

>>> cloudburst.scripts[2]
{
  "acl": (nested object),
  "archive": (nested object),
  "command": "${WAS_PROFILE_ROOT}/bin/wsadmin.sh",
  "commandargs": "-conntype SOAP -user ${WAS_USERNAME} -password ${WAS_PASSWORD}
 -lang jython -f /tmp/myLabApp/appInstall.jy",
  "created": Aug 26, 2009 3:02:37 PM,
  "currentstatus": "RM01027",
  "currentstatus_text": "Draft",
  "description": None,
  "environment": (nested object),
  "id": 19,
  "location": "/tmp/myLabApp1",
  "log": "${WAS_PROFILE_ROOT}/logs",
  "name": "My Demo Script Package",
  "owner": (nested object),
  "timeout": 0,
  "updated": Aug 26, 2009 3:02:38 PM
}
>>> cloudburst.scripts.delete(19)
>>>
```

Details of pattern creation is beyond the scope of this article, but let's look at a few pertinent points with respect to script packages.

When you create your pattern using a script package that has user defined variables, you will be given the opportunity to define values for keys that do not have a value, or to change the value of those variables that already have a value defined (Figure 11).

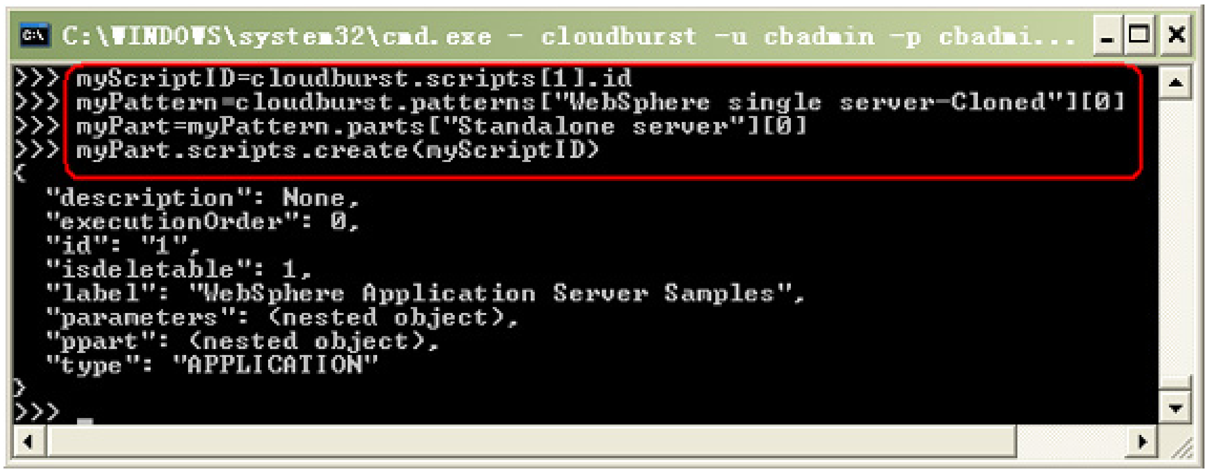
As mentioned earlier, patterns are made up of virtual image parts and each of those virtual image parts will get its own virtual machine to run in when it is deployed. Script packages should be associated with the virtual machine on which you want that script package to be executed. In the case of pattern creation through the user interface, you would drag and drop the script package onto the virtual image part.

Figure 11. Assign value to user defined variables


Editing My Single Server Pattern
Done editing
Virtual image:
Last updated on Aug 20, 2009 10:18:40 AM
Parameters for script My Demo Script Package
DB2_HOSTNAME:
OK
Cancel
My Demo Script Package

It is also possible to associate a script package with a virtual image part using the command line tool. You can do this by accessing the pattern object, but not by using the script object. Figure 12 shows a script package being associated with a pattern.

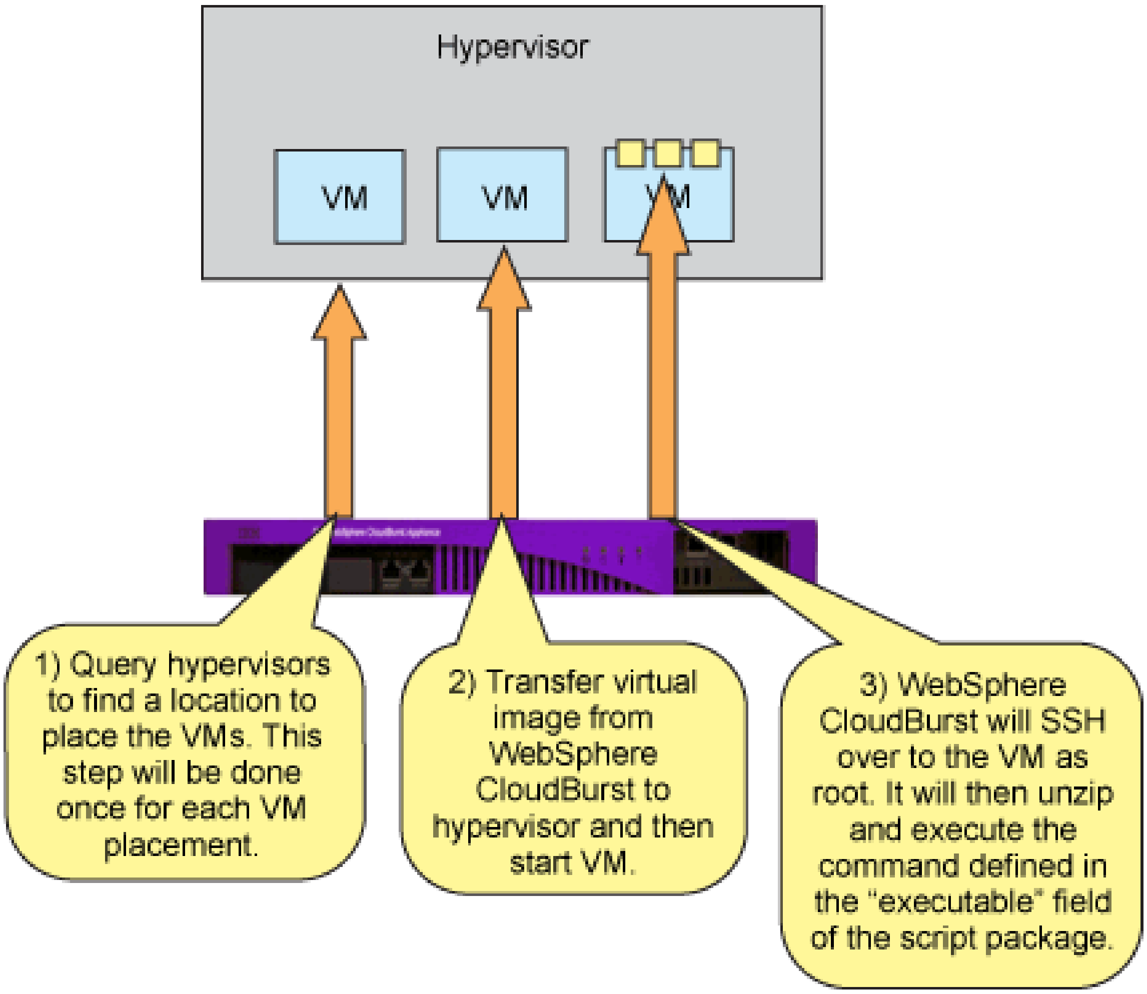
Figure 12. Associate a script package to a virtual image part using command-line interfaces



```
C:\WINDOWS\system32\cmd.exe - cloudburst -u cbadmin -p cbadmi...
>>> myScriptID=cloudburst.scripts[1].id
>>> myPattern=cloudburst.patterns["WebSphere single server-Cloned"][0]
>>> myPart=myPattern.parts["Standalone server"][0]
>>> myPart.scripts.create(myScriptID)
{
  "description": None,
  "executionOrder": 0,
  "id": "1",
  "isdeletable": 1,
  "label": "WebSphere Application Server Samples",
  "parameters": (nested object),
  "ppart": (nested object),
  "type": "APPLICATION"
}
>>> _
```

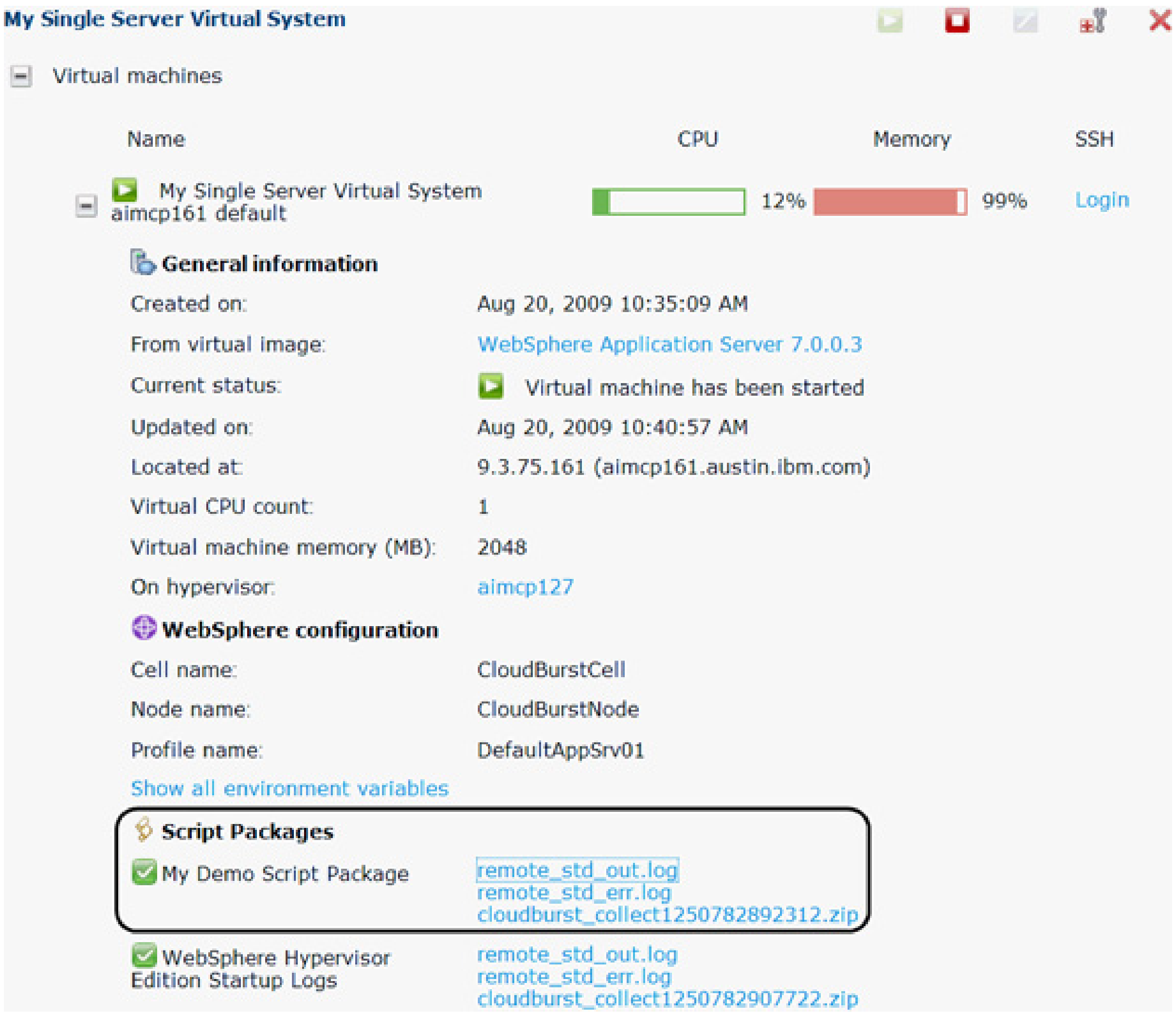
During the deployment process, WebSphere CloudBurst first creates and configures a virtual machine for each virtual image part. Once the topology is completely configured and running, WebSphere CloudBurst will SSH over as root context to the virtual machine to which the script package is targeted, and unzip the archive into the location you defined in the **Working directory** field of the script package. Next, it will run the command defined in the **Executable** field, passing in the arguments defined in the **Arguments** field. Figure 13 shows a simple flow of these steps.

Figure 13. Script package execution flow


Hypervisor
VM
VM
VM
1) Query hypervisors to find a location to place the VMs. This step will be done once for each VM placement.
2) Transfer virtual image from WebSphere CloudBurst to hypervisor and then start VM.
3) WebSphere CloudBurst will SSH over to the VM as root. It will then unzip and execute the command defined in the "executable" field of the script package.

After a script package runs, the logs are gathered up by WebSphere CloudBurst and made available through the virtual systems view, if you defined a log location in the **Logging directory** of the script package. Figure 14 highlights the location of the script package logs. There will be separate logs for each script package executed.

Figure 14. Script package logs



My Single Server Virtual System

Virtual machines

Name	CPU	Memory	SSH
My Single Server Virtual System aimcp161 default	12%	99%	Login

General information

Created on:	Aug 20, 2009 10:35:09 AM
From virtual image:	WebSphere Application Server 7.0.0.3
Current status:	Virtual machine has been started
Updated on:	Aug 20, 2009 10:40:57 AM
Located at:	9.3.75.161 (aimcp161.austin.ibm.com)
Virtual CPU count:	1
Virtual machine memory (MB):	2048
On hypervisor:	aimcp127

WebSphere configuration

Cell name:	CloudBurstCell
Node name:	CloudBurstNode
Profile name:	DefaultAppSrv01

Show all environment variables

Script Packages

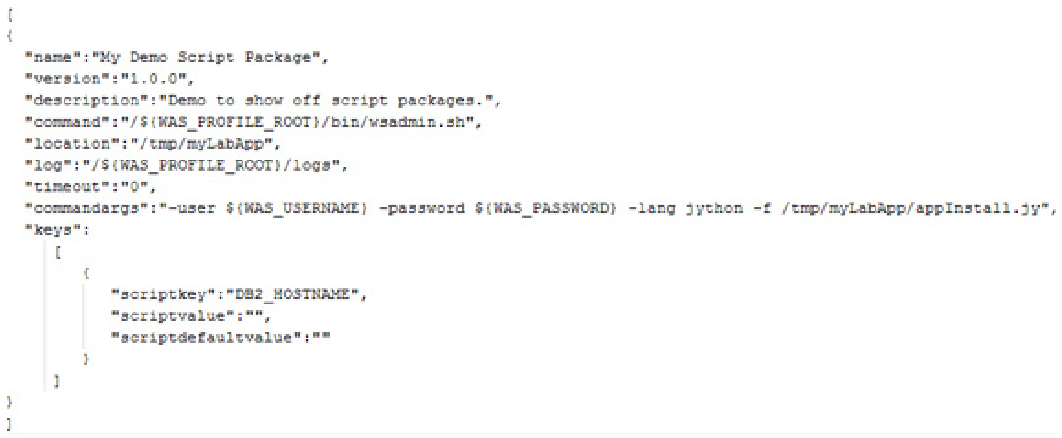
My Demo Script Package	remote_std_out.log remote_std_err.log cloudburst_collect1250782892312.zip
WebSphere Hypervisor Edition Startup Logs	remote_std_out.log remote_std_err.log cloudburst_collect1250782907722.zip

Other considerations

- **Prime script package using cbscript.json**
  There might be times when you would like to harden the configuration of a script package. The script package format enables you to add a configuration file called **cbscript.json** to the root directory of the archive. This configuration file contains your script package definition. By adding cbscript.json directly in your archive, you are relieved of having to enter this information when you create the script package. You might want to do this to avoid typographical errors on the administrative console or to share the script package among appliances.

  Figure 15 shows cbscript.json for the example script package above.

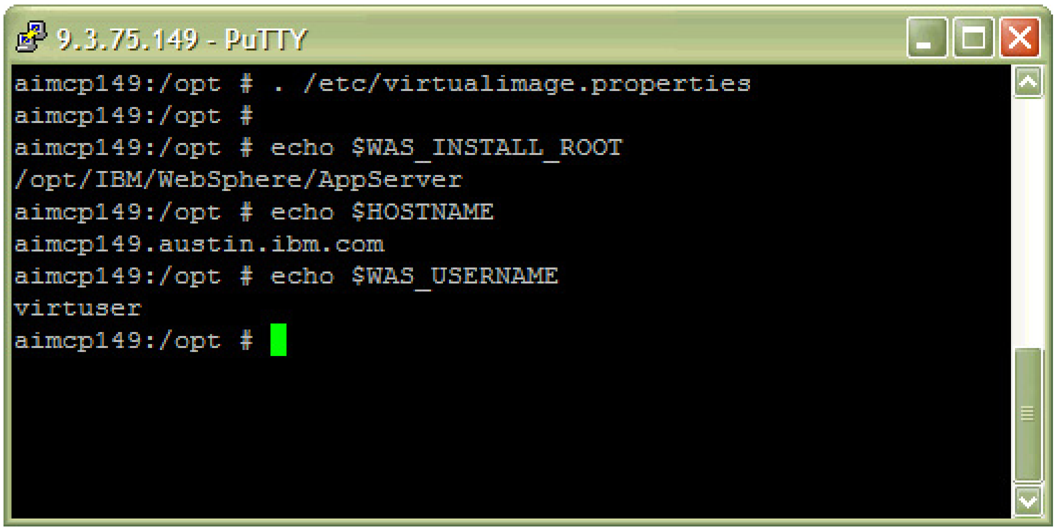
  Figure 15. Sample cbscript.json file



```
[
{
  "name":"My Demo Script Package",
  "version":"1.0.0",
  "description":"Demo to show off script packages.",
  "command":"/${WAS_PROFILE_ROOT}/bin/wsadmin.sh",
  "location":"/tmp/myLabApp",
  "log":"/${WAS_PROFILE_ROOT}/logs",
  "timeout":"0",
  "commandargs":"-user ${WAS_USERNAME} -password ${WAS_PASSWORD} -lang jython -f /tmp/myLabApp/appInstall.jy",
  "keys":
     [
        {
           "scriptkey":"DB2_HOSTNAME",
           "scriptvalue":"",
           "scriptdefaultvalue":""
        }
     ]
}
]
```

- **Script package environment**
  When executing scripts defined within the script package, WebSphere CloudBurst has access to many predefined environment variables. Table 1 shows a subset of these. The predefined variables can be accessed by sourcing the /etc/virtualimage.properties file on the virtual machine in which you want access to the variables. After sourcing the file, you can make use of all the variables WebSphere CloudBurst has defined. To source the properties file, you will need to enter `. /etc/virtualimage.properties`. Figure 16 shows an example of sourcing the properties file, followed by a few echo statements to show that the predefined variables are now available.

Figure 16. Sourcing /etc/virtualimage.properties example



```
aimcp149:/opt # . /etc/virtualimage.properties
aimcp149:/opt #
aimcp149:/opt # echo $WAS_INSTALL_ROOT
/opt/IBM/WebSphere/AppServer
aimcp149:/opt # echo $HOSTNAME
aimcp149.austin.ibm.com
aimcp149:/opt # echo $WAS_USERNAME
virtuser
aimcp149:/opt #
```

Summary

Script packages provide you with the capability to further enhance and customize the behavior of your WebSphere Application Server environments. You can customize settings as small as a single WebSphere Application Server attribute, or as large as installing and configuring an entirely new product. As you can see, the script package concept was designed to be generic and flexible, and puts no limit on what you can do with script packages. This flexibility enables the script package to be usable in any scenario where additional customization or enhancement is needed above and beyond what is available from a pattern.

Resources

Learn

- Customizing with WebSphere CloudBurst (series)
    - Part 1: Creating highly customized private clouds
    - Part 2: Using WebSphere CloudBurst to customize a WebSphere middleware environment
- Cloud computing for the enterprise (series)
    - Part 1: Capturing the cloud
    - Part 2: WebSphere sMash and DB2 Express-C on the Amazon EC2 public cloud
    - Part 3: Using WebSphere CloudBurst to create private clouds
- WebSphere CloudBurst Appliance product information
- Cloud Computing Journal
- Is there value in cloud computing?
- IBM developerWorks WebSphere

Discuss

- Space: WebSphere Cloud Computing for Developers
- Space: Cloud Computing Central
- YouTube: WebSphere CloudBurst Appliance videos
- Forum: WebSphere CloudBurst Appliance Forum
- Blog: A view from the clouds: Cloud computing for the WebSphere developer
- Follow us on Twitter

About the authors

Brian Stelzer
Brian Stelzer is a Staff Software Engineer in IBM’s Early Programs team working on the WebSphere CloudBurst beta program. In this assignment he works directly with the customer on pre-release versions of WebSphere CloudBurst. Prior to this assignment, Brian designed and implemented migration solutions for the WebSphere

Application Server line of products. Brian has over 9 years of experience in software development.

Xin Peng Liu

Xin Peng Liu is a Staff Software Engineer working at the SOA Design Center in IBM’s China Software Development Lab. Upon Joining IBM, he has been experienced with SOA-IF, SOA policy, SOA governance, and enterprise cloud-enabled SOA solution engineering. His interests include J2EE, SOA, MDA/MDD, AOP, Semantic Web, Cloud Computing, and more. He is currently performing design and development work related to WebSphere CloudBurst-based SOA solution enablement and a customer technical pilot.