Managing ArcGIS Server 10.1 with Python

Spring NEARC - UMass Amherst
May 14, 2013

Sean Sweeney
City of Cambridge
@hiker4k

What I want to do

1. Stop all services then restart only those that were previously started.

2. Stop a select list of services then restart them.

Stop all services then restart only those that were previously started.

Stopping and starting through Manager is cumbersome for lots of services.

Sometimes we have services that are stopped for a reason, especially on our development server. Keeping track of these while stopping and starting through Manager is a pain.

Stop a select list of services then restart them.

We have five SDE databases. If we are doing maintenance on only one then we only need to stop services that reference that one.

The "old" way of doing things

```
AGSSOM rover -list

    Service Status:
    MapServer 'AddressDashboard': Stopped
    MapServer 'GISEmbeddedLayers': Started

AGSSOM rover -stop GISEmbeddedLayers

AGSSOM rover -start GISEmbeddedLayers

AGSSOM rover -stop '*all*'
```

http://arcscripts.esri.com/details.asp?dbid=16293

In 10.0 we used the AGSSOM ArcScript code and a patchwork of manual and scripted steps.

10.1 Server command line utilities

Convert Cache Storage Format utility
Create Cache Schema utility
Create Service utility
Delete Cache utility
Manage Cache Tiles utility
Manage Service utility
Manage Site utility

http://resources.arcgis.com/en/help/main/10.1/index.html#//015400000626000000

In 10.1 ESRI has provided a set of Python scripts that tap into the REST Management API..

The Manage Service utility (manageservice.py) provides most of the functionality of the AGSSOM tools.

Manage Service

```
manageservice -s http://myserver:6080 -t -l -u xx -p xx

    AddressDashboard.MapServer         | STOPPED
    GISEmbeddedLayers.MapServer        | STARTED

manageservice -s http://myserver:6080 -t -o stop ^
    -n GISEmbeddedLayers -u xx -p xx

manageservice -s http://myserver:6080 -t -o start ^
    -n GISEmbeddedLayers -u xx -p xx

manageservice -s http://myserver:6080 -t -o stop ^
    -n '*all*' -u xx -p xx
```

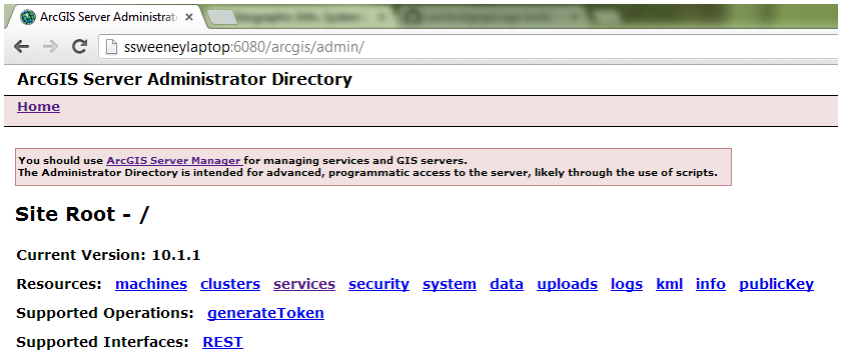
http://resources.arcgis.com/en/help/main/10.1/index.html#/Manage_Service_utility/015400000625000000/

This covers many of our use cases, but it would still require some custom shell scripts to do exactly what we want.

Most notably it doesn't have the "stop all" option.

These are Python scripts (using the Server ArcPy library) and can be mined for code.

Administrator API for 10.1



http://resources.arcgis.com/en/help/server-admin-api/

The new ArcGIS Server Administrator API for 10.1 is a REST API that lets you do your server management through HTTP.

One way to do this is through a browser, but a more powerful method is through a language like Python. You could also use C#, PowerShell, or any other language that supports HTTP.

When accessing the API through a browser you get the default Administrator Directory, which is an HTML interface to the API.

Making the request in code

Python HTTPLIB

```
# Connect to URL and post parameters
httpConn = httplib.HTTPConnection(serverName, serverPort)
httpConn.request("POST", tokenURL, body, headers)

# Read response
response = httpConn.getresponse()
```

The Python library httplib can be used to make http requests and receive responses from the remote server. These responses can then be further processed in the code.

ESRI Example Scripts

ESRI has many example scripts in the Resource Center. These can be used as-is or mined for code/technique.

Relevant Examples:

- Stop or start all services in a folder
- Check a folder for stopped services

Using the API

1. Get token

```
# A function to generate a token given username,
# password and the adminURL.
def getToken(username, password, serverName, serverPort):
    tokenURL = "/arcgis/admin/generateToken"
    ....
    return token['token']
```

```
Token:
1wGGKgBFJuHLDwqABHTfUKYabhIs6mhcnQOuHAjlpLa8HWJkorLfwkFttqOdI5ik
```

This is the equivalent to logging on in the web interface.

ESRI has provided a function in the documentation to do this, which I copied into my library (agsextras.py).

From the documentation:

> The token does not last forever; it is designed to time out so that it cannot be stolen and used indefinitely by a malicious user. You have to request a new token each time you run your script (but not each time you make a request).

Using the API

2. Create the POST message body

```
# The requests used below only need the token and the response
# formatting parameter (json)
body = urllib.urlencode({'token': token, 'f': 'json'})
```

```
token=1wGGKgBFJuHLDwqABHTfUKYabhIs6mhcnQOuHAjlpLa8HWJkorLfwkFttqOdI5ik
&f=json
```

The message body for the POST request is quite simple for these requests. You only need to provide the token and specify the output format. Here we use JSON as the output format (f) so the results can be easily processed by the script. The default is HTML for the Administrator Directory.

Using the API

3. Create headers

```
# The request headers are also fixed
headers = {"Content-type": "application/x-www-form-urlencoded",
"Accept": "text/plain"}
```

The headers for this request are also quite simple. We are treating the request like a URL encoded form request.

Using the API

4. Create URL

```
# Construct URL to get the status, then make the request
reqURL = "/arcgis/admin/services/" + folder + fullSvcName + "/stop"
```

```
http://myserver:6080/arcgis/admin/services/MyMap.MapServer/stop
```

The URL (minus the server and port number) points directly to the operation we want to perform on our resource.

REST APIs consist of resources and operations to perform on resources. In this case the resource is *MyMap.MapServer* and the operation is *Stop*.

Using the API

5. Send request (part 1)

```
def sendRequest(serverName, serverPort, reqURL, body, headers):
    httpConn = httplib.HTTPConnection(serverName, serverPort)
    httpConn.request("POST", reqURL, body, headers)

    # Read response
    response = httpConn.getresponse()
    if (response.status != 200):
        httpConn.close()
        raise RequestException('Invalid response from request.')
```

To send the request, create a new instance of the *HTTPConnection* object for the server and port provided, then use this object to post the request using the parameters gathered previously.

The HTTP response from the server can be checked for errors using the *getresponse()* method.

Using the API

5. Send request (part 2)

```
respData = response.read()

# Done with the connection - close before error checking
httpConn.close()

# Check that data returned is not an error object
if not assertJsonSuccess(respData):
    raise JsonErrorException(str(respData))

# Deserialize response into Python object
data = json.loads(respData)
return data
```

If there are no HTTP errors, close the connection and parse the JSON response for errors. The *assertJsonSuccess()* function is copied from the ESRI samples.

If there are no JSON errors, parse the response data into a Python object for further processing.

My library (agsextras.py)

```
# Function to parse the command line and return the standard arguments
def getArgs(parser):

# A function to generate a token given username, password and the adminURL.
# TODO: Refactor to use sendRequest function
def getToken(username, password, serverName, serverPort):

# Define some custom exception classes for sendRequest
# This will help us distinguish any errors on the calling side
class RequestException(Exception):
class JsonErrorException(Exception):

# Perform a request
def sendRequest(serverName, serverPort, reqURL, body, headers):

# A function that checks that the input JSON object
#  is not an error object.
def assertJsonSuccess(data):

# Serialize a list to disk
def saveList(data,filename):

# Read a serialized list from disk
def readList(filename):
```

I created my own library of functions and classes for use in these scripts called *agsextras.py*. Some of these were lifted from the ESRI sample scripts (*getToken()*, *assertJsonSuccess()*) and the others were created to simplify and consolidate my scripts.

My scripts

agsstopallstarted
agsstopfromlist
agsstatusfromlist
agsstartfromlist

-s server
-u user
-p password
-f filename

So far I have created four scripts:

1. agsstopallstarted.py - stops all services that are currently started and writes an output file with a list of the services that were stopped.
2. agsstopfromlist.py - stops all services listed in a file.
3. agsstatusfromlist.py - gets the status of all the services listed in a file.
4. agsstartfromlist.py - starts all services listed in a file.

There are four common command line arguments for these scripts:

1. Server - Name of the server
2. User - Username for login
3. Password - Password for login
4. Filename - Filename for outputting or inputting the list of services

If the username and password are not provided the user will be prompted. If the filename is not provided a default filename will be used.

Future enhancements

1. Add folder support
2. More consolidation?
3. Refactor all http code to urllib(2)?

1. Right now the tools only support the root Server folder. Support for other folders is left to a future enhancement.

2. Some of the scripts are almost completely the same source-wise and could probably be consolidated for easier maintenance. For example, agsstartfromlist and agsstopfromlist are the same except for one variable name and the actual operation performed (start vs. stop).

3. The consensus on the Web seems to be to use urllib2 in favor of httplib where possible as a best practice. This will require some more investigation but on the surface appears to be doable.

More info

These slides:
http://goo.gl/0AQa9
The scripts:
https://github.com/cambridgegis/ags-tools

Both links tweeted to @hiker4k with #nearc

Setting up the Python environment

A few simple settings can make your life a lot easier when working with Python from the command line.

1. Add python.exe to Windows path

```
SET PATH=%PATH%;C:\Python27\ArcGISx6410.1


C:\Python27\ArcGISx6410.1\python.exe ^
"C:\Program Files\ArcGIS\Server\tools\admin\managesite.py"
^
-u admin -p admin -s http://myserver:6080 -t -lc
```

OR

```
python.exe ^
"C:\Program Files\ArcGIS\Server\tools\admin\managesite.py"
^
-u admin -p admin -s http://myserver:6080 -t -lc
```

Setting up the Python environment

1. Add python.exe to Windows Path

In PowerShell:

```
$env:Path+= ";C:\Python27\ArcGISx6410.1“
```

In cmd.exe:

```
SET PATH=%PATH%;C:\Python27\ArcGISx6410.1
```

Setting up the Python environment

2. Add ArcServer scripts to your local computer

Copy:

```
C:\Program Files\ArcGIS\Server\tools\admin\
```

from your ArcGIS Server server to the same location on your desktop computer.

Setting up the Python environment

3. Run Python scripts directly from the command line

In PowerShell:

```
$env:pathext += ';.PY'
$env:Path += ';C:\Program
Files\ArcGIS\Server\tools\admin'
```

In cmd.exe:

```
SET PATHEXT=%PATHEXT%;.PY
SET PATH=%PATH%;"C:\Program
Files\ArcGIS\Server\tools\admin"
```

Setting up the Python environment

4. Create your own Python library

In PowerShell:

```
$env:PythonPath += ';C:\Python27\Cambridge'
$env:Path += ';C:\Python27\Cambridge'
```

In cmd.exe:

```
SET PYTHONPATH=%PYTHONPATH%;"C:\Python27\Cambridge"
SET PATH=%PATH%;"C:\Python27\Cambridge"
```

Then:

```
import agssextras
```

Parsing arguments in Python

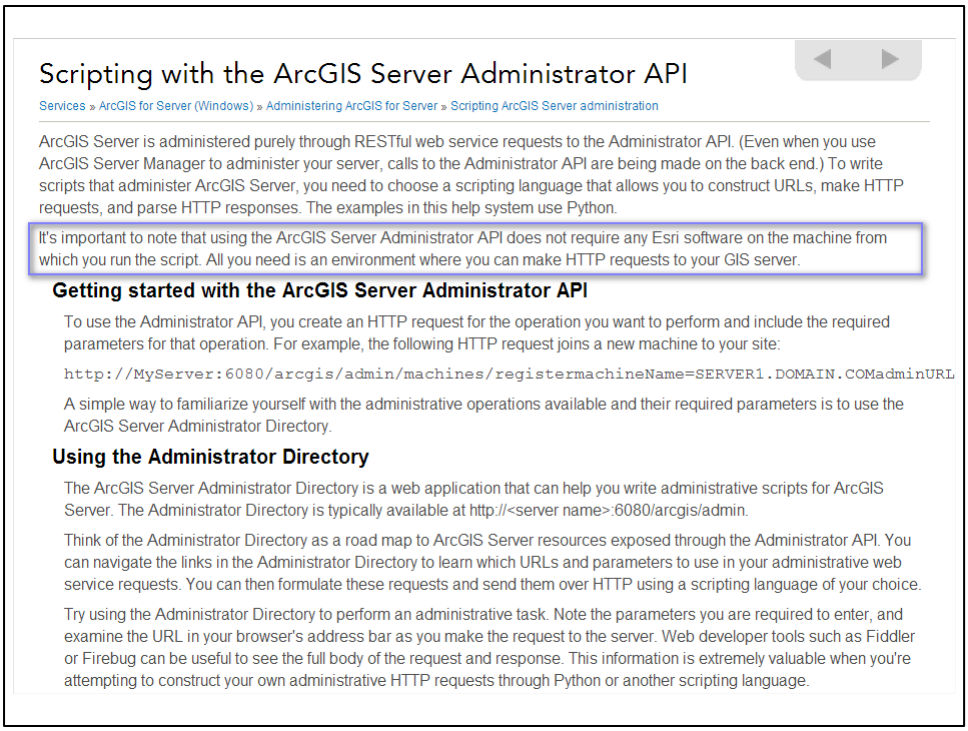
```
# Command line args
import argparse

# Function to parse the command line and return the standard arguments
def getArgs(parser):
    parser.add_argument( '-s', '--server', required=True, help='Server name')
    parser.add_argument( '-u', '--user', required=False, help='User name')
    parser.add_argument( '-p', '--password', required=False, help='Password')
    parser.add_argument( '-f', '--filename', required=False, help='Output file name',
                        default=environ[ 'TEMP'] + '\\agsstarted.txt')

    args = parser.parse_args()

    # Prompt for username if not provided
    if not args.user:
        args.user =  raw_input("Enter user name: ")

    # Prompt for password using getpass if not provided
    if not args.password:
        args.password = getpass.getpass( "Enter password: ")

    return args
```

The Python built-in argparse module can be used to easily parse command line arguments.

Use the add_argument method to add your arguments, set them as required or optional, set help, and set default.

The parse_args() method takes care of the rest. The returned object will have an attribute for each defined argument that was found during the parsing using the long name if it's defined (args.filename for --filename).

There are many more options for creating arguments. This is a fairly simple implementation. See the documentation for details.



Scripting with the ArcGIS Server Administrator API

Services » ArcGIS for Server (Windows) » Administering ArcGIS for Server » Scripting ArcGIS Server administration

ArcGIS Server is administered purely through RESTful web service requests to the Administrator API. (Even when you use ArcGIS Server Manager to administer your server, calls to the Administrator API are being made on the back end.) To write scripts that administer ArcGIS Server, you need to choose a scripting language that allows you to construct URLs, make HTTP requests, and parse HTTP responses. The examples in this help system use Python.

It's important to note that using the ArcGIS Server Administrator API does not require any Esri software on the machine from which you run the script. All you need is an environment where you can make HTTP requests to your GIS server.

Getting started with the ArcGIS Server Administrator API

To use the Administrator API, you create an HTTP request for the operation you want to perform and include the required parameters for that operation. For example, the following HTTP request joins a new machine to your site:

```
http://MyServer:6080/arcgis/admin/machines/registermachineName=SERVER1.DOMAIN.COMadminURL
```

A simple way to familiarize yourself with the administrative operations available and their required parameters is to use the ArcGIS Server Administrator Directory.

Using the Administrator Directory

The ArcGIS Server Administrator Directory is a web application that can help you write administrative scripts for ArcGIS Server. The Administrator Directory is typically available at http://<server name>:6080/arcgis/admin.

Think of the Administrator Directory as a road map to ArcGIS Server resources exposed through the Administrator API. You can navigate the links in the Administrator Directory to learn which URLs and parameters to use in your administrative web service requests. You can then formulate these requests and send them over HTTP using a scripting language of your choice.

Try using the Administrator Directory to perform an administrative task. Note the parameters you are required to enter, and examine the URL in your browser's address bar as you make the request to the server. Web developer tools such as Fiddler or Firebug can be useful to see the full body of the request and response. This information is extremely valuable when you're attempting to construct your own administrative HTTP requests through Python or another scripting language.

> *It's important to note that using the ArcGIS Server Administrator API does not require any Esri software on the machine from which you run the script. All you need is an environment where you can make HTTP requests to your GIS server.*

- http://resources.arcgis.com/en/help/main/10.1/index.html#/Scripting_with_the_ArcGIS_Server_Administrator_API/0154000005r1000000/

This is true, but if you want to take advantage of the work they've already done you need their libraries. Desktop install is good enough as it includes the server arcpy library: C:\Program Files (x86)\ArcGIS\Desktop10.1\arcpy_server_admin.