Bareos Python Plugins Hacking Workshop

Open Source Backup Conference

22 - 23 September 2014 | Cologne

Stephan Dühr & Maik Außendorf

Sep 24, 2014
Agenda

- Bareos architecture and terminology
- Introduction
- Plugin overview (FD, SD, DIR)
- Detailed View at FileDaemon Plugins
- FD Plugin Examples
- Hacking: write your own plugin or extend existing ones
- Resume:
  - short description of work done
  - Feedback about plugin interface, questions, ideas...
Why Plugins?

- Extend Bareos functionality
  - Without touching the Bareos code
  - Can react on numerous events (in contrast to pre- and postscripts)
  - Modify Fileset
  - Extra treatment for files
  - Connect to other systems (Monitoring, Ticket, Hypervisors, Cloud, Logging, Indexer i.e. elasticsearch)
  - Application specific actions on backup and restore
C Plugin Interface

- C Coding knowledge needed
- Compiling / Linking to create a shared library
- Rarely used (only well-known: bpipe)
- No known Bacula / Bareos C-Plugins for SD and Dir
- Incomplete implementation in Bacula / Bareos
- Near to zero documentation
- Legacy Python interface existed, removed from Bareos in favor of new designed interface
New Bareos Python Plugin interface

- Python knowledge widely spread among technical consultants, admins, and devops
- Arbitrary Python modules available to handle a large numbers of applications / APIs
- Plain Python script for FD / SD / DIR plugins
- For FD additional class-based approach available
- Need Python version 2.6 or newer
- Uses distribution provided Python packages
- C code already prepared for Python 3.x
New Bareos Python Plugin interface

- Plugins configured via Bareos configuration
  Pass plugin options to FD plugins
- Bareos core calls functions from the plugins on defined events
- Plugins can influence the backup process and modify Bareos variables
- Plugin usage must be explicitly enabled:
  Plugin Directory = /usr/lib/bareos/plugins
  Plugin Names = python
Director Plugins

- One plugin, fixed name: `bareos-dir.py`
  Hint: use symbolic link from your plugin to `bareos-dir.py`

- Configuration snippet for director resource:
  ```
  Director {
    ...
    Plugin Directory = /usr/lib/bareos/plugins
    Plugin Names = python
  }
  ```
Director Plugins

- Plugin registers for events it wants to be called:
  
  ```
  events = [];
  events.append(bDirEventType['bDirEventJobStart']);
  events.append(bDirEventType['bDirEventJobEnd']);
  events.append(bDirEventType['bDirEventJobInit']);
  events.append(bDirEventType['bDirEventJobRun']);
  RegisterEvents(context, events);
  ```

- Bareos core calls function `handle_plugin_event` with event type and context
Director Plugins

• Context variable can be read:
  
  ```
  jobId = GetValue(context, brDirVariable['bDirVarJobId']);
  ```


```
  bDirVarJob = 1
  bDirVarLevel = 2
  bDirVarType = 3
  bDirVarJobId = 4
  bDirVarClient = 5
  bDirVarNumVols = 6
  bDirVarPool = 7
  bDirVarStorage = 8
  bDirVarWriteStorage = 9
  bDirVarReadStorage = 10
  bDirVarCatalog = 11
  bDirVarMediaType = 12
  bDirVarJobName = 13
  bDirVarJobStatus = 14
  bDirVarPriority = 15,
  bDirVarVolumeName = 16,
  bDirVarCatalogRes = 17
  bDirVarJobErrors = 18
  bDirVarJobFiles = 19
  bDirVarSDJobFiles = 20
  bDirVarSDErrors = 21
  bDirVarFDJobStatus = 22,
  bDirVarSDJobStatus = 23,
  bDirVarPluginDir = 24
  bDirVarLastRate = 25
  bDirVarJobBytes = 26, 
  bDirVarReadBytes = 27
```
**Director Plugins**

- Example: `bareos-dir-nagios.py` (stub)
  - On `bDirEventJobEnd` send job information to Nagios / Icinga
  - Evaluating `bDirVarJobStatus`

- Code excerpt:
  ```python
def handle_plugin_event(context, event):
    ...
    elif event == bDirEventType['bDirEventJobEnd']:
      jobName = GetValue(context, bDirVariable['bDirVarJobName']);
      jobStatus = chr(GetValue(context, bDirVariable['bDirVarJobStatus']))
      ...
      if (jobStatus == 'E' or jobStatus == 'f'):
        nagiosResult = 2; # critical
        nagiosMessage = "CRITICAL: Bareos job %s on %s with id %d failed with %s errors (%s), %d jobBytes" %
        (jobName, jobClient, jobId, jobErrors, jobStatus, jobBytes);
        # send nagiosResult and nagiosMessage to Icinga / Nagios
    ```

- Send to Nagios / Icinga needs to be adapted
SD Plugins

• One plugin, fixed name: `bareos-sd.py`  
  Hint: use symbolic link from your plugin to `bareos-sd.py`

• Configuration snippet to enable plugins in Storage resource of `bareos-sd.conf`:

  ```
  Storage {
    ...
    Plugin Directory = /usr/lib64/bareos/plugins
    Plugin Names = python
  }
  ```
SD Plugins

- when omitting Plugin Names, sd will load all plugins in plugin directory matching glob *-sd.so
- explicitly specifying multiple plugins:
  Plugin Names = python:autoxflate
  will load python-sd.so and autoxflate-sd.so
- registering events and variable handling like dir plugin
- until now, only variables bsdVarJobId, bsdVarJobName and bsdVarPluginDir are exposed
  (python-sd.c still in early development state)
- future: all variables as described should be available, see
example: bareos-sd.py

```python
from bareossd import *
from bareos_sd_consts import *

def load_bareos_plugin(context):
    DebugMessage(context, 100, "load_bareos_plugin called\n")
    events = []
    events.append(bsdEventType['bsdEventJobStart'])
    events.append(bsdEventType['bsdEventJobEnd'])
    RegisterEvents(context, events)
    return bRCs['bRC_OK']

def handle_plugin_event(context, event):
    if event == bsdEventType['bsdEventJobStart']:
        DebugMessage(context, 100, "bsdEventJobStart event triggered\n")
        jobname = GetValue(context, bsdrVariable['bsdVarJobName'])
        DebugMessage(context, 100, "Job " + jobname + " starting\n")
    ...
```
FD Plugins

• how to enable Python Plugins in FD?
• install `bareos-filedaemon-python-plugin`
• in `bareos-fd.conf` add or uncomment:
  ```
  FileDaemon {
      ...
      Plugin Directory = /usr/lib64/bareos/plugins
      Plugin Names = python
  }...
  ```
• like for SD and Dir Plugins, `Plugin Names` can be omitted. Then all Plugins matching glob `*-fd.so` will be loaded
FD Plugins

- multiple plugins possible
- the Plugin parameter in Director's FileSet resource determines which python plugin is used with which parameters.

Syntax:
Plugin = python:module_path=<path-to-python-modules>:module_name=<python-module-to-load>:<custom-param1>=<custom-value1>: ...

- module_path and module_name are mandatory (used by python-fd.so)
- anything else is arbitrary, the complete string is passed to the hook function parse_plugin_definitions()
- two Plugin-Types: Command-Plugins and Option-Plugins
How do FD Plugins work (1)

- When a Job is run, Director passes plugin definition to FD, eg. module_path=/usr/lib64/bareos/plugins:module_name=bareos-fd

FD (python-fd.so) does the following:

- instantiates new Python interpreter
- extends the Python search path with the given module_path
- imports the module given by module_name (for the example above, would be bareos-fd.py)
- makes callback methods available for Python, use from bareosfd import * in Python code
How do FD Plugins work (2)

- **calls** `load_bareos_plugin()` in the python plugin code
- **calls** `parse_plugin_definition(context, plugindef)` in the python code
  - `plugindef` is the complete string as configured in Director (Plugin = ...), to be parsed by python code
- different processing loop depending on type of Plugin (Command/Option)
FD Command-Plugin Configuration

- Command Plugin Configuration in Include section of FileSet Resource in bareos-dir.conf:

```plaintext
FileSet {
    Name = "test_PyLocalFileset_Set"
    Include {
        Plugin = "python:module_path=/usr/lib64/bareos/plugins:module_name=bareos-fd-local-fileset:filename=/tmp/datafile"
    }
}
```
FD Option-Plugin Configuration

- Option Plugin Configuration in Options section of Include Section of FileSet Resource in bareos-dir.conf:

```bash
FileSet {
    Name = "test_PyOptionInteract_Set"
    Include {
        File = /data/project_1
        Options {
            Plugin = "python:module_path=/usr/lib64/bareos/plugins:module_name=bareos-fd-file-interact"
        }
    }
}
```

- Note: for Option-Plugin must define files to backup using `File = ...` while for Command-Plugin need not
Difference FD Command-/Option-Plugins (1)

- **Major Difference:**
  - Command-Plugin determines what is being backed up, must also handle Diff/Inc itself
  - Option-Plugin gets which files to backup based on what's configured in Director, Diff/Inc handling done by FD
Difference FD Command-/Option-Plugins (2)

- Command-Plugin processing
  - `start_backup_file(context, savepkt)` must set savepkt properties for each file to back up
  - `plugin_io(context, IOP)` must handle IO Operations
    - Backup: open(r), read, close
  - `end_backup_file(context)`
    - must return `bRCs['bRC_More']` if more files to backup
    - must return `bRCs['bRC_OK']` to finish the looping
  - `handle_backup_file()` is not called
Difference FD Command-/Option-Plugins (3)

• Option-Plugin processing
  - `handle_backup_file(context, savepkt)` called for each file to be processed, `savepkt` defined by FD
  - `plugin_io()` handling in the same manner as for Command-Plugin
  - `start_backup_file()` and `end_backup_file()` are not called
FD Plugins – Callback Functions

- Functions provided by python-fd.so that can be called from Python code, enabled by
  from bareosfd import *

- Complete list: see http://regress.bareos.org/doxygen/html/d5/d0e/python-fd_8h_source.html

- Most important callback functions:
  - JobMessage(): Error-/Info-/Warning-Messages
    • are passed to Director, appear in messages and logs
  - DebugMessage(): Debug-Messages with numeric level
    • only visible when running FD in debug-mode with -d <level>
  - GetValue(): used to get variables from FD
FD Plugins – Class Based Approach

- Python FD Plugin can be monolithic
- Better: use classes and inheritance to reuse existing code easier and reduce code redundancy
- To support this approach, the package `bareos-filedaemon-python-plugin` package provides
  - `BareosFdPluginBaseclass.py`
    - Parent Class to inherit from
  - `BareosFdWrapper.py`
    - defines all functions a plugin needs and “wraps” them to the corresponding methods in the plugin class
  - a Plugin entry-point module glues them together
Messaging

- **DebugMessage**: Debug only
  - `DebugMessage(context, level, "message\n");`
    - **context**: used to pass information from core to plugin, don't touch
    - **Level**: Debug Level, use `>= 100`
  - **Sample**:
    ```
    DebugMessage(context, 100, "handle_backup_file called with " + str(savepkt) + "\n");
    ```
Messing

- **JobMessage**: Sent to messaging system
  - `JobMessage(context, bJobMessageType, "Message\n");`

- **Type**: Controls job result, `M_INFO`, `M_ERROR`, `M_WARNING`, `M_ABORT`

  - [Sample](http://regress.bareos.org/doxygen/html/dd/dbb/namespacebareos__fd__consts.html)

  - `JobMessage(context, bJobMessageType['M_INFO'], "Option Plugin file interact on" + savepkt.fname + "\n");`
Return Codes

• Return Codes control processing, no impact on overall job status.

• Depending on context / function

• Use consts:

```c
return bRCs['bRC_OK'];
return bRCs['bRC_Skip']; # skips current file
return bRCs['bRC_Error']; # error but continue
return bRCs['bRC_More']; # in end_backup_file, more files to backup
...
```
FD Plugin: bareos-fd-local-fileset.py

- Reads a local file on fd with filenames to backup
  - Demonstration / template plugin, functionality can be achieved better by fileset configuration:
    
    ```
    File = "\\</localfile/on/client"
    ```

- Configuration in fileset resource as command plugin (extends fileset):
  ```
  Plugin = "python:module_path=/usr/lib64/bareos/plugins:module_name=bareos-fd-local-fileset:filename=/tmp/datafile"
  ```

- Plugin: /usr/lib64/bareos/plugins/bareos-fd-local-fileset.py
  Code excerpt:
  ```
  from bareosfd import *
  from bareos_fd_consts import *
  from BareosFdWrapper import *
  from BareosFdPluginLocalFileset import *
  def load_bareos_plugin(context, plugindef):
      BareosFdWrapper.bareos_fd_plugin_object = BareosFdPluginLocalFileset (context, plugindef);
      return bRCs['bRC_OK'];
  ```

- Rest is done in class BareosFdPluginLocalFileset
BareosFdPluginLocalFileset

- Class inherits from BareosFdPluginBaseclass
- Method `parse_plugin_definition`:
  Parses the options, filename is mandatory
  Reads filenames from file into array
  ```python
  self.files_to_backup
  ```
- Method `start_backup_file` asks plugin, if there is anything to backup, sets `savepkt`:
  ```python
  file_to_backup = self.files_to_backup.pop();
  savepkt.fname = file_to_backup;
  savepkt.type = bFileType['FT_REG'];
  return bRCs['bRC_OK'];
  ```
- **Method** `end_backup_file` called to ask plugin if there is more to backup:

  ```python
  if self.files_to_backup:
    # there is more to backup, go to start_backup_file again
    return bRCs['bRC_More'];
  else
    # no more to backup from this plugin, done
    return bRCs['bRC_OK'];
  ```

- **Basic IO operations covered in base class**
  - Method `plugin_io` handles file read / write operations
For restore: some more things to do

- Directories have to be created
  
  ```python
  def create_file(self, context, restorepkt):
      FNAME = restorepkt.ofname;
      dirname = os.path.dirname (FNAME);
      if not os.path.exists(dirname):
          os.makedirs(dirname);
      open (FNAME,'wb').close();
      restorepkt.create_status = bCFs['CF_EXTRACT'];
      return bRCs['bRC_OK'];
  ```

- Similar in method plugin_io for writing

- Overload this method in your class, if you need different handling
FD Plugin: bareos-fd-file-interact.py

- Stub / example for option plugin
- Gets called before a file is read for backup, giving the possibility to do anything with this file
- Configuration in fileset resource as option plugin in options section:
  ```
  Plugin = "python:module_path=/usr/lib64/bareos/plugins:module_name=bareos-fd-file-interact"
  ```
- Plugin: /usr/lib64/bareos/plugins/bareos-fd-local-fileset.py
  Uses BareosFdWrapper with class:
  BareosFdPluginFileInteract
BareosFdPluginFileInteract

- Method `handle_backup_file` called for each file to backup with param `savepkt`
  - Backup file with full path in `savepkt.fname`
  - File type in `savepkt.type`

- File types (selection):
  - `FT_REG` = 3 # Regular file
  - `FT_LNK` = 4 # Soft link
  - `FT_FIFO` = 17 # Raw FiFo device

- `FT_RESTORE_FIRST` = 25
  - Special Restore object, can contain metadata needed for restore. Needs to be created as virtual file, these objects will be restored first, handled by your plugin.
My SQL Plugin

- FD Plugin for MySQL Backup contributed by Evan Felix (https://github.com/karcaw)
- Available at https://github.com/bareos/bareos-contrib/tree/master/fd-plugins/mysql-python
- runs `mysql -B -N -e 'show databases'` to get the list of databases to back up
- runs `mysqldump %s --events --single-transaction | gzip` for each database, using `os.popen()` (pipe)
- `plugin_io()` reads the pipe, no temporary local diskspace needed for the dump
My SQL Plugin

• Possible enhancements:
  – add option to specify which DB(s) to backup
  – add option to specify compression program
    • pbzip2 or pigz may accelerate the backup, if system has multiple CPUs/Cores
    • add the ability to disable compression by pipe, to have Bareos handle compression
VMware plugin

- Work in progress to allow Snapshot based VM Backups, not yet published
- Use VMware's CBT (Changed Block Tracking) to allow space efficient Full and Incremental Backups
- Coping with the complex VMware API is not easy
- Using Java was a no-go for Bareos
- Until December 2013 several more or less useful Projects to use the API with Python were around: PySphere, Psphere, PyVISDK
- in December 2013 pyvmomi appeared on github, a Python SDK for the Vsphere API, sponsored/supported by VMware
VMware plugin

- Also requires using the Virtual Disk Development Kit (VDDK), it's a collection of C Libraries, sometimes also named vmware-vix-disklib (propriety)
- No good or VMware sponsored/supported VDDK binding for Python exists
- Using https://github.com/xuru/vixDiskLib in a Python FD Plugin failed because VDDK comes with some older libs that caused unresolvable conflicts/errors
- New approach uses a separate program developed in C that handles VDDK and pipes data to the FD Plugin
VMware plugin

• What works already:
  – Creating/Removing a snapshot
  – CBT base dump (Full), CBT advantage: only active portions of virtual disks dumped
  – manual restore

• Still Missing:
  – Saving VM Metadata/Config, needed to re-create the same VM or a identically configured VM for restore
  – Creating a VM for restore and the restore process itself
    • Citation from http://pubs.vmware.com/vsphere-55/index.jsp#com.vmware.vddk.pg.doc/vddkBkupVadp.9.4.html?path=7_4_0_6_2_0#1020552
      “This section shows how to create a VirtualMachine object, which is complicated but necessary so you can restore data into it.”
  – Differential/Incremental Backup/Restore
Live Hacking

• Group together (2/3 people per group)
• Get one of the existing plugins up and running
• Extend existing plugin, e.g.
  – Icinga / Director plugin: configurable interface to Nagios / Icinga (send_nsca...)
  – Mysql: make databases to backup configurable, gzip optional, restore directly to db optional
  – Local Fileset plugin: directories, optionally include / exclude files belonging to a specific user
Live Hacking

- More ideas:
  - Director plugin: connect to ticket system (otrs, rt)
  - FD option plugin: pass files to elasticsearch for indexing plus backup meta information
  - FD option plugin: create local log for every file in backup with timestamp and checksum
  - FD option plugin: gpg encrypt every file on the fly
Live Hacking

• More ideas – application specific plugins
  – IMAP / Cyrus: restore to specific mailbox directory
  – Open Xchange (backup / restore of single objects)
  – Kolab
  – other SQL or NoSQL Databases
  – oVirt/RHEV Backup (REST API now has backup functions)
  – Snapshot based KVM (some ideas next slide)
  – Other applications?
Live Hacking

**Ideas regarding KVM Backup**

- KVM/qemu has nothing like VMware CBT
- Promising design proposals like [http://wiki.qemu.org/Features/Livebackup](http://wiki.qemu.org/Features/Livebackup) have never been completed
- Guest-Agent quiescing actions should be looked at
- Performance impact of overlay chaining?
- Use Python libvirt bindings
Contact and links

- Subscription, Support, References, Partner: http://www.bareos.com
- Community, Documentation, Download: http://www.bareos.org
- GIT Bareos: https://github.com/bareos
- GIT Bareos contrib for plugins: https://github.com/bareos/bareos-contrib
- Bug- and feature- tracker Mantis: https://bugs.bareos.org