Bareos Python Plugins Hacking Workshop

Open Source Backup Conference

29 - 30 September 2015 | Cologne

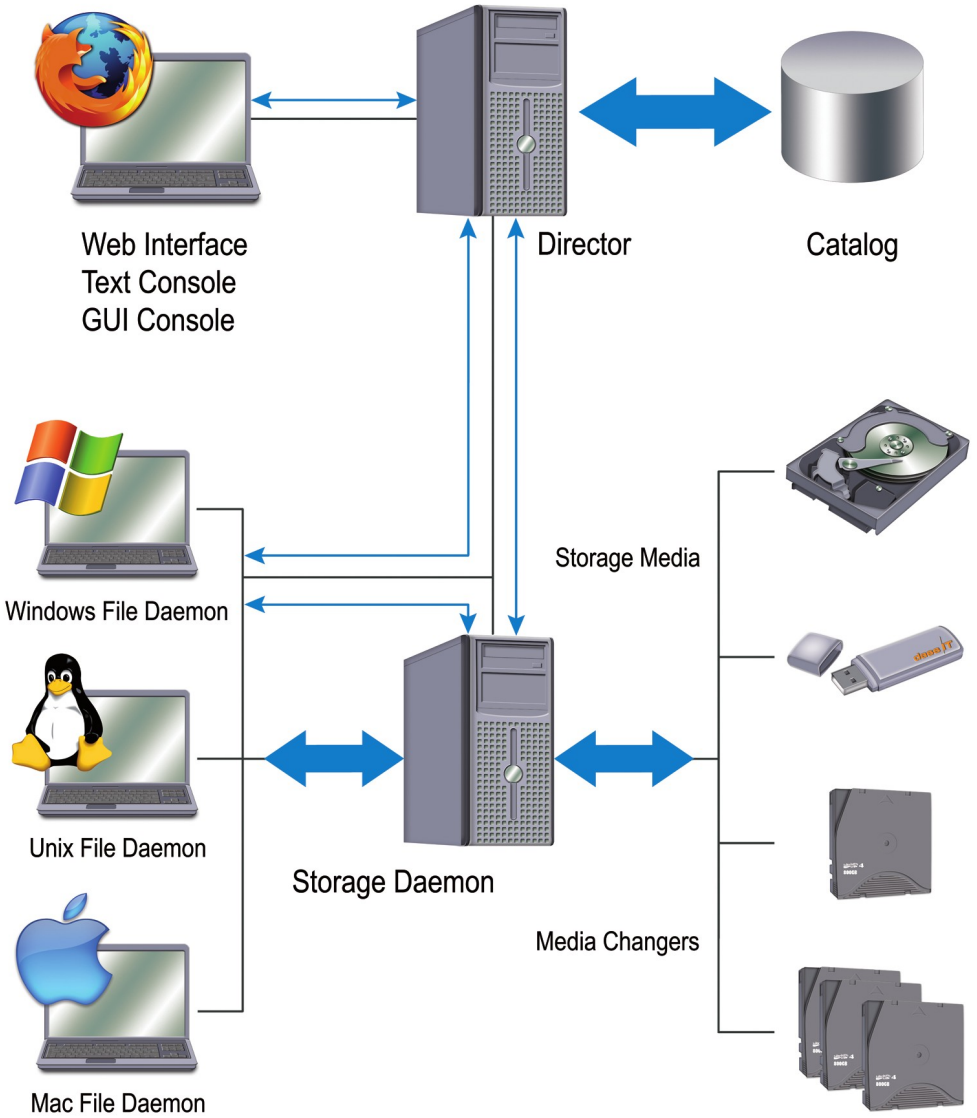
Stephan Dühr & Maik Außendorf

Oct 9, 2015

Agenda

- Bareos architecture and terminology
- Introduction
- Plugin overview (FD, SD, DIR)
- Director Plugin Example: Icinga/Nagios plugin (NSCA-sender)
- Detailed View at FileDaemon Plugins
- FD Plugin Examples
- Director API and usage samples with Python
- Hacking: write your own plugin or extend existing ones
- Resume:
  - short description of work done
  - Feedback about plugin interface, questions, ideas...

Architecture Overview



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

New Bareos Python Plugin interface

- Python knowledge wide spread among technical consultants, admins and devops
- Arbitrary Python modules available to handle a large numbers of application / APIs
- Plain Python script for FD / SD / DIR plugins
- For FD additional class based approach, with 15.2 also for SD and DIR
- Need Python version 2.6 or newer
- Uses distribution provided Python packages
- C code already prepared for Python 3.x

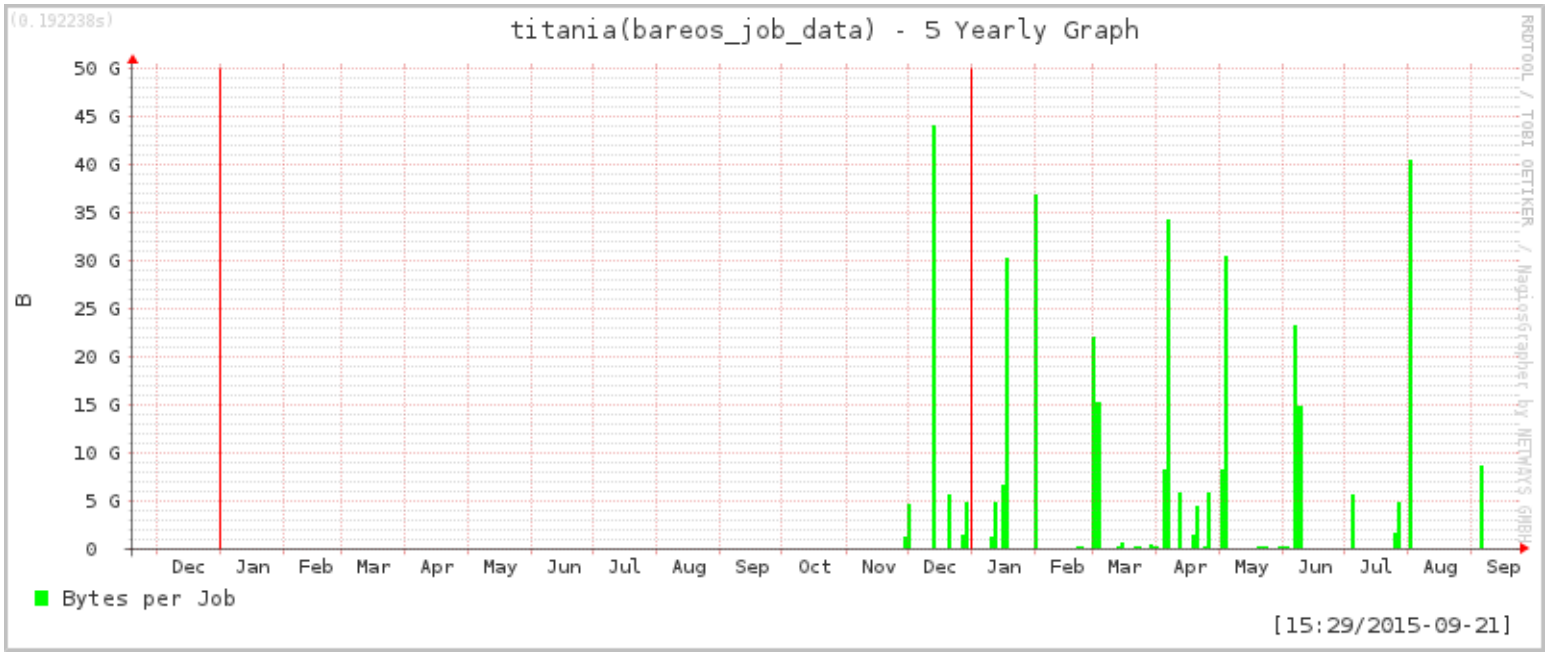
Bareos Python Plugin interface

- 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: NSCA-sender

- Icinga / Nagios NSCA plugin
  - Submits job results and performance data by NSCA right after a job has finished.
  - OK: Bareos job titania-data.2015-09-20_20.05.01_47 on titania-fd with id 19374 level D, 0 errors, 75433922 jobBytes, 24 files terminated with status T



Director Plugins: NSCA-sender

- Icinga / Nagios NSCA plugin configuration as Job directive:

```
Director {
   Plugin Directory = /usr/lib64/bareos/plugins
   Plugin Names = "python”
   …
}

Job {
   ...
   DIR Plugin Options="python:module_path=/usr/lib64/bareos/plugins:
module_name=bareos-dir-nsca-sender:monitorHost=icingahost:
checkHost=my_bareosFD:checkService=bareos_backup"
…
}
```

- https://github.com/bareos/bareos-contrib/tree/master/dir-plugins/nagios_icinga

Director Plugins

- Base Class available, that provides basic and derived job information:
  - self.jobName = bareosdir.GetValue(context, brDirVariable['bDirVarJobName'])
  - self.jobLevel = chr(bareosdir.GetValue(context, brDirVariable['bDirVarLevel']))
  - self.jobType = bareosdir.GetValue(context, brDirVariable['bDirVarType'])
  - self.jobId = int(bareosdir.GetValue(context, brDirVariable['bDirVarJobId']))
  - self.jobClient = bareosdir.GetValue(context, brDirVariable['bDirVarClient'])
  - self.jobStatus = bareosdir.GetValue(context, brDirVariable['bDirVarJobStatus'])
  - self.Priority = bareosdir.GetValue(context, brDirVariable['bDirVarPriority'])
  - self.jobPool = bareosdir.GetValue(context, brDirVariable['bDirVarPool'])
  - self.jobStorage = bareosdir.GetValue(context, brDirVariable['bDirVarStorage'])
  - self.jobMediaType = bareosdir.GetValue(context, brDirVariable['bDirVarMediaType'])
- Derived information
  - self.jobTotalTime = self.jobEndTime - self.jobInitTime
  - self.jobRunningTime = self.jobEndTime - self.jobRunTime
  - self.throughput = self.jobBytes / self.jobRunningTime

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 paramters.
  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_definition()
- 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 `import bareosfd` in Python code

How do FD Plugins work (2)

- makes constants used as callback method parameters available for Python, use eg.
  `from bareos_fd_consts import bJobMessageType, bFileType, bRCs`
  in Python code. All defined constants see:
  http://regress.bareos.org/doxygen/html/dd/dbb/namespacebareos__fd__consts.html
- 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:

```
FileSet {
  Name = "test_PyLocalFileset_Set"
  Include {
     Plugin =
"python:module_path=/usr/lib64/bareos/plugins:module_n
ame=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:

```
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 whats 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
  `import bareosfd`
- Complete list: see http://regress.bareos.org/doxygen/html/d5/d0e/python-fd_8h_source.html
- Most important callback functions:
  - `bareosfd.JobMessage()`: Error-/Info-/Warning-Messages
    - are passed to Director, appear in messages and logs
  - `bareosfd.DebugMessage()`: Debug-Messages with numeric level
    - only visible when running FD in debug-mode with -d <level>
  - `bareosfd.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

- bareosfd.DebugMessage: Debug only
  - `bareosfd.DebugMessage(context, level, "message\n");`
    - context: used to pass information from core to plugin, don't touch
    - Level: Debug Level, use >= 100
  - Sample:

```
bareosfd.DebugMessage(context, 100, "handle_backup_file called with " +
str(savepkt) + "\n");
```

Messaging

- bareosfd.JobMessage: Sent to messaging system
  - `bareosfd.JobMessage(context, bJobMessageType, “Message\n");`
    - Type: Controls job result, `M_INFO, M_ERROR, M_WARNING, M_ABORT`
      http://regress.bareos.org/doxygen/html/dd/dbb/namespacebareos__fd__consts.html
  - Sample:

```
bareosfd.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:

```
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:

```
import bareos_fd_consts
import BareosFdWrapper
from BareosFdWrapper import *
import BareosFdPluginLocalFileset
def load_bareos_plugin(context, plugindef):
    BareosFdWrapper.bareos_fd_plugin_object = \
        BareosFdPluginLocalFileset.BareosFdPluginLocalFileset(
            context, plugindef)
    return bareos_fd_consts.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
  `self.files_to_backup`
- Method `start_backup_file` asks plugin, if there is anything to backup, sets `savepkt`:

```
file_to_backup = self.files_to_backup.pop();
savepkt.fname = file_to_backup;
savepkt.type = bFileType['FT_REG'];
return bRCs['bRC_OK'];
```

BareosFdPluginLocalFileset

- Method `end_backup_file` called to ask plugin if there is more to backup:

```
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

BareosFdPluginLocalFileset

- For restore: some more things to do
  - Directories have to be created

```
def create_file (self,context, restorepkt):
    FNAME = restorepkt.ofname;
    dirname = os.path.dirname (FNAME);
    if not os.path.exists(dirname):
        os.makedirs(dirname);
    if restorepkt.type == bFileType['FT_REG']:
        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

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 or use databases specified by option `db`
- runs `mysqldump %s --events --single-transaction` for each database, using `os.popen()` (pipe)
- `plugin_io()` reads the pipe, no temporary local diskspace needed for the dump
- Restore to dumpfile

My SQL Plugin

- Configuration in Fileset-Include resource:
  `Plugin= "python:module_path=/usr/lib64/bareos/plugins:module_name=bareos-fd-mysql:db=test,mysql"`
- More options with default settings:
  - mysqlhost = localhost
  - Dumpoptions = --events --single-transaction
  - drop_and_recreate = true
    Adds --add-drop-database –databases to mysqldump options
  - mysqluser = <bareos-fd user (root)>
  - mysqlpassword =
  - dumpbinary = mysqldump
- Possible enhancements:
  - add restore-option to directly pipe data into mysql instead of creating a dump file

VMware plugin

- FD Plugin to allow Snapshot based VM Backups
- 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 (properity)
- 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
- More Details: Presentation tomorrow at 9:15 am

Bareos Director API

- Python-bareos
  - Source: https://github.com/bareos/python-bareos
  - From http://download.bareos.org/bareos/contrib/
- Use Bconsole commands from Python

Calling Director Commands

- `import bareos.bsock`
- `password=bareos.bsock.Password('bareos')`
- `bsock=bareos.bsock.BSock(address='localhost', name='admin', password=password)`
- `print bsock.call("list clients")`

Python “bconsole”

- `import bareos.bsock`
- `password=bareos.bsock.Password('bareos')`
- `bsock=bareos.bsock.BSock(address='localhost', name='admin', password=password)`
- `bsock.interactive()`

API JSON

- On bsonsole, run

```
.api json
```

- Commands will return JSON output.
- Output is oriented on JSON-RPC
- See Bareos Developer Guide:
  http://doc.bareos.org/master/html/bareos-developer-guide.html#api-mode-2-json

Director Commands: JSON

- `import bareos.bsock`
- `password=bareos.bsock.Password('bareos')`
- `bsock=bareos.bsock.BSockJson(address='localhost', name='admin', password=password)`
- `bsock.call('list pools')`

Examples

- `bconsole-json.py --name admin -p bareos localhost`
- `mkdir /tmp/bareosfs`
- `bareos-fuse.py -o address=localhost,name=bareosfs,password=bareosfs /tmp/bareosfs`
- as root:
  - `mount -t bareosfs -o address=localhost,name=bareosfs,password=bareosfs fuse /mnt`

Extend bareosfs

- Create a directory, that shows all jobs that have failed in the last 24 hours:
- `/usr/lib/python2.7/site-packages/bareos/fuse/node/jobs.py`
- In `do_update(self)` add the line

```
self.add_subnode(JobsList,
"mydirectory", "jobstatus=E days=1")
```

- remount

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:
  - FD command plugin: backup user accounts (eg. using getent passwd, getent shadow), allow restore of single user account
  - 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
  - Director plugin: connect to ticket system (otrs, rt)
  - Director API Tasks
  - Your own idea

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 have never been completed
  - a CBT-like approach using external QCOW2 snapshots/overlays could be derived from https://kashyapc.fedorapeople.org/virt/lc-2012/snapshots-handout.html
  - Guest-Agent quiescing actions should be looked at
  - Performance impact of overlay chaining?
  - use python libvirt bindings

Live Hacking: MySQL Plugin

- Get existing plugin up and running
  - Test backup and restore of a database
- Add an option to restore directly to database
  - Add option `directRestore`, if set to “yes” as restore Plugin option, restore should go into Database instead of plain dump file

```
Plugin Options:
python:module_path=/usr/lib64/bareos/plugins:mo
dule_name=bareos-fd-mysql:directRestore=yes
```

Live Hacking: MySQL Plugin

- Guide
  - Add method create_file, which is called during restore
    - Direct restore set to NO:
      - Call method from super class
    - Otherwise:
      - Get database name from restore filename (restorepkt.ofname)

Live Hacking: MySQL Plugin

- Guide
  - Modify method plugin_io
    - Direct restore set to NO:
      - Call method from super class
    - Otherwise:
      - Case bIOPS['IO_OPEN']:
        - Open stream to mysql with a subprocess
      - Case bIOPS['IO_WRITE']:
        - Write IOP.buf to stream / mysql command

Live Hacking: MySQL Plugin

- Guide
  - Consider exception and error handling
  - Cleanup:
    - Implement method end_restore_file: close stream / sub-process and catch messages, see end_backup_file for reference.
  - Test / Document
  - Publish
    - Make a fork of https://github.com/bareos/bareos-contrib and propose a patch

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