Mentat System

Introduction for Developers

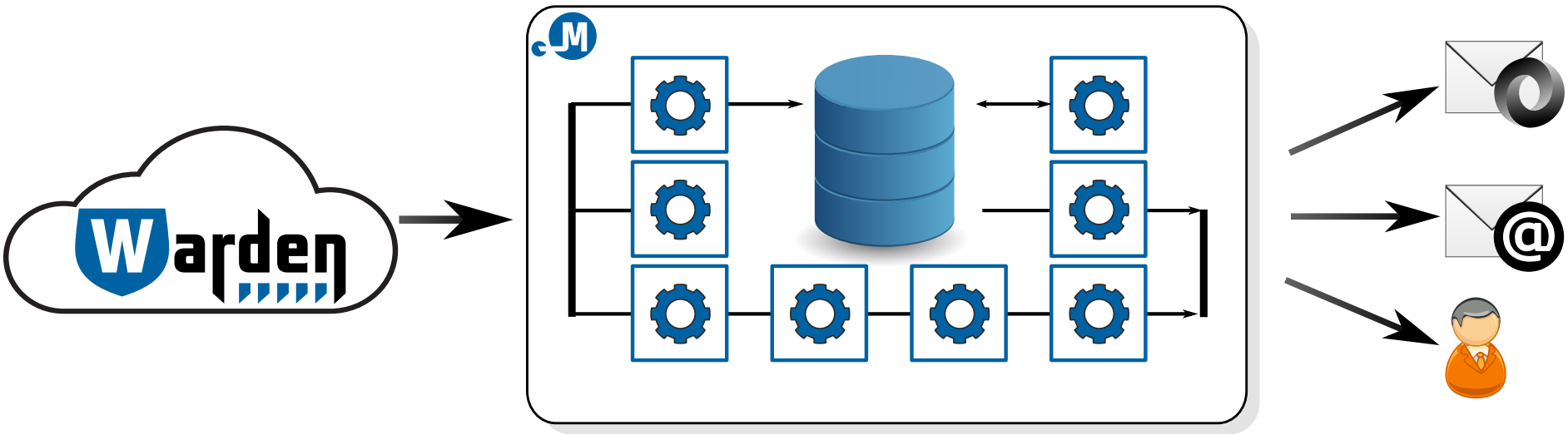
Jan Mach

Intro 2017-12-04

Agenda

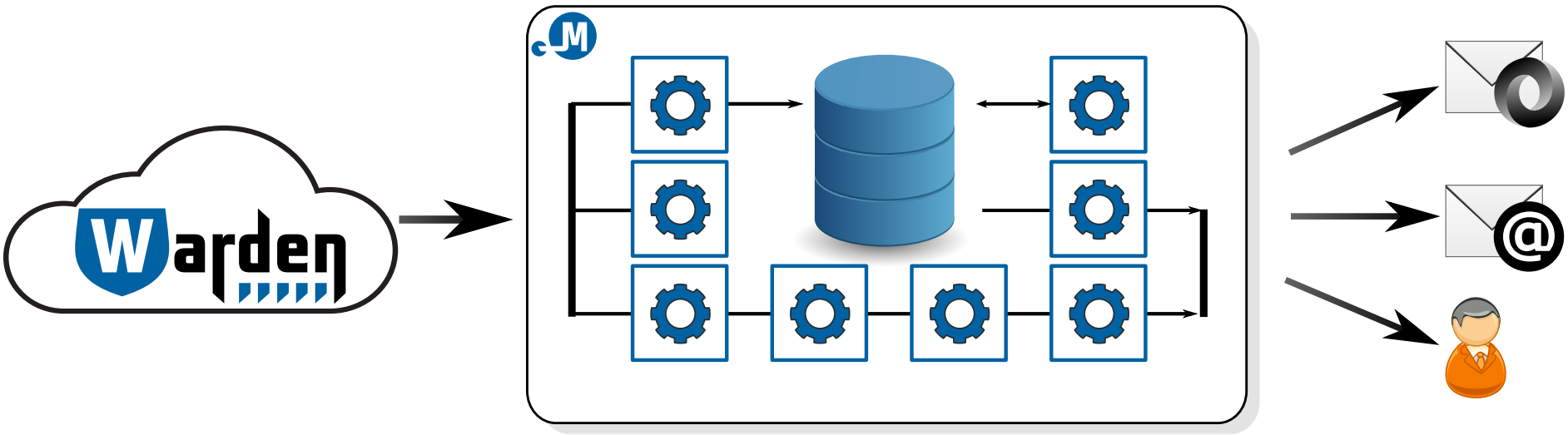
Agenda

System overview



- Implementation language: Python3
- Data model: IDEA
- Data storage: MongoDB
- Network communication protocol: Warden

General system features



- Design inspired by Postfix MTA
  - Hierarchical structure of many small one task daemons and support scripts
  - Filesystem directory message queues (aka. filer protocol)
  - Easy paralelization
- Common internal module design and framework for module development

Current status

- The project is still evolving
- Migration to Python3 (80% done)
- Migration to PostgreSQL (50% done)
- Webpage: `https://mentat.cesnet.cz/`
- Documentation: `https://alchemist.cesnet.cz/`
- Git repository: `https://alchemist.cesnet.cz/`

Alchemist build system

https://alchemist.cesnet.cz/

- Automated build system for Mentat and related libraries
- Contents:
  - General information
  - Build environment settings
  - Autogenerated documentation
  - Git repositories
  - Debian packages
  - Python wheels

Agenda

Agenda

Warden

`https://warden.cesnet.cz/en/index`

- A system for efficient sharing information about detected events (threats)
- Simple client-server architecture
- Sending and receiving clients
- Based on HTTPS protocol with bidirectional certificate authentication
- Communication possible with any HTTPS capable library
- Python client library and simple filer daemon in distribution
- Community approach in data sharing

Data model: IDEA

`https://idea.cesnet.cz/en/index`

- Intrusion Detection Extensible Alert
- JSON based format (NoSQL friendly)
- Shallow structure, strong typed (SQL friendly)
- Easily extendable and customizable
- Possibility to mark anonymised, inaccurate, incomplete or forged data
- Support for aggregated, correlated events
- Support for various data attachments
- Dictionaries for description of various event attributes (Category, Source/Target type, etc.)

IDEA: Example message

- Example Botnet C&C report event

```
{
    "Format": "IDEA0",
    "ID": "cca3325c-a989-4f8c-998f-5b0e971f6ef0",
    "DetectTime": "2014-03-05T15:52:22Z",
    "Category": ["Intrusion.Botnet"],
    "Description": "Botnet Command and Control",
    "Source": [
        {
            "Type": ["Botnet", "CC"],
            "IP4": ["93.184.216.119"],
            "Proto": ["tcp", "ircu"],
            "Port": [6667]
        }
    ]
}
```

Database storage: MongoDB

`https://www.mongodb.com/`

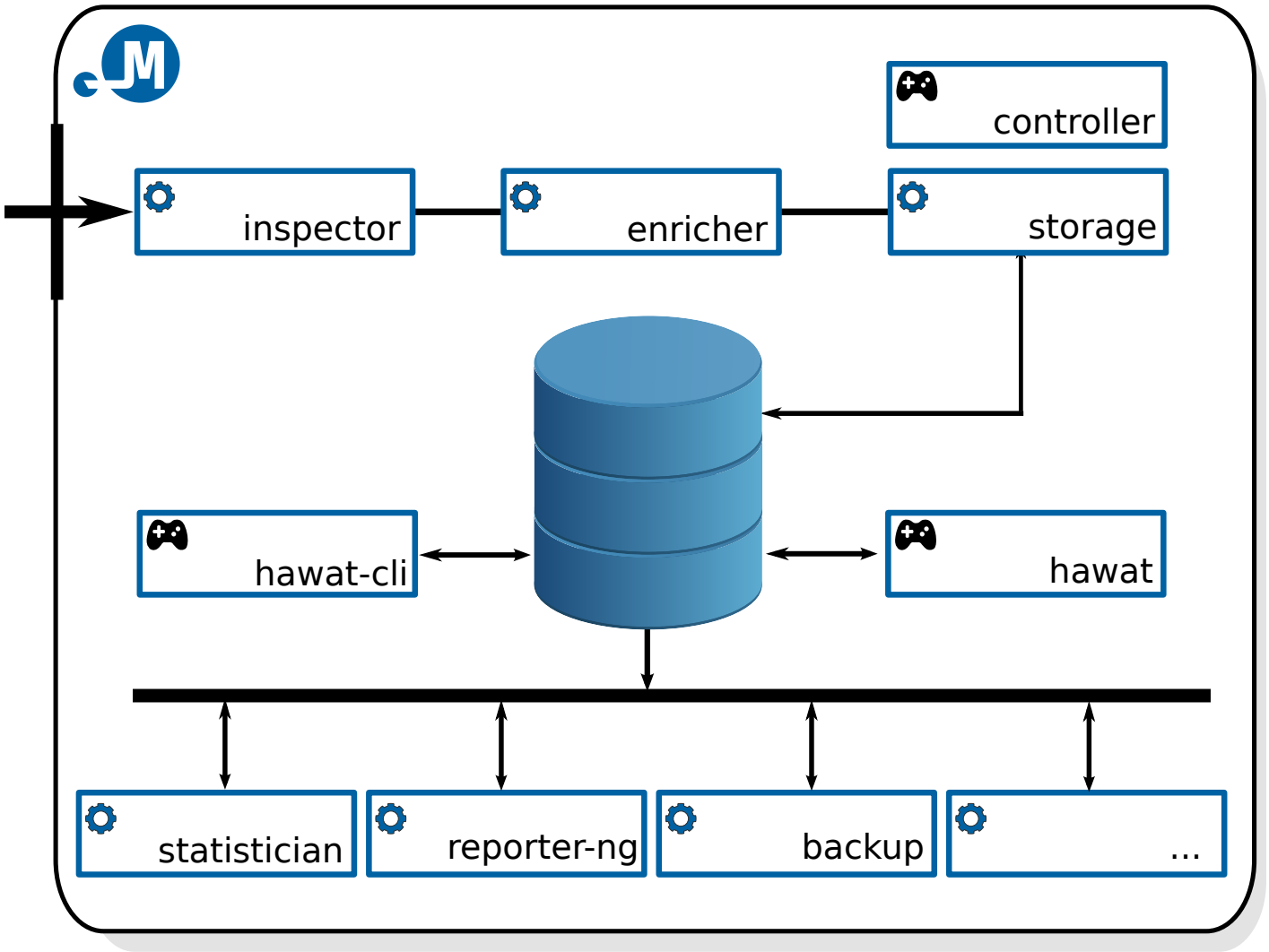
- Single database instance
- Statistics
  - ~96,000,000 events in total
  - Receiving about 3,000,000 new events daily
  - Database size is ~100 GB
  - Average object size in DB is 1.7 KB
  - Record TTL is 6 months for interesting events, 4 weeks otherwise
  - Keeping infinite history of statistical summaries
  - Optimized indices for most common searches

Libraries

- idea-format: Library for working with IDEA messages
- pynspect: Data filtering library
- pyzenkit: Application development framework

Agenda

System architecture



System modules

- Real-time event processing modules
  - mentat-inspector
  - mentat-enricher
  - mentat-storage
- Event post processing modules (via database)
  - mentat-statistician
  - (management scripts)
- Control modules and user interfaces
  - mentat-controller

Module design

- Design inspired by Postfix MTA
  - Hierarchical structure of many small one task daemons
  - Filesystem directory message queues
- Process paralelization support, more instances can work with the same queue
- PyZenKit as common framework for module development

Message exchange queue (1)

- aka. filer protocol
- simple filesystem directory with substructure:
  - incoming: input queue, only complete messages
  - pending: daemon work directory, messages in progress
  - tmp: work directory
  - errors: messages causing problems during processing
- key requirement: atomic move

Message exchange queue (2)

- Inserting message into queue:
  - create new file in tmp subdirectory
  - filename is arbitrary, but must be unique within all subdirectories
  - when done writing, move/rename the file to incoming
  - move must be atomic, so all subdirectories must be on same partition

Agenda

Design goals

- provide feature rich application out of the box
- enable customizability and extendability
  - built-in features are configurable by text files, and/or command line arguments
  - callback hooks for subclasses
  - prepared for inheritance and method overloading

pyzenkit.jsonconf

- reading and writing of JSON configuration files
- merging multiple JSON configuration files
- support for configuration directories
- support for single line comments in JSON files

pyzenkit.daemonizer

- setup directories and limits
- setup user and group permissions
- double fork and split session
- setup signal handlers
- close all open file descriptors (except for possible log files)
- redirect stdin, stdout, stderr to /dev/null
- detect current PID and store it to appropriate PID file
- at exit remove PID file

pyzenkit.baseapp (1)

- base implementation for generic console application
- Features:
  - application configuration service
  - command line argument parsing service
  - logging service
  - persistent state service
  - application runlog service
  - plugin system (experimental)
  - application actions

pyzenkit.baseapp (2)

- Application usage modes:
  - run
  - plugin
- Application life cycle:
  - init
  - setup
  - process
  - evaluate
  - teardown

pyzenkit.baseapp (3)

- Built-in actions:
  - config-view
  - runlog-dump
  - runlog-view
  - runlogs-dump
  - runlogs-list
  - runlogs-evaluate

pyzenkit.baseapp (4)

- example implementation can be found in module source code
- documentation: https://alchemist.cesnet.cz/

```
# On Debian Jessie try following (as root):
cd /usr/local/lib/python3.4/dist-packages
python3 pyzenkit/baseapp.py --help
python3 pyzenkit/baseapp.py
python3 pyzenkit/baseapp.py --action runlogs-evaluate
```

pyzenkit.zenscript (1)

- base implementation for generic console script application
- based on pyzenkit.baseapp
- Additional features:
  - support for executing multiple different commands
  - execution modes: default, regular, shell
  - support for executions in regular time intervals

pyzenkit.zenscript (2)

- example implementation can be found in module source code
- documentation: `https://alchemist.cesnet.cz/`

```
# On Debian Jessie try following (as root):
cd /usr/local/lib/python3.4/dist-packages
python3 pyzenkit/zenscript.py --help
python3 pyzenkit/zenscript.py
python3 pyzenkit/zenscript.py --command alternative
python3 pyzenkit/zenscript.py --action runlogs-evaluate
```

pyzenkit.zendaemon (1)

- base implementation for generic daemon application
- based on pyzenkit.baseapp
- Additional features:
  - fully automated daemonization process
  - event driven design
  - support for handling arbitrary signals
  - support for modularity with daemon components

pyzenkit.zendaemon (2)

- Event driven design:
  - infinite event loop and event scheduler
  - events are being emited in different parts of application
  - event callbacks must be registered to handle events
  - multiple event callback may handle single event (pipeline)
- Event scheduling:
  - `schedule`
  - `schedule_next`
  - `schedule_after`
  - `schedule_at`

pyzenkit.zendaemon (3)

- Signal handling:
  - SIGINT
  - SIGUSR1
  - SIGUSR2
- Sending signals:

```
# On Debian Jessie try following (as root):
cd /usr/local/lib/python3.4/dist-packages
python3 pyzenkit/zendaemon.py --no-daemon
python3 pyzenkit/zendaemon.py --action signal-usr1
python3 pyzenkit/zendaemon.py --action=signal-usr2
```

pyzenkit.zendaemon (4)

- Daemon components:
  - actual workers in the design
  - the daemon object is in fact only a container for components
  - components must be registered into the daemon object
  - great for code reusability

pyzenkit.zendaemon (5)

- example implementations can be found in module source code
- documentation: https://alchemist.cesnet.cz/

```
# On Debian Jessie try following (as root):
cd /usr/local/lib/python3.4/dist-packages
python3 pyzenkit/zendaemon.py --help
python3 pyzenkit/zendaemon.py --no-daemon
python3 pyzenkit/zendaemon.py --action runlogs-evaluate
```

Agenda

mentat.daemon.piper (1)

- base implementation pipe-like message processing daemon
- based on pyzenkit.zendaemon
- Additional features:
  - preconfigured message queue features:
    - automated inclusion and bootstrapping of mentat.daemon.component.filer daemon component
    - additional configurations and command line arguments related to filer protocol.

mentat.daemon.piper (2)

- example implementation can be found in module source code
- documentation: https://alchemist.cesnet.cz/

```
# On Debian Jessie try following (as root):
cd /usr/lib/python3/dist-packages
python3 mentat/daemon/piper.py --help
python3 mentat/daemon/piper.py --no-daemon
python3 mentat/daemon/piper.py --action runlogs-evaluate
```

Remarks

- project is still evolving
- there are many examples directly in the module source code
- use existing modules as templates for creating new ones
- local Makefile may be usefull

Mentat repository structure

- /bin: executables (simple)
- /conf: configuration files and cron scripts
- /lib: Python libraries
- /submodules: local copies of custom libraries

Agenda

Agenda

Overview

- Option 1: real-time processing module can be anything that can work according to the filer protocol
- Option 2: use pyzenkit and mentat frameworks

Agenda

DemoPiperDaemon (2)

```
import pyzenkit
import mentat.const
import mentat.daemon.piper

class DemoPrintComponent(pyzenkit.zendaemon.ZenDaemonComponent):

    def get_events(self):
        return [
            {
                'event': 'message_process',
                'callback': self.cbk_event_message_process,
                'prepend': False
            }
        ]

    def cbk_event_message_process(self, daemon, args):
        daemon.logger.info(
            "Processing message: '{}': '{}'".format(
                args['id'], str(args['data']).strip()
            )
        )
        daemon.queue.schedule('message_commit', args)
        self.inc_statistic('cnt_printed')
        return (daemon.FLAG_CONTINUE, None)
```

DemoPiperDaemon (2)

```
class DemoPiperDaemon(mentat.daemon.piper.PiperDaemon):

    def __init__(self):
        super().__init__(
            name        = 'mentat-demopiper.py',
            description = 'DemoPiperDaemon - Demonstration daemon',
            path_bin    = '/usr/local/bin',
            path_cfg    = '/tmp',
            path_log    = '/var/mentat/log',
            path_run    = '/var/mentat/run',
            path_tmp    = '/tmp',

            default_config_dir    = None,
            default_queue_in_dir  = '/var/mentat/spool/mentat-demopiper.py',
            default_queue_out_dir = None,

            schedule = [
                ('message_enqueue', {'data': '{"testA1": 1, "testA2": 2}'}),
                ('message_enqueue', {'data': '{"testB1": 1, "testB2": 2}'}),
                (mentat.const.DFLT_EVENT_START,)
            ],
            schedule_after = [
                (mentat.const.DFLT_INTERVAL_STATISTICS, mentat.const.DFLT_EVENT_LOG_STATISTICS)
            ],

            components = [
                DemoPrintComponent()
            ]
        )
```

DemoPiperDaemon (3)

```
if __name__ == "__main__":
    DemoPiperDaemon().run()
```

DemoPiperDaemon (4)

- save previous code to file:
  `/etc/mentat/examples/mentat-demopiper.py`
- create configuration file:
  `/tmp/mentat-demopiper.py`
- add module pipeline in:
  `/etc/mentat/conf/mentat-storage.py.conf`
- add module to
  `/etc/mentat/conf/mentat-controller.py.conf`

```
# Create symlink to example:
ln -s /etc/mentat/examples/mentat-demopiper.py /usr/local/bin/mentat-demopiper.py
# Stop all currently running components
mentat-controller.py --command stop
# Start all currently components
mentat-controller.py --command start
# Generate test messages
mentat-ideagen.py --count 10
# View log file
tail -f /var/mentat/log/mentat-demopiper.py.log
```

DemoPiperDaemon (5)

- adding more command line arguments:

```
# Add to DemoPiperDaemon class
def _init_argparser(self, **kwargs):
    """
    :param kwargs: Various additional parameters passed down from object constructor.
    :return: Valid argument parser object.
    :rtype: argparse.ArgumentParser
    """
    argparser = super()._init_argparser(**kwargs)

    arggroup_daemon = argparser.add_argument_group('custom daemon arguments')
    arggroup_daemon.add_argument(
        '--reload-interval',
        type = int,
        default = None,
        help = 'time interval for reloading enrichment plugins in seconds')

    return argparser
```

DemoPiperDaemon (6)

- default values for configurations

```
# Add to DemoPiperDaemon class
def _init_config(self, cfgs, **kwargs):
    """
    :param list cfgs: Additional set of configurations.
    :param kwargs: Various additional parameters passed down from constructor.
    :return: Default configuration structure.
    :rtype: dict
    """
    cfgs = (
        ('something', None),
        ('reload_interval', 300)
    ) + cfgs
    return super()._init_config(cfgs, **kwargs)
```

Agenda

Resources

- Project Mentat: official website
- Project Warden: official website
- IDEA: official website
- MongoDB: official website
- Alchemist: Automated build system

Thank you for your attention

Jan Mach
Jan.Mach@cesnet.cz