Application
Performance
Monitoring in Tryton
1/Why?
1.1/Bugs/crashes?

- Usually crashes are pretty easy to fix
- Other kind of bugs
1.2/Performance

- It is hard to analyze specially on production because it's always **too late**
- Lots of elements:
  - Server hardware
  - Virtualization & OS
  - PostgreSQL
  - Python
  - Trytond
  - Trytond Modules
  - Network
  - Client
1.3/ Specialized instrumentation

- Ideally: should take into account as much layers as possible
- But, let's concentrate on Tryton
2/What is apm.py?
2.1/The patch: Signals

- A patch/hack on trytond/protocols
- Signals:
  - SIGUSR1: Print currently running RPC calls in the log
    - User ID
    - Object & RPC method
    - Postgres PID – APM!
    - Current total execution time of the request
  - SIGUSR2: Print backtrace in the log – great for finding performance issues
2.2/The patch: Profiling

- Profiling using:
  - cProfile – A lot of overhead – should not be used in production (usually)
  - vmprof (from pypy project) – 2~3% overhead
- When?
  - Criteria in trytond.conf (should it be dynamic??)
  - Criteria can include: user, rpc request
- Storage of the trace based on criteria (same as above + total time of request) – stores a Transaction ID
2.3/The patch: Logging

- Logging in JSON format:
  - Transaction ID
  - Time spent on transaction
  - RPC request
  - User
- Only log based on criteria in trytond.conf
2.4/The patch: Friendly user message for crashes

- Similar to Sentry
- Give the user a “Transaction ID” for tracking the error
- Maybe we should convert the Transaction ID into a docker-like string: adjective_name such as angry_einstein
3/What about data gathering?
3.1/Recull

- Python
- Send and store in JSON format
- Pick information from servers to RabbitMQ server
- Store that information and query from the command line
http://www.NaN-tic.com

Albert Cervera i Areny
albert@nan-tic.com
@albertnan
linkedin.com/in/albertca
Nan-tic
Perquè vols canviar, i canvis.