

# **Implementation of an email-based alert system for large-scale system resources**

Robert Poenaru

*Department of Computational Physics and Information Technology, IFIN-HH*

**#roedunet2021**

# Table of Contents

1. Motivation
2. Aim
3. Development Stages
4. Conclusions

# Motivation

Within a research department:

## Scientific community

- Tackle different problems
- Construct a codebase for a particular issue
- Develop a scenario for executing simulations
- Request access to computing resources (submit jobs)

## System administration community

- Manage allocation of the computing resources for each job
- Monitor executing simulations
- Monitor idling resources
- Keep track of incoming jobs

# Simulations

Scientific community

- *Unoptimized* simulations lead to:
  - Long execution time (will cause delays in the pipeline)
  - Low degree of parallelism (cannot take full advantage of multiple core/threads)
  - Excessive memory consumption (limited resource)
- Simulation testing + optimization is required

# Resource management + monitoring

Sysadmin community




- Allocate jobs (e.g., simulations) to the computing cluster
- Manage computing nodes (updates, services)
- Observe *unexpected* behavior of the running simulations
- Check *idling* resources for potential issues



- Keeping track of all these aspects 24/7 is very challenging

# Project Goals

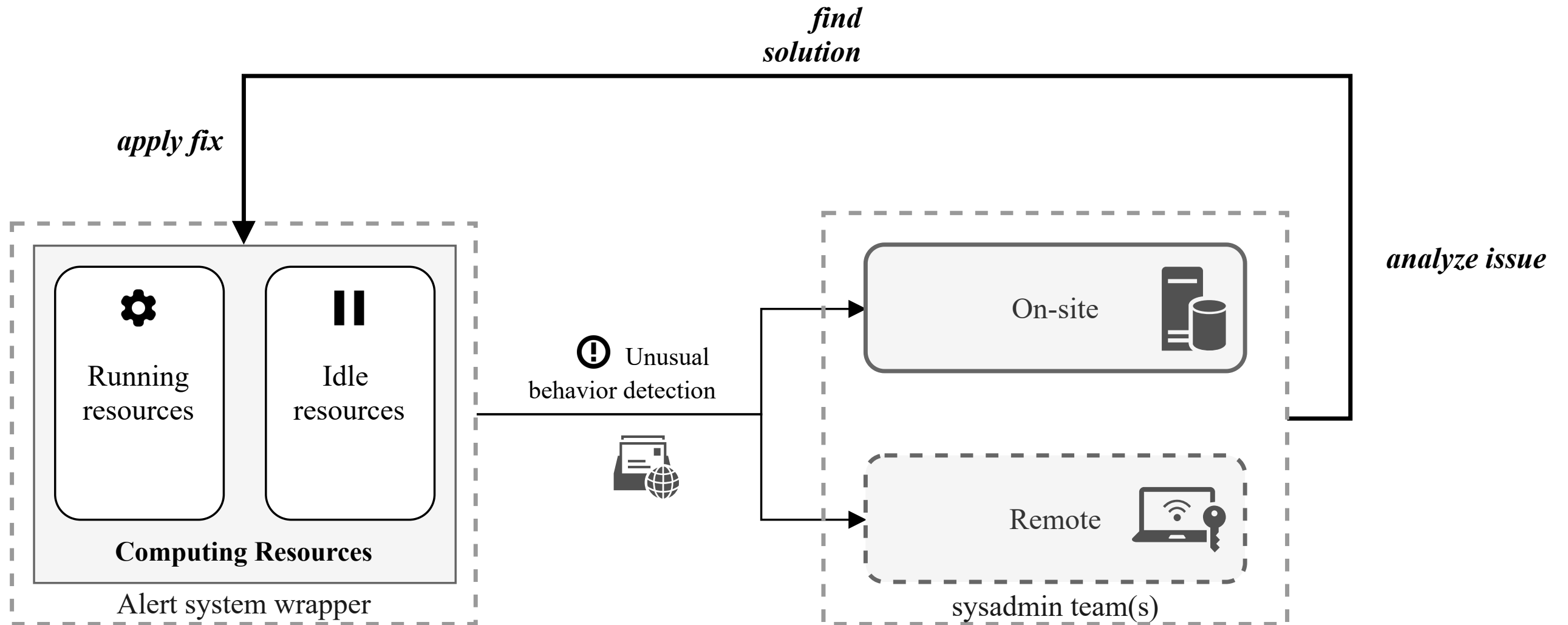
- Create a service which:

1.  Monitor multiple computing nodes/clusters (system resources, executing services, etc.)
2.  Identify potential issues within the resources
3.  Inform the sysadmin in realtime on the occurring issue(s) - via e-mail

*Alert system*

# Alert system

## General workflow



# Alert system

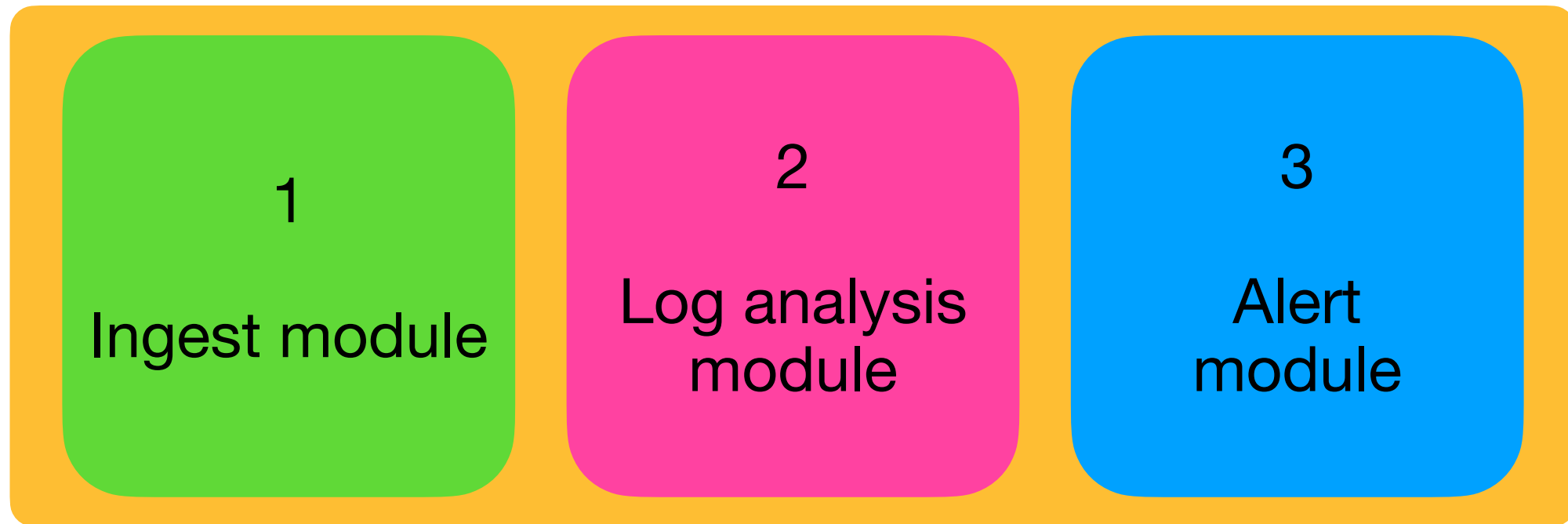
## Main features

- Developed in Python
  - Great system compatibility
  - Plenty of packages
  - Strong development community
- Works with virtual environments
- Improved package management using `pipenv`
- Built in a modular way



# Alert system

## Modules



1. Get the incoming log information from its corresponding file(s)
2. Perform analysis on the ingested log data
3. Decides if alert(s) should be sent to the sysadmin

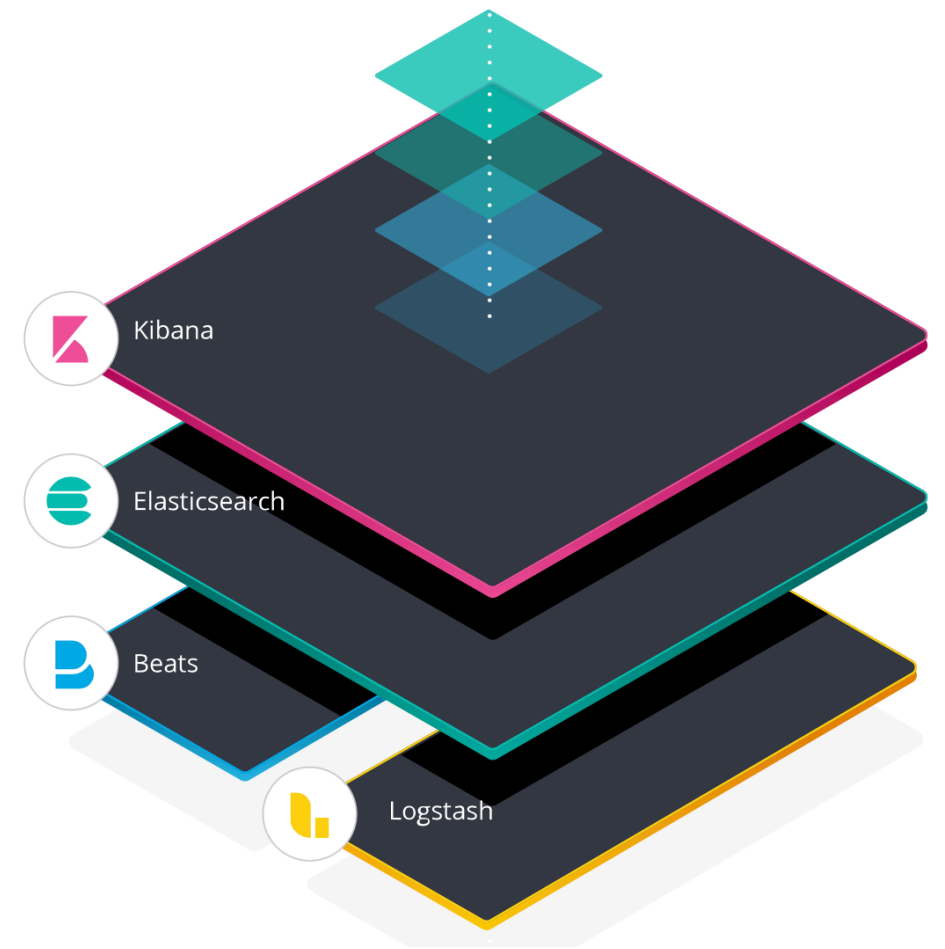
# Receiving system information

- The underlying computing infrastructure must send information containing its current status
- Each node on the cluster should send system information (e.g., CPU usage, RAM usage, network activity, running services) to a centralized *master node*
- The alert system runs on the master node
- Information is send as log files, via a *log shipper*.

# Filebeat

## Log shipper

- Developed by Elastic™
- Part of the Elasticsearch stack (ELK)
- Lightweight shipper for logs
  - Runs as a service on the system
  - Sends logs to (not only) any other node on the network



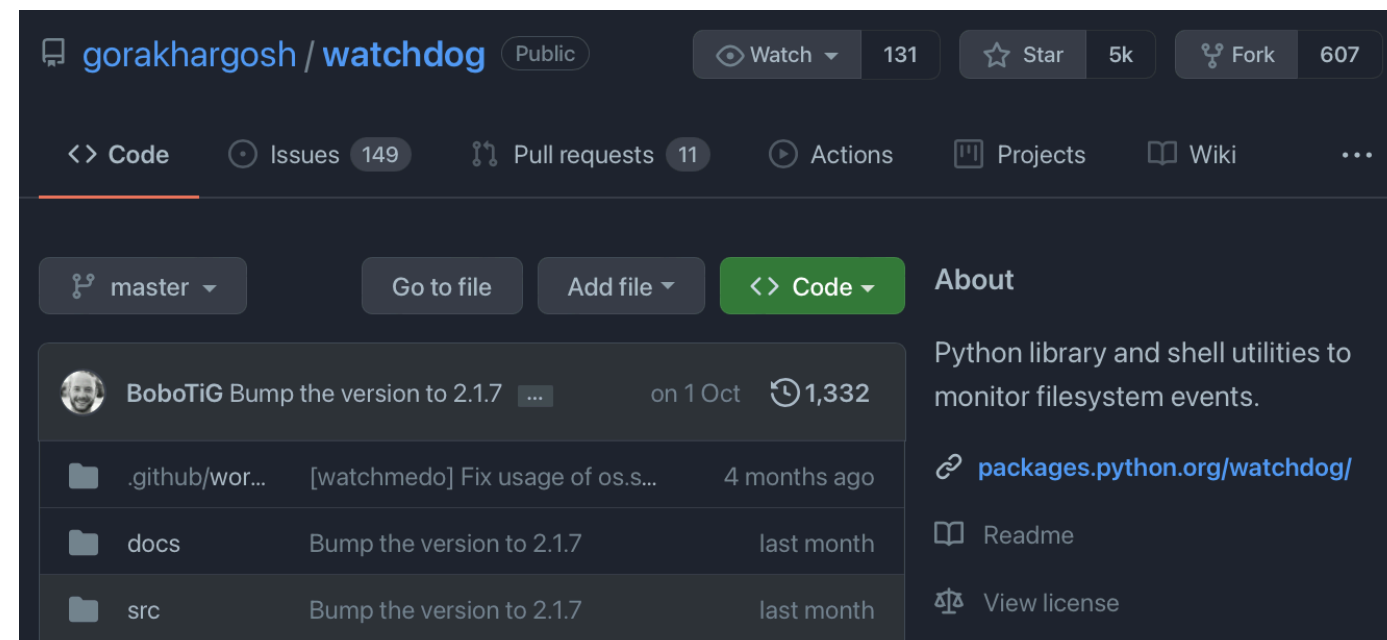
# Log monitoring 1

## Watching log file(s) for changes



- The master node runs the python alert service
- Module-1 reads the log files that arrive on the master node
- Module-1 is able to read every new log entry as it arrives. This is possible using the watchdog package.
- Watchdog uses so-called event handlers to keep track of any changes in the monitored files

```
event_handler = LoggingEventHandler()
observer = Observer()
observer.schedule(event_handler, path, recursive=True)
observer.start()
try:
    while True:
        /* do something */
finally:
    observer.stop()
    observer.join()
```



# **Log monitoring 2**

**Parse & collect data**