Prophasis - An IT Infrastructure Monitoring Solution - Interim Report

Cameron Gray

Fourth Year Project Report

School of Informatics

University of Edinburgh

2016

Abstract

Prophasis is an IT infrastructure monitoring system that is designed to suit small to medium size businesses where a system needs to be intuitive to manage. Management of the entire system can therefore be handled from a single, responsive web interface. It is also suitable as a one-stop tool with support for both time series monitoring in addition to real time alerting. Traditionally two different tools would be needed to gain this level of monitoring.

Table of Contents

Chapter 1

Introduction

1.1 Background

In recent years, almost all businesses have been expanding their IT infrastructure to handle the modern demand for IT systems. As these systems grow and become increasingly important for business operation it is crucial that they are sufficiently monitored to prevent faults and periods of downtime going unnoticed. There is already a large market of tools for monitoring IT systems however they are designed for use on massive scale networks managed by teams of specialised systems administrators. They are therefore complicated to set up and manage and multiple tools are often required to gain a suitable level of monitoring.

For example, tools generally either fall into the category of real time alerting (i.e. telling someone when something breaks) and time series monitoring (i.e. capturing data about the performance of systems and presenting graphs and statistics based on it), there is a large gap in the market for tools that provide both of these in one package. This reduces the time required to manage the system as it eliminates the need to set up and configure two completely separate tools.

These tools are also generally managed and configured through various configuration files split across different machines on the network. This means that in order to efficiently use these tools a configuration management system such as Puppet must be used. In a small business with limited IT resources, a completely self contained system is often preferable.

1.2 Improvements

Prophasis is designed for use in a small to medium business with limited IT resources. They may have a small IT team with limited resources or may not even have a dedicated IT team at all, instead relying on one or two employees in other roles who manage the business's IT systems on the side of their regular jobs. Therefore the system needs to be quick to deploy and manage with a shallow learning curve. In order to use the system efficiently there should be no requirement for additional tooling to be deployed across the company.

1.2.1 Configuration Management

It should be possible to manage the configuration of the system from a single location. Prophasis therefore provides a responsive web interface where every aspect of the system's operation can be configured, Prophasis then handles distributing this configuration to all other machines in the system in the background. Custom code for plugins is handled in the same way; it is uploaded to the single management machine and is then automatically distributed to the appropriate remote machines when it is required.

1.2.2 Time Series Monitoring & Real Time Alerting

Prophasis provides both the ability to alert administrators in real time when a fault is discovered with the system alongside functionality to collect performance metrics over time and use this data to generate statistics about how the system has been performing. This time series data can be used to both investigate the cause of a failure in post-mortem investigations in addition to being able to be used to predict future failures by looking at trends in the collected data.

1.2.3 Expandability

It is important that a monitoring tool can be expanded to support the monitoring of custom hardware and software. An example of this would be hardware RAID cards. Getting the drive health from these types of devices can range from probing for SMART data all the way to communicating with the card over a serial port. It is therefore crucial that Prophasis can be easily expanded to support custom functionality such as this. Therefore Prophasis supports a system of custom ”plugins” which can be written and uploaded to the monitoring server where they can then be configured to monitor machines. These plugins are designed to be self contained and to follow a well defined and documented structure. This provides scope for a plugin ”market-place” much like there already exists for plugins for software such as Wordpress and Drupal allowing plugins to be easily shared and installed for monitoring various pieces of hardware and software, therefore eliminating the need for every user to implement custom monitoring code for the systems they are using.

Chapter 2

Interim Report

2.1 Current Progress

2.1.1 Agent

I have built a complete monitoring agent which runs on remote hosts being monitored. This exposes an HTTPS API which is used to interface with the agent. The agent can receive plugin code over the network to allow the installation and update of custom plugins and can then execute them in order to collect data about the machine. This data is then returned using the API.

2.1.2 Core

The core runs on the node doing the monitoring and communicates with the remote agents in order to trigger and collect data from checks. It also handles scheduling to run checks at the appropriate times.

The core is multi threaded so it can quickly check multiple hosts at the same time, this prevents long running checks from slowing down the system as a whole. A new thread is spawned for every machine that is currently being checked and then each thread will run each check on that machine sequentially. This means that there is no risk of multiple checks being performed on a single host at any one time, this prevents issues such as checks conflicting or overloading the machine being monitored.

2.1.3 Web

I have built a large portion of the web interface used to manage the system, this provides an easy to use interface for configuring the monitoring as well as dashboards that show the results of checks and to highlight any issues. The web interface was built to be responsive from the start and therefore works extremely well on mobile devices without any loss in functionality.

Plugins

I have designed a structure and interface for the plugins that are used to actually perform the checks on the remote machines. I have then built several sample plugins to demonstrate their capability to monitor different aspects of a machine such as the CPU load, memory load, check if the machine will respond to a heartbeat signal and to check the status of a Linux MD raid device.

One challenge I found was how to build a system to classify the result of a plugin into whether it is ”ok”, ”critical”, ”minor”.etc. My initial plan involved a web interface with various options however in order to make this reasonably powerful it started become extremely cluttered and complicated. I therefore designed a system where Lua code can be written to classify the data. This is therefore very flexible and can be written at the same time as the rest of the plugin but can be customised on a per host basis (a VM host for example may have higher tolerances for memory usage compared to other types of machine). The difficulty here was that I wanted this code to be editable from the web interface but if I were to use something such as Python this creates a large security risk due to being able to access core system APIs. I therefore settled on Lua as it is designed as an embedded language so therefore provides simple functionality for creating a sandbox execution environment with only explicitly assigned functions from being available in the code.

2.1.4 Alerting

I have designed a structure for and started work on implementing the web interface for managing alerts, i.e. deciding when to send out alerts related to changes within the monitoring system.

2.2 Future Work

2.2.1 Finish Alerting

Now that the web interface for managing alert conditions is complete I need to decide on how to support different methods of sending alerts, ideally through some sort of plugin system, this means I can not only use simple emails for alerts but I can also have scope for external services such as Twilio for SMS and telephone alerting and Pushbullet for push notifications. One this design is decided on I can then implement it and therefore complete the alert functionality.

2.2.2 Improve Dashboards

There are currently a couple of simple dashboards for viewing the data collected but this needs to be improved to be able to present the data in various different ways such as being broken down by service or check. I will also investigate allowing the user to customise dashboards to their preferences.

2.2.3 Distributed Monitoring

Currently Prophasis (along with many other monitoring systems) only runs on a single host, this means that if this host or a network link between it and the machines being monitored fails, the monitoring will stop working. I would like to investigate the possibility of distributing this so there can be multiple machines performing the monitoring at the same time and then have the ability to easily recover (either automatically or manually) if a failure occurs somewhere in the monitoring system.

This could initially be accomplished by designating each monitoring node a set of machines that it is responsible for monitoring, this prevent conflicts caused by multiple nodes monitoring the same machine. Data could then be kept in sync through using database level replication. Care however would need to be taken to prevent conflicts between different database copies. The simplest way to do this would be to have only one machine that is allowed to update the configuration at any one time, all other monitoring nodes are simply allowed to insert the data they have collected and update data about the schedules for the hosts they are assigned to monitor. This will ensure that no two machines will update the same database record at the same time.

Another tricky aspect of distributing the monitoring is how to handle failures of a monitoring node. There are already well established algorithms that can be used to handle this which I would look into however an initial version of this could simply rely on the user manually clearing up after a failure where the machine that failed would no longer monitor the hosts it's assigned but the rest of them will continue to monitor their respective hosts as normal.

2.2.4 Documentation and Packaging

Due to the complex nature of this system I will need to write comprehensive documentation about setting up, managing and using the system. This will range from how to simply use the web interface down to the level of technical documentation about how to write custom plugins.

I will also need to package the application up nicely in a distributable way with some sort of installer/setup script. Currently I work by simply manually executing the Python source files inside a virtual environment which is ideal for development work but is not well suited for production use. I will also need to investigate swapping out the Flask development web server that currently powers the web interface for something more production ready.

2.2.5 Plugin Signing

Code for plugins is distributed to the agent from the monitoring server, this means that if the monitoring server were to be compromised, an attacker could use it to send malicious code to and execute the code on the remote hosts. I would like to investigate being able to GPG sign the plugin archives so that even if an attacker gains access to the monitoring host, they cannot send their own malicious code to a remote machine.

Chapter 3

Design

3.1 Technology Choice

3.1.1 Why Python?

3.1.2 Why HTTPS?

3.2 System Structure

- Explain different components
- Diagram
- Why separate?

3.3 Monitoring Methodology

3.3.1 Host Management

- Stuff about hosts and host groups

3.3.2 Plugins

- What is a plugin?

3.3.3 Checks

- What is a check?

3.3.4 Schedules

- What is a schedule?

3.3.5 Services

- What is a service?
- Dependencies & Redundancy Groups
- Why use dependencies?
  - Alert only if service functionality is severely impacted
  - Prevents unnecessary alerts due to failures in redundant infrastructure
  - Provides clearer view of impact of a given failure

3.3.6 Alerts

- What is an alert?
- Separate alerts for different checks/hosts/plugins/services
- State transition restrictions - Reduce unnecessary alerts

Chapter 4

Implementation

4.1 Code Structure

- Separate applications for each task - Why?
- The ”common” package

4.2 Plugin Interface

- Explain structure of a plugin
- UML Diagram?
- Why Lua for classification logic?

4.3 Database Design

4.4 Web Interface

- Go into detail about web structure WRT templates, assets.etc

4.5 Core

- Scheduling
- Multi-threaded dispatcher

4.6 Agent

- Communication
- Authentication

Chapter 5

Testing

- Unit testing
- Use within Tardis & Lynchpin

Chapter 6

Evaluation

Chapter 7

Conclusion