MIKELANGELO

D5.7

First report on the Instrumentation and Monitoring of the complete MIKELANGELO software stack

Workpackage	5	Infrastructure Integration	
Author(s)	John Kennedy		INTEL
	Marcin Spoczynski		INTEL
Reviewer	Benoît Canet		ScyllaDB
Reviewer	Gregor Berginc		XLAB
Dissemination Level	Public		

Date	Author	Comments	Version	Status
9 Dec 2015	John Kennedy	Initial merge of contributions	V0.0	Draft
13 Dec 2015	John Kennedy, Marcin Spoczynski	Document ready for review	V1.0	Review
23 Dec 2015	John Kennedy	Document ready for submission	V2.0	Final

Executive Summary

The MIKELANGELO project [1] seeks to improve the I/O performance and security of Cloud and HPC deployments running on the OSv and sKVM software stack. The project requires a rich instrumentation and monitoring solution to help identify and isolate opportunities for enhancements, and to measure the improvements resulting from any such changes,

After a consortium-wide requirements gathering exercise and a review of the state-of-the-art in instrumentation and monitoring systems, the decision was made to leverage the evolving INTEL snap [2] open-source telemetry project to help meet the needs of the MIKELANGELO project. The framework and extensive list of capabilities delivered by snap would be immediately available to MIKELANGELO, whilst any necessary additional functionality could be developed and contributed as need.

INTEL's MIKELANGELO resources have since developed functionality that can capture rich data from libvirt - the linux virtualisation API that supports many Linux hypervisors including the sKVM hypervisor in MIKELANGELO - and OSv - the high-performance cloud guest operating system being further enhanced by the project.

All of these plugins have been successfully integrated into snap in time for the initial open-source release of the project on December 2nd 2015. With that release, MIKELANGELO can now capture, process and publish the rich hardware, operating system and other metrics that the complete range of snap plugins now provides.

MIKELANGELO is well placed to leverage the contributions of the snap community going forward, and equally looks forward to contributing - open-source - additional functionality to progress the state-of-the-art as the instrumentation and monitoring needs of the MIKELANGELO use-cases evolve over the lifetime of the project.

Acknowledgement

The work described in this document has been conducted within the Research & Innovation action MIKELANGELO (project no. 645402), started in January 2015, and co-funded by the European Commission under the Information and Communication Technologies (ICT) theme of the H2020 framework programme (H2020-ICT-07-2014: Advanced Cloud Infrastructures and Services)

Table of contents

Table of Figures

Table of Tables

1 Introduction

The MIKELANGELO project seeks to improve the I/O performance and security of Cloud and HPC software running on the OSv and sKVM software stack. To identify candidate subsystems and code modules for modification, and to measure the improvements resulting from any such changes, a rich instrumentation and monitoring solution is required.

This document describes the steps taken in the first year of MIKELANGELO to deliver a sufficiently powerful Instrumentation and Monitoring system. The task assigned to this activity - Task 5.3 Instrumentation and Monitoring - officially started in Month 6 of the project (June 2015) and is scheduled to continue until Month 36 (December 2017).

The progress up to Month 12 of this task (December 2015) is summarised in the following chapters. Chapter 2 presents the key initial requirements identified for instrumentation and monitoring in the MIKELANGELO project, as well as some general considerations. These requirements were gathered in Work-Package 2 activities and serve as the input to this entire effort.

Chapter 3 presents a State-of-the-Art of current Instrumentation and Monitoring systems, with a focus on some of the key requirements that the MIKELANGELO project has identified.

Chapter 4 introduces the architecture of the instrumentation and monitoring solution selected for the MIKELANGELO project. It is based on the snap telemetry framework, the initial version of which was fully open-sourced by Intel on December 2nd 2015 [2]. MIKELANGELO engaged with the snap team from the initial scoping phase of the project. Concretely MIKELANGELO also contributed several key modules of code that were included in the initial open-source release.

Chapter 5 details the implementation activities to date in this task. Specific functionality which was developed by this task is explained. This includes the ability to collect metrics from KVM, sKVM and indeed other hypervisors through libvirt, the ability to collect metrics from the OSv guest operating system, and the ability to publish telemetry data to a PostgreSQL database,

Chapter 6 presents some observations on the instrumentation and monitoring system developed to date, and introduces current plans on how it will be further enhanced through the remainder of the project.

Chapter 7 summarises the key takeaways from this task to-date, Chapter 8 provides some concluding remarks and references are provided in Chapter 9.

2 Requirements

2.1 Introduction

A powerful instrumentation and monitoring framework is required by the MIKELANGELO project to measure and confirm any changes in performance of the MIKELANGELO software stack across any of its four use cases.

An initial requirements gathering exercise was performed by MIKELANGELO in work-package 2 during the first six months of the project as documented in Deliverable D2.19, The first MIKELANGELO Architecture [3]. These project-specific requirements are now discussed, along with some more general requirements that any cloud or HPC instrumentation and monitoring solution should consider,

2.2 Hardware Metrics

It will be important to capture detailed information about the performance of the hardware hosting the systems under test. This includes detail on physical attributes such as power consumption and temperature, as well as system performance data on devices such as CPUs, memory, cache, drives and network interfaces.

2.3 Hypervisor Metrics

As MIKELANGELO is focused on improving the performance of Cloud and HPC platforms, it is critical that the precise performance of any hosted virtual machines and containers is measurable. Whilst these measurements could be achieved by deploying software probes inside virtual machines or containers, for efficiency this data should be extractable from the hypervisor layer or the container-hosting environment of the host operating system.

2.4 Guest OS Metrics

The MIKELANGELO project has selected OSv [4] as the default guest operating system and so to ensure performance is at least maintained if not improved by enhancements, MIKELANGELO must be able to retrieve and review detailed metrics from inside any OSv instances hosted by a deployment. This data from inside the hosted VMs complements the data from the hypervisor that hosts them.

2.5 Hosted Application and Service Metrics

It is important that the entire software stack of MIKELANGELO is instrumented and monitored to allow application-level performance to be compared with the performance of the underlying system - software and hardware. Thus there is a requirement that the

MIKELANGELO monitoring system be flexible enough to allow arbitrary hosted applications and services to be monitored. Note that hosted applications and services may be written in any language, and deployed on any middleware that the Cloud Service Provider chooses to support.

2.6 Monitoring GUI

For usability reasons it is important that the instrumentation and monitoring system in MIKELANGELO supports a rich graphical user interface (or interfaces) allowing quick and powerful querying of data gathered, as well as insightful visualisations.

2.7 General requirements

Although not proposed as explicit requirements for the instrumentation and monitoring framework in year 1 of the project, the following more general requirements are also deemed highly relevant.

2.7.1 Performance

The instrumentation and monitoring system must be architected and implemented in such a way as to minimise the overhead of telemetry gathering and processing on the systems being measured. Additionally, measures should be taken to reduce the overall load of the telemetry, whilst maximising the usefulness of the data.

2.7.2 Scalability

The instrumentation and monitoring system must be designed and built with scalability in mind: deployments of thousands of nodes and more should be readily supported. An important implication of this is that the system should be manageable at such scale: suitably powerful deployment, configuration and management facilities should be available.

2.7.3 Extensibility

With a wide range of powerful monitoring tools already available, both open and close source, it is highly desirable that the MIKELANGELO instrumentation and monitoring system be highly extensible and able to leverage them. It should be possible to use external third-party tools as sources of telemetry data, destinations of telemetry data, and analytical and processing engines. It should also be possible to expand or contract the instrumentation of monitored systems dynamically to minimise impact and management overhead.

2.7.4 Security

Appropriate security must be considered from the outset. It must be possible to verify the integrity of the various components of the telemetry system, be they at the edge or back-end, and it also must be possible to secure communications between the various components.

2.7.5 Openness and Licensing

The MIKELANGELO consortium has agreed in principal to open-sourcing results where possible. To support community adoption and engagement the MIKELANGELO instrumentation and monitoring system should be open-source, and released under a community-friendly license. It should be noted that licenses such as the GNU General Public License [5] force users of the software to open-source any functionality they wish to develop and distribute. This may not suit all potential adopters. A license such as the Apache License Version 2.0 [6], which gives users the option of distributing both open-source and closed-source solutions, may be more appreciated by the community.

3 State of the Art

3.1 Introduction

Before selecting or designing an instrumentation and monitoring system to meet the needs of MIKELANGELO, existing tools in this space were reviewed for suitability. The state-of-the-art review in this chapter builds on previous state-of-the-art reviews that the INTEL team have performed in the past, e.g. in the IOLANES project [7],

Despite the large number of instrumentation and monitoring systems readily available [8], [9], most monitoring systems can be shown to be composed of the core building blocks illustrated in Figure 1.

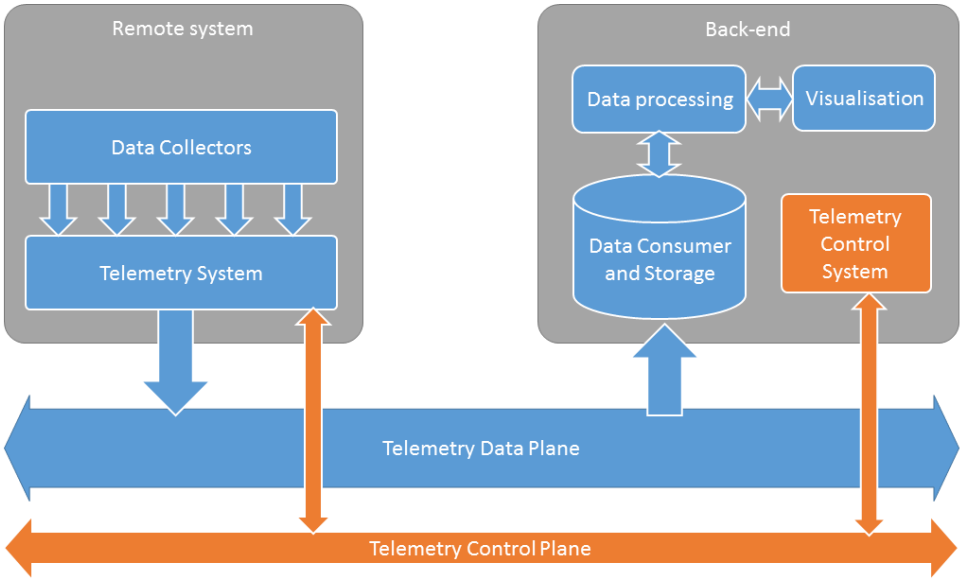

Figure 1: Basic building blocks of a typical monitoring system.

These various basic building blocks have more or less prominence and functionality depending on the specific application domain. This is due to the number of important and complementary goals [10], [11] such as looking at minimising power consumption, maximising resource utilisation, speeding up elasticity, delivering more predictable performance, improving customer satisfaction (QoS, SLA), and so on. Whilst monitoring systems are typically passive systems designed to capture and understand the state and performance of a distributed system, they are a fundamental source of data for troubleshooting, scheduling and orchestrating tools and thus often support both manual and automated interventions.

With these basic concepts in mind, we now survey some of the more popular monitoring systems from the following key perspectives:

- Manageability: does the system support auto-discovery of services and resources? Can we hierarchically group metrics for easier management?
- Extensibility: is there access to source code to customise data collectors?; is the data collection and publishing clearly decoupled from processing? Can we easily incorporate third-party collectors and databases?
- Scalability: does the system allow processing at the edge? How scalable is the backend? Can we use multiple dispatchers at once?
- Security: how robust is the system to standard attacks? How secure is the data being transmitted? Can the data returned be trusted?
- Visualisation: does the system assist in reducing complexity? How customisable are the visualisations?
- Analytics: can a function or filter be applied to data? Can summaries across different timeframes be presented? Can custom modules be easily invoked for root cause analysis?

While all monitoring systems share the same basic concepts, they are designed for different users in mind. Specific requirements exist for industrial telemetry, digital and analog signals, and aeronautics for example [12]. In the following sections we go through all of these aspects and introduce some of the more relevant open source and commercial monitoring tools, referring to relevant literature when appropriate.

3.2 Manageability

Monitoring activities can be highly demanding in terms of consumed bandwidth and data storage, but also in terms of the personnel required to oversee them. It is difficult to determine these costs before establishing a monitoring solution. Manageability refers to how easy it is to initially deploy and configure a functional solution, as well as how easy it is to maintain in the long run. An effective monitoring tool needs to be able to provide a complete overview of a monitored infrastructure and also provide tools to more easily control that infrastructure. One of the more helpful monitoring tools is auto-discovery. Auto-discovery automatically identifies nodes and resources within the datacenter that can be monitored. This feature is present in various systems, including OpenNMS [13], Zabbix [14] and Spectrum [15], Figure 2 illustrates a typical auto-discovery mechanism.

Simplifying management further, tools such as the clustering of nodes with similar monitoring configurations and needs allows arbitrarily large numbers of nodes to be managed via a single command. For example the tribe implementation in snap repeats any command issued on any node in a tribe cluster to all other members of that tribe cluster.

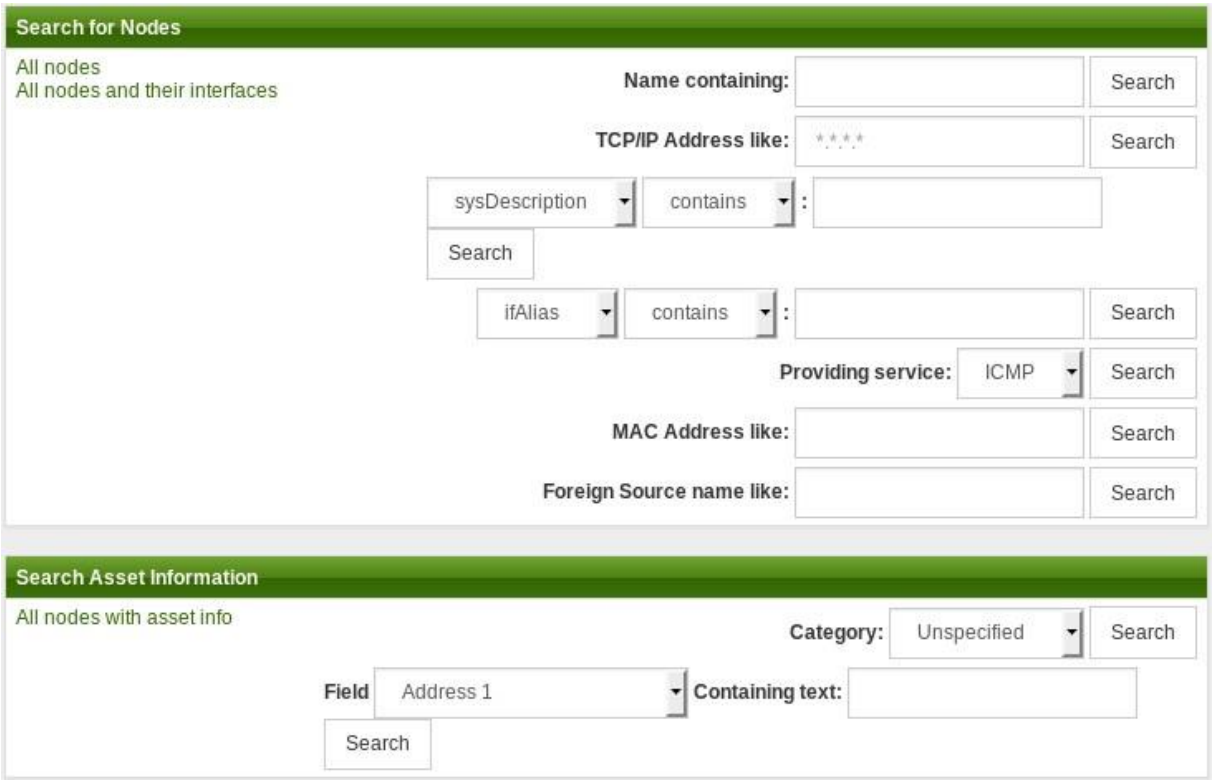

Figure 2: Auto-discovery in OpenNMS. A range of IP addresses and protocols can be searched.

Management of metrics is also an important feature of modern monitoring solutions. It should be possible to list all available metrics for a given host and decide and select which are and which are not applicable to a given monitoring scenario. Monitoring packages such as Zabbix and snap allow the querying of metrics over a RESTful API, which allows the metrics not only to be accessed manually but also as part of an automated solution, often desirable in cloud and HPC deployments.

Changes to the monitored hosts, networks or monitoring servers regularly require an update to the monitoring agent, thus remote management of the agents is an important requirement. These changes may include adjusting sample times, updating transmission protocols, redirecting metrics and changing the edge data processing algorithms. Ganglia [16], Prometheus [17], and Zabbix allow this type of remote management of the monitoring agents, with others such as snap and Sensu [18] providing APIs to programmatically set these parameters without the need for human intervention. This automated ability is particularly useful in cloud environments where schedulers and analytics engines can focus investigation of resources dynamically.

3.3 Extensibility

An extensible monitoring system is one that can adapt and expand to changes in the infrastructure and also the needs of the user. The most common adaptation required is the collection of new or previously unforeseen metrics. Many monitoring systems such as Ganglia and snap allow for this type of extension by updating the monitoring agents on the hosts or by adding data collection plugins. Snap can allow for plugins to be remotely deployed to a server without access to the host or without the need to restart the monitoring agent. These type of low impact changes are favourable in environments where many services are depending on the same host and where a constant stream of metric data is required.

Interfaces to storage systems are another type of extension which monitoring systems often support. Such extensions can be developed to send metric data over any protocol, placing it on any device, using any persistence technology. Some monitoring systems such as snap allow the metric data to be replicated across multiple storage devices and technologies at once, reducing the overall processing required when multiple back-end systems need to be updated..

3.4 Scalability

The differences between installing a basic monitoring tool for testing purposes in a limited proof of concept deployment, and deploying a full-scale monitoring solution spanning thousands of devices spread across many locations in a production environment are significant. Scaling can be expensive, with associated hardware, software licensing and personnel costs.

To scale effectively, all aspects of a system must be carefully considered. In a typical monitoring system this includes client-side agents, publishers, receivers, maintenance and storage facilities.

Back-end data stores can become an issue when monitoring deployments scale up. While SQL databases can produce excellent results for diverse monitoring systems [19], their performance, can significantly degrade when scaled to a few thousand devices. Some SQL databases have been developed to deliver performance at scale [20], [21], however alternatives such as Hadoop based back-ends have also been successfully adopted by some monitoring systems such as OpenTSDB [22].

Configuration can become an issue at scale also. It is common for agent configurations to require alteration from time to time. Sampling rates may need to be updated or agent plugins may need to be changed. It can be straight-forward to reconfigure a single machine, but when dealing with large data-centres manual server-specific configuration becomes

unwieldy and prone to error. Thus many monitoring systems allow mass updates of configuration. Zabbix, Sensu and Nagios allow servers to be grouped together and configuration actions applied to all in the group. The snap peer-to-peer clustering model called 'tribe' has been mentioned previously: users push configuration and plugins to any node in a tribe cluster and it is automatically replicated peer-to-peer to all other nodes within the group.

As a general principal, monitoring systems which are loosely coupled allow scalability problems to be isolated relatively easily when they do arise, and focused solutions to then be explored.

3.5 Security

Although priority is often given to securing resources which directly handle customer data, monitoring systems also need to be secured as attackers could use monitoring data to derive useful insights into the operations and infrastructure of the cloud, and the software that is being hosted.

Monitoring systems usually suggest best practices to keep monitoring data secure, such as closing firewalls, configuring VPN proxies, and ensuring all dependency software is patched and operating at the highest level of security. But more complete systems are beginning to bake the security directly in, with particular attention paid to data transmission. Secure communication is now offered by most newer monitoring systems. Nagios and snap use encrypted channels to send telemetry data, while others such as Sensu validate each request to the agent. The rapid growth of third-party agent plugins also presents a potential backdoor to attackers, this fear has resulted in the use of signed plugins to validate plugin authenticity in some systems such as snap.

Recovery from an attack is also addressed by the latest monitoring systems, with both snap and Prometheus caching metrics in the event that a denial-of-service attack disables the communication link connecting a monitored server to a back-end system. Once a connection is re-established then the cached metrics are flushed to the back-end.

3.6 Visualisation

Well designed visualisations can transform the usefulness of a monitoring system by quickly highlighting the key information that an operator needs to see. This is very relevant in large scale instrumentation and monitoring systems and the volumes of resources and data being monitoring can be overwhelming. Typical visualisations used in monitoring scenarios include graphs, maps, dashboards and event notifications.

Figure 3: An example monitoring dashboard implemented in Grafana.

Some monitoring solutions come with built-in graphical user interfaces supporting many of these visualisation tools. Prometheus has a modern, clean and scriptable visualisation tool built in whilst Zabbix also supports visualisations of security threats across the entire data centre. Others tools such as Ganglia and Nagios [24] come without a default front-end, but have a very active community providing powerful open-source visualisation tools on top of them.

Many monitoring stacks include third-party visualisation tools that are not necessarily targeted at monitoring. Graphite [25], D3.js [26] and Grafana [27] are examples of these more generic tools designed to quickly build arbitrary, focused, visualisations. Although they allow the user to create whatever visualisations they consider appropriate, the results may not embrace best practice in visual analytics.

3.7 Analytics

Dashboards are often not powerful enough to identify, troubleshoot and prevent outages. Simple interfaces attempting to display multiple metrics at once, e.g. as Figure 4 illustrates, may create additional problems for the monitoring operator. Visualisation of raw data is useful, but automatically identifying and highlighting problems improves manageability significantly. Monitoring systems often support automated alarms and escalations, and colour-coding in dashboards can highlight priority events. Powerful analytical techniques such as anomaly detection and dimensional reduction can power such tools.

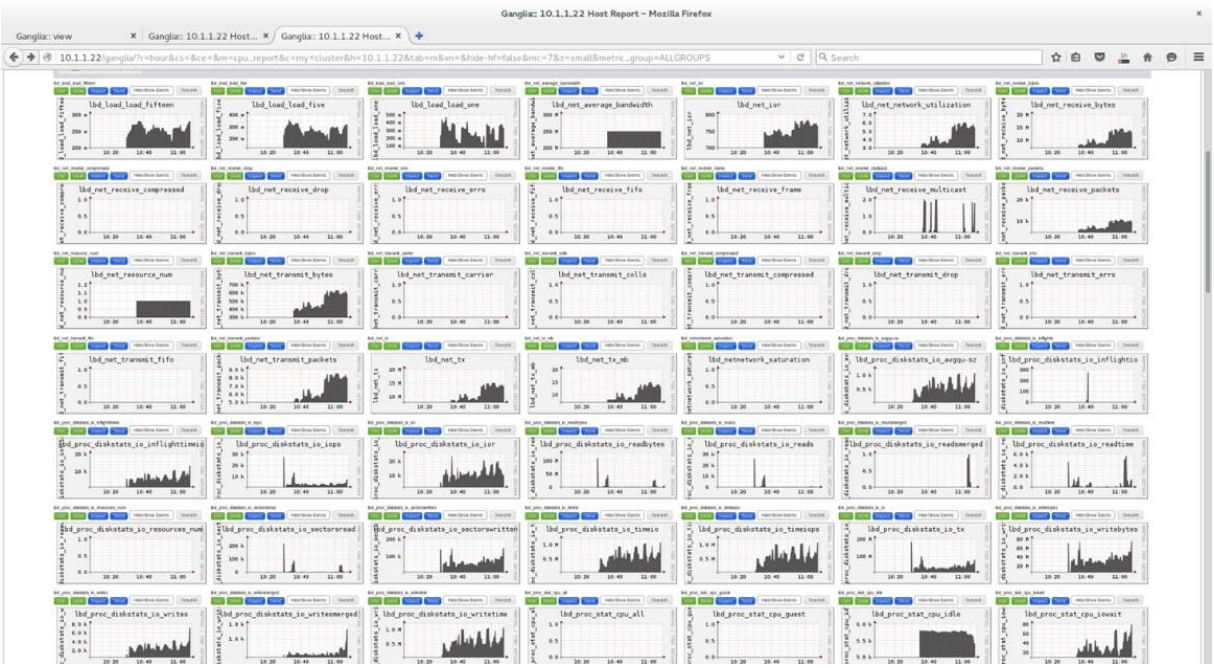

Figure 4: A view of metrics using Ganglia.

Beyond basic alarms, most network monitoring software stacks lack predictive and root cause analysis tools. Some notable exceptions include Spectrum, IBM Tivoli [28], HP SiteScope [29] and VMWare [30]. The growth rate at which resources are consumed can provide valuable input to operators [31]. This is particularly true for large deployments, where the analytics displayed in dashboards comes to the fore.

Hierarchical and graph-based views of metrics and their associated metadata can also assist analysis. Spectrum uses these concepts as illustrated in Figure 5 to help operators quickly understand the location, nature and impact of any issue.

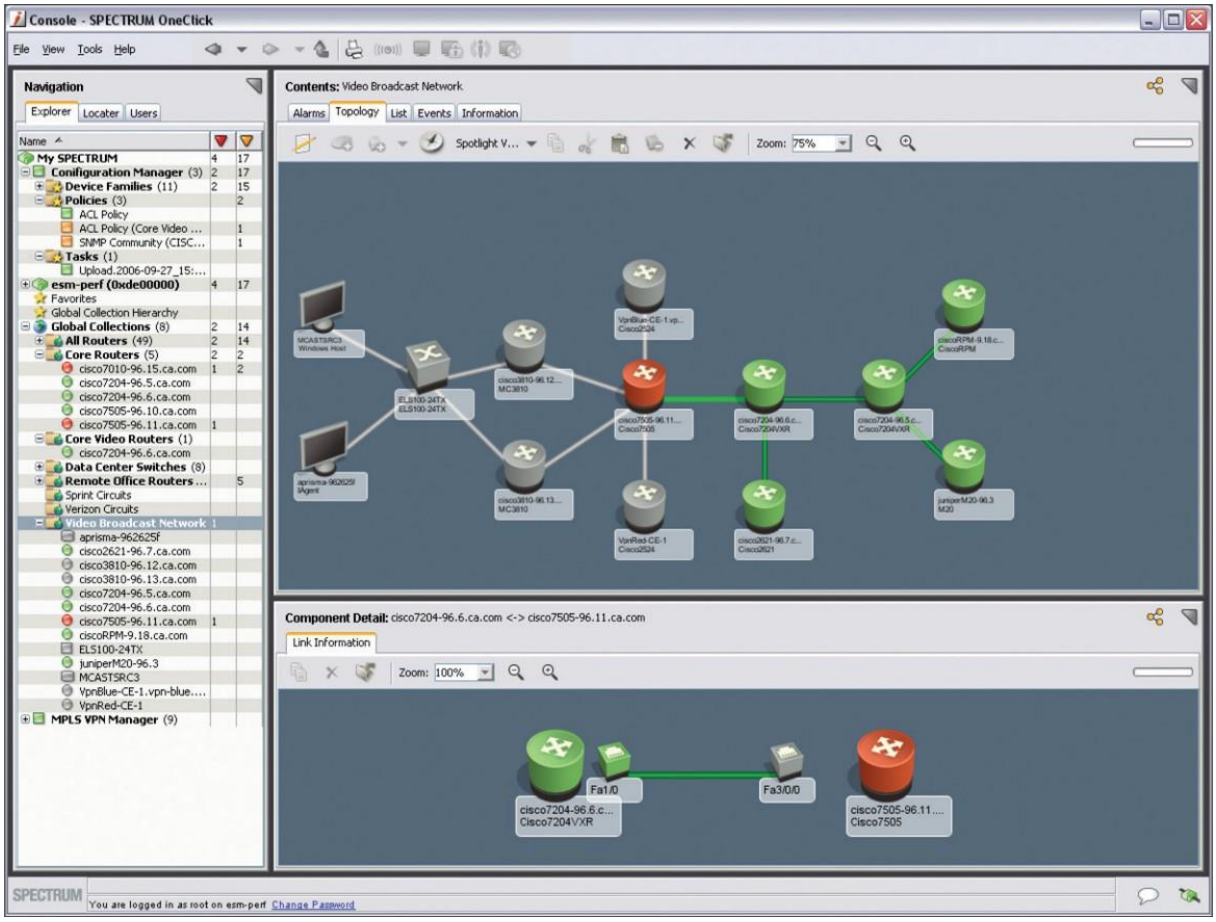

Figure 5: Hierarchical and graph representations can assist root-cause analysis.

Such graph-based views do not explain the growth rate at which certain resources are being consumed. Ganglia has some trending capabilities that display a linear interpolation of points over certain period of time (e.g. six months), then extrapolate into the future.

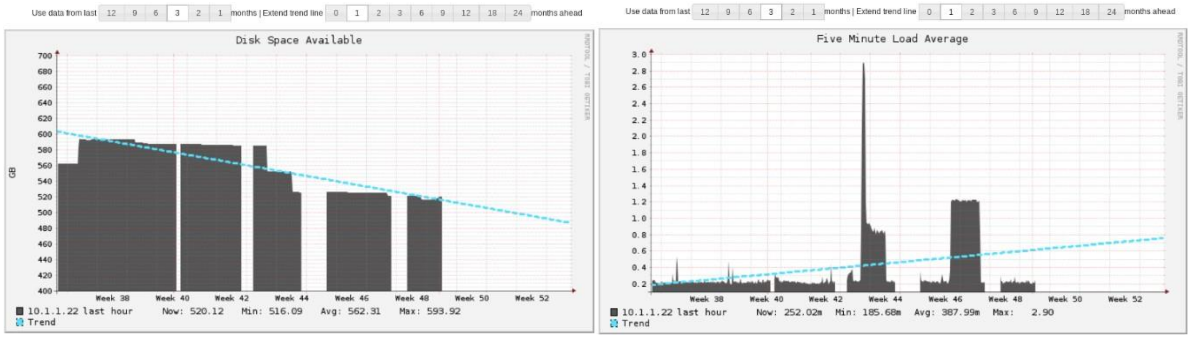

Figure 6: Interpolations and extrapolations in Ganglia: Disk space (left), CPU load (right).

Figure 6 illustrates two scenarios in Ganglia where such regressions are employed. While it could be argued that the extrapolation on the left could provide valuable insight to the operator, perhaps the extrapolation on the right is less valuable. Are there ways in which the

usefulness of extrapolations can be automatically determined? There appears to be an opportunity for a powerful performance analysis framework which can quickly perform a large range of analyses on one or more datasets and automatically extract the metrics and results and trends of most potential interest.

4 Architecture

4.1 Introduction

Several key factors were considered in the selection of the MIKELANGELO instrumentation and monitoring solution. MIKELANGELO requirements and the external state-of-the-art have already been described. Additionally INTEL had an internal monitoring solution now known as Cimmaron [32] that it had been enhancing and evolving over many years since the FP7 project IOLANES, but was not yet open-source or publically available. Whilst considering the rewriting and open-sourcing of this system, a sister-group within INTEL secured the resources to construct a production-ready, scalable, open-source telemetry framework that has since become known as snap. The INTEL MIKELANGELO team met with the snap leadership team on several occasions, fed in inputs from our experiences with Cimmaron and the state-of-the-art, and identified some additional features not on the roadmap of snap, but that would be necessary if it was to meet the needs of MIKELANGELO.

Building a community-driven open telemetry solution is a key goal of snap team and so they were keen to work with us, take our feedback on possible enhancements to the snap framework, and include any suitably mature MIKELANGELO-developed contributions in the initial open-source release of snap on December 2nd 2015.

4.2 Introducing snap

snap is an open-source telemetry framework specifically designed to help simplify the gathering and processing of rich metrics within a data center. With deeper instrumentation and analysis of such infrastructure and hosted applications, more subtle measurements of performance can be gathered, and more efficiencies can be realised.

As an extensible and open telemetry gathering system, snap aims to provide a convenient and highly scalable framework from which arbitrary metric-collecting systems, analytics frameworks, and data stores can be leveraged. Thus, full stack monitoring is achievable, allowing data from hardware, out-of-band sources such as Node Manager, DCIM and IPMI to be analysed together with data from the host operating systems, hypervisors, guest operating systems, middleware and hosted applications and services. These metrics can be transformed or filtered or be processed using any external tool locally, before being published to any destinations, be they local or remote.

4.3 The Architecture of snap

snap was designed from the ground up as an open framework that facilitates powerful gathering, processing and consumption of telemetry in the data center. Rich management

capabilities have been built into snap to ensure that it is practical to use in very large scale deployments.

snap is organised into several core components as illustrated in Figure 7.

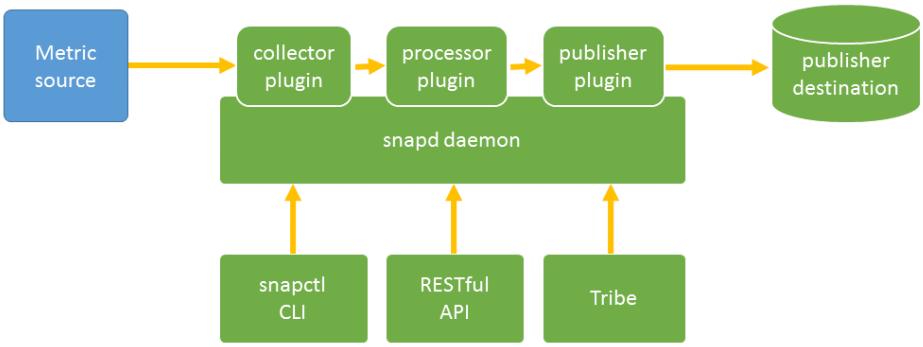

Figure 7: Core components of snap.

snapd is a daemon that runs on nodes that collect, process and publish telemetry information. snapd may collect the information from the local node (localhost), or the information may be captured from remote nodes, over the network. The latter allows out-of-band metrics to be captured from systems that are in the powered down state. The collection, processing and publishing of telemetry information is done through a flexible and dynamic plugin architecture that is described later in this chapter. This daemon also schedules the tasks that define what data is collected what data is gathered, how it is processed and published. This daemon has a RESTful API.

snapctl is a command line interface that allows snap to be managed. It allows snap metrics, plugins and specific monitoring instructions known as tasks to be queried and manipulated as required. All three of these elements can also be manipulated via a RESTful API.

snap has a flexible plugin architecture. To facilitate administration plugins can be versioned and signed. Three type of plugins are supported:

Collector plugins allow data to be fed into snap from a particular source. The metrics exposed by a collector plugin are added to a dynamically generated catalogue of all available metrics. A collector plugin may be engineered to capture data from any source including hardware, operating system, hypervisor, or application source. The data may come from out of band-systems. The data may come from data-center utility systems – or indeed sources outside the data-center.

Processor plugins allow snap telemetry data to be queried and manipulated before being transferred. A processor plugin can be used to encrypt the data, or perhaps convert the data

from one format to another. Data can be cached or filtered or transformed – e.g. into rolling averages.

Publisher plugins are used to direct telemetry data to a back-end system. Data could be published to a database, to a bus, or directly to an analytics platform. Destinations may be open source or proprietary.

Available plugins are loaded into the snap framework dynamically, and exposed functionality is available without needing to restart any service or node. The currently available plugins are listed in the plugin catalogue [33]. Once plugins are loaded into the local snap daemon, specific workflows called Tasks can be defined to detail what data is gathered where, and how it is processed and shared. Table 1 lists the plugins that are available at the time of writing.

Table 1: snap plugins as of December 2015

Plugin Type	Plugin Name	Plugin Description
Collector	CEPH	Captures data from CEPH
	Docker	Captures data from Docker
	Facter	Captures data from Facter
	Libvirt	Captures data from libvirt
	NodeManager	Captures data from Intel Node Manager
	PCM	Captures data from PCM.x
	Perfevents	Collects perfevents from Linux
	PSUtil	Captures data from psutil
	SMART	Collects SMART metrics from Intel SSDs
	OSv	Collects from OSv
Processor	MovingAverage	Calculates a moving average
Publisher	HANA	Writes to SAP HANA database

	InfluxDB	Writes to Influx database
	Kafka	Writes to Kafka messaging system
	MySQL	Writes to MySQL database
	OpenTSDB	Writes to OpenTSDB database
	PostgreSQL	Writes to PostgreSQL database
	RabbitMQ	Writes to RabbitMQ
	Riemann	Writes to Riemann monitoring system

Nodes that host snap agents can be managed and manipulated in groups using snap tribe. For scalability, management instructions are automatically distributed node-to-node within these groups known as agreements. Nodes can be auto-discovered, and automatically load the plugins and tasks from other nodes in their agreement. Management instructions can be issued via the snap command line interface, or via a RESTful HTTP interface.

4.4 Using snap

A Task defines what metrics to capture, and how those metrics are to be processed and published. The metrics to be gathered are selected from the dynamic catalogue of metrics. Any configuration information required to capture that metric are also supplied in the task definition. Tasks can be defined in either JSON or YAML format.

```
collect:
    metrics:
      /intel/mock/foo: {}
      /intel/mock/bar: {}
      /intel/mock/*/baz: {}
    config:
      /intel/mock:
        user: "root"
        password: "secret"
    process:
      -
        plugin_name: "passthru"
```

```
        publish:
          -
            plugin_name: "file"
            config:
              file: "/tmp/published"
```

The captured data may be fed into any number of processor or publisher plugins. A processor plugin may feed into any number of processor or publisher plugins. Thus arbitrarily deep workflows can be created such as illustrated in Figure 8.

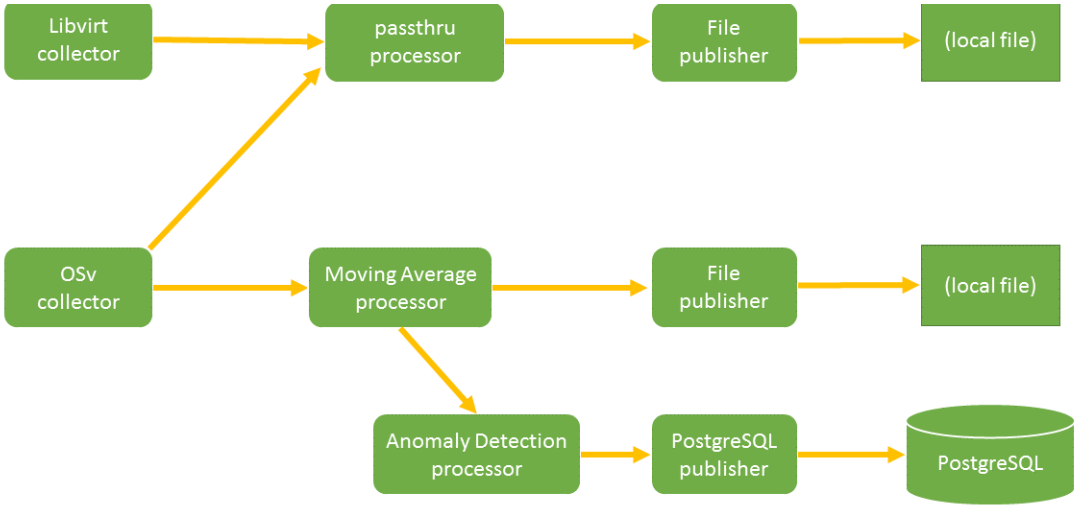

Figure 8: A task can define an arbitrarily complex chain of plugins.

A typical interaction with snap could involve the following steps:

- Loading the relevant plugins
- Listing the plugins
- Listing the available metrics
- Defining a new task if necessary
- Starting a task
- Listing the running tasks

Detailed examples of these interactions are described in the online documentation for snap [2].

4.5 Extending snap

The snap project is fully open-sourced and is released under the Apache License Version 2 for maximum flexibility: plugin developers are not forced to release their plugins under an open-

source license. However, open-source contributions to snap from the community are actively encouraged and supported through the GitHub Pull Request mechanism.

Although snap is written in GO, plugins can be developed in any language once they can communicate with snap through HTTP or TCP protocols. HTTP JSON RPC is recommended, unless developing in GoLang in which case the native client can be used directly.

Plugins can include whatever packages are necessary to communicate with whatever telemetry source, processing engine or storage back-end is necessary. Plugins are versioned, and run as separated processes so there is no risk of colliding dependencies.

Plugins can be signed, and communications between plugins are encrypted by default.

A detailed explanation of how to develop additional plugins is available online [34].

5 Design and Implementation

5.1 Introduction

Following consultation with the MIKELANGELO consortium and the snap team it was agreed that progress could be made most efficiently if the INTEL MIKEANGELO team could focus their efforts on delivering several key plugins for snap. For the first release of snap the priority plugins for the INTEL MIKELANGELO team were a collector of data from libvirt-based hypervisors, a collector of data from the OSv guest operating system, and a publisher plugin that could route data into a PostgreSQL database. All three plugins were successfully implemented in time for the initial open-source release on December 2nd 2015. Work also began on a processor plugin that automatically transmits rich telemetry data only when anomalies are detected. A summary of the design and implementation of these four plugins follows.

5.2 INTEL-MIKELANGELO Plugins

5.2.1 Collector Plugins

5.2.1.1 Libvirt Collector Plugin

MIKELANGELO uses the snap libvirt collector plugin to collect metrics from the libvirt API. This API is used by many linux hypervisors, including MICHELANGELO's sKVM, to support virtualisation and host virtual machines.

The snap libvirt collector plugin can be configured to work as an internal or external collector it can capture data from a libvirt locally, or on a remote machine. It can gather specific metric types from domain subsystems including CPU, disk, memory and network. For the CPU it provides the accumulated data for cputime – the sum of all cores. The plugin also has the ability to send the cputime for each processor core separately. For other subsystems, it is possible to monitor the following metrics:

For each disk device:

- wrreq - Write Requests
- rdreq - Read Requests
- wrbytes - Write Bytes
- rdbytes - Read Bytes

For each network device:

- rxbytes - Bytes received
- rxpackets - Packets received
- rxerrs - Errors on receive

- rxdrop - Drops on receive
- txbytes - Bytes transmitted
- txpackets - Packets transmitted
- txerrs - Errors on transmit
- txdrop - Drops on transmit

For memory:

- mem - Amount of memory specified on domain creation
- swap_in - Amount of memory swapped in
- swap_out - Amount of memory swapped out
- major_fault - Number of major faults
- minor_fault - Number of minor faults
- free - Total amount of free memory
- max - Total amount of memory

All metrics except cputime are represented as a 64 bit integer. Cputime is a 16 bit integer.

Metrics can be enumerated explicitly via a concrete namespace, or a wildcard (*) can be used. The namespaces are keys to another nested object which may contain a specific version of a plugin. Illustrating both of these in an example: to force all used memory metrics on node3 to be gathered by version 4 of the libvirt plugin, the following syntax could be used in the task configuration file:

```
/libvirt/node3/*/mem/used:
  version: 4
```

The location of the plugin within the MIKELANGELO stack is highlighted in yellow in the following figure.

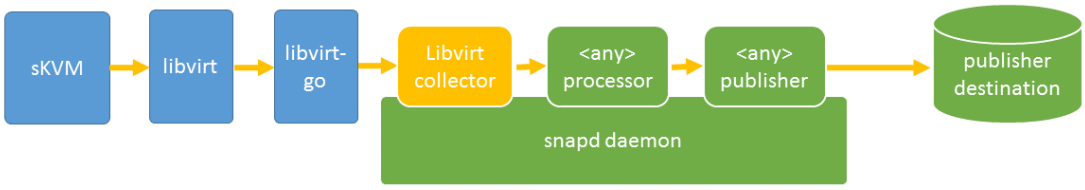

Figure 9: libvirt collector plugin within the MIKELANGELO stack.

5.2.2 OSv Collector Plugin

Mikelangelo uses a snap OSv collector plugin to collect metrics from the OSv operating system. OSv is the key cloud guest operating system targeted for further optimisation by the MIKELANGELO project.

The snap OSv collector plugin has the ability to collect metrics from memory and cpu time as well as traces exposed in the swagger http interface. All traces are divided into the specific groups listed as follows:

- virtio – virtual io driver counters
- net – network counters
- tcp – tcp specific counters
- memory – memory specific counters
- callout – callout specific counters
- waitqueue – waitqueue specific metric
- async – async operations specific metric
- vfs – virtual file system specific metrics

In total the OSv collector plugin supports the capture of approximately 260 different metrics as described in Chapter 6 of Deliverable D2.16, The First OSv Guest operating System MIKELANGELO Architecture [3].

All metrics are represented as a 64 bit integer. Metrics can be enumerated explicitly via a concrete namespace, or a wildcard (*) can be used. The namespaces are keys to another nested object which may contain a specific version of a plugin. Illustrating both of these in an example: to force all trace-wait metrics on node3 to be gathered by version 5 of the OSv plugin, the following syntax could be used in the task configuration file:

```
/osv/node3/trace/wait/*

 version: 5
```

The location of the plugin within the MIKELANGELO stack is highlighted in yellow in the following figure.

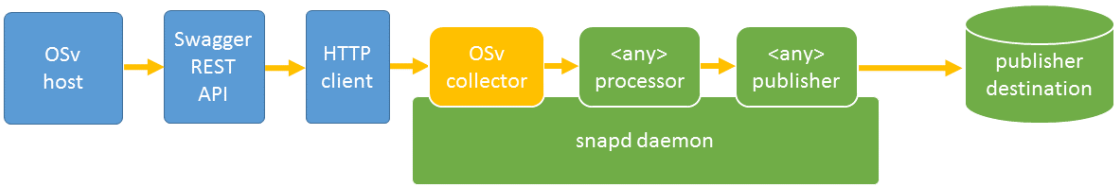

Figure 10: OSv collector plugin within the MIKELANGELO stack.

The snap OSv collector plugin includes several key features including:

Trace module – an extendable API for collecting traces from OSv, with support for multithreading and concurrent calls.

Wildcard support – an easy way to filter metrics of a specific type. It can also filter implementation by groups such as callout or async. When a group is specified, all metrics that are included in the group will be collected.

To retrieve the plugin source code, the following commands can be run:

```
git clone https://github.com/intelsdi-x/snap-plugin-collector-osv
cd snap-plugin-collector-osv
export SNAP_OSV_PLUGIN_DIR=`pwd`
```

The project contains a README.md file which describes how to build the project from source. Most applications have prerequisites. Components such as the golang runtime and godep tool must be available on a target computer in order for the application to be built. Documentation on how to install golang is available on the golang project website [35] . To install the godep tool, run:

```
$ go get github.com/tools/godep
```

and then

```
$ make
```

The following instructions describe how to run the OSv collector with the snap passthru processor, and write the data to a file.

In one terminal window, start the snap daemon:

```
$ snapd -l 1
```

In another terminal window, load the snap osv plugin:

```
$ snapctl plugin load $SNAP_OSV_PLUGIN_DIR/build/rootfs/snap-plugin-collector-osv
```

To view the available metrics for your system run:

```
$ snapctl metric list
```

Define a task in a JSON file:

```
{
        "version": 1,
        "schedule": {
        "type": "simple",
        "interval": "1s"
        },
        "workflow": {
        "collect": {
                "metrics": {
                  "/osv/trace/wait/waitqueue_wake_one": {},
                "/osv/trace/callout/callout_reset": {},
                "/osv/cpu/cputime": {},
                "/osv/memory/free": {}
                },
                "config": {
                "/osv": {
                        "swag_ip": "192.168.122.89",
                        "swag_port": 8000
                }
                },
                "process": [
                {
                        "plugin_name": "passthru",
                        "process": null,
                        "publish": [
                        {
                           "plugin_name": "file",
                               "config": {
                                  "file": "/tmp/published_osv"
                               }
                        }
                        ],
                        "config": null
                }
                ],
                "publish": null
        }
        }
}
```

Load the passthru plugin for processing:

```
$ snapctl plugin load $SNAP_DIR/build/rootfs/plugin/snap-processor-passthru
Plugin loaded
```

```
Name: passthru
Version: 1
Type: processor
Signed: false
Loaded Time: Fri, 20 Nov 2015 11:44:03 PST
```

Load the file plugin for publishing:

```
$ snapctl plugin load $SNAP_DIR/build/rootfs/plugin/snap-publisher-file
Plugin loaded
Name: file
Version: 3
Type: publisher
Signed: false
Loaded Time: Fri, 20 Nov 2015 11:41:39 PST
```

Change the IP address and port of the OSv host in the task manifest:

```
vim $SNAP_OSV_PLUGIN_DIR/example/osv-file-example.json
```

Create a task:

```
$ snapctl task create -t $SNAP_OSV_PLUGIN_DIR/example/osv-file-example.json
Using the task manifest to create a task
Task created
ID: 02dd7ff4-8106-47e9-8b86-70067cd0a850
Name: Task-02dd7ff4-8106-47e9-8b86-70067cd0a850
State: Running
```

Finally, review the file output. It will include data similar ti the following:

```
NAMESPACE  DATA  TIMESTAMP   SOURCE
/osv/cpu/cputime      176521305      2015-11-25 15:36:04.225846442 +0000 UTC
192.168.122.89
/osv/memory/free      2023403520     2015-11-25 15:36:04.226192641 +0000 UTC
192.168.122.89
/osv/trace/callout/callout_reset 206217           2015-11-25 15:36:04.226534352
+0000 UTC            192.168.122.89
/osv/trace/wait/waitqueue_wake_one   1319942             2015-11-25
15:36:04.226810341 +0000 UTC               192.168.122.89
```

5.2.3 Processor Plugins

5.2.3.1 Anomaly Detection Processor Plugin

Mikelangelo will use the Tukey anomaly detection plugin as a processor plugin. This plugin is currently in development. It was designed and created as part of the MIKELANGELO project. The plugin has the ability to process all metrics transmitted from snap collectors and to only send high granularity data to the publisher when an anomaly is detected. When the source is idle or the measurement is stable, data is sent at regular intervals, by default every 10 seconds.

The Tukey method was selected because it constructs a lower threshold and an upper threshold, where the thresholds are used to flag data as anomalous. The Tukey method does not make any distributional assumptions about the data.

The Tukey method is a simple but effective procedure for identifying anomalies. For the purposes of illustration, let x1, x2, . . . xn be a series of observations, such as the processor utilization of a server. The data is arranged in an ascending order from the smallest to the largest observation. The ordered data is broken into four quarters, the boundary of each quarter defined by Q1, Q2, and Q3, referred to herein as the "first quartile," "second quartile," and "third quartile," respectively. The difference |Q3−Q1| is referred to as the "inter-quartile range."

The location of the plugin within the MIKELANGELO stack is highlighted in yellow in the following figure.

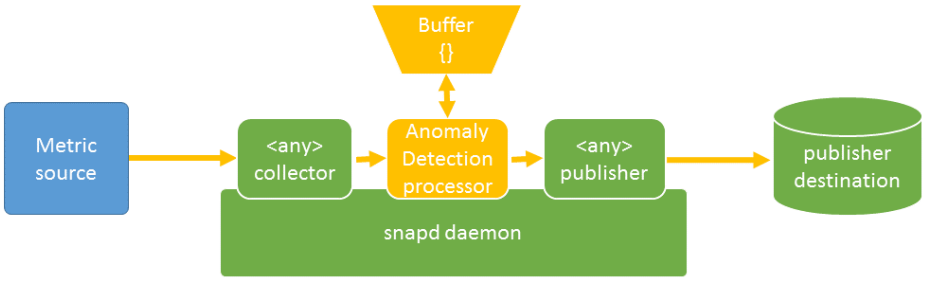

Figure 11: Anomaly Detection processor plugin within the MIKELANGELO stack.

At the time of writing a local proof-of-concept of the implementation of this plugin has been successfully developed. This proof-of-concept will be matured into a fully supported open-source plugin in the first months of 2016.

5.2.4 Publisher Plugins

5.2.4.1 PostgreSQL Publisher Plugin

MIKELANGELO uses the PostgreSQL publisher plugin to store collected, processed metrics into a PostgreSQL database. PostgreSQL is a powerful scalable object-relational database management system. It contains many advanced features, is very fast and is standards compliant. These features make PostgreSQL easy to deploy and a very useful, scalable backend for metric data.

The snap publisher plugin for PostgreSQL developed by MIKELANGELO uses the official PostgreSQL library for Go. The plugin can handle many types of values like string maps,

integer maps, boolean data types, integers, floats and strings. All maps are converted to strings.

The plugin can work in two different modes: transaction mode and single mode. Transaction mode mode stores data in one bulk insert including all metrics reported by the processor. Single mode uses a separate insert for each metric.

5.3 Other Plugins

By choosing snap, the MIKELANGELO project is able to leverage all of the plugins that are developed by the broader snap community. The open-source plugins in Table 2 below were included in the initial open-source release of snap and are of particular relevance and use to MIKELANGELO. They illustrate some of the richness of metrics that will be available during the evaluation and analysis phases of the MIKELANGELO project.

Table 2: snap plugins in the initial open-source release of particular relevance to MIKELANGELO

Plugin Type	Plugin Name	Plugin Description
Collector	NodeManager	Collect detailed platform airflow, CPU, power and temperature metrics from the system motherboard
	SMART	Collect detailed disk performance data including reads, writes, sector reallocation, power, remaining space, temperature, and wearout metrics
	PSUtil	Collect detailed process and system metrics covering CPU, load, network and memory
	PCM	Collect detailed CPU, cache, energy, temperature and other metrics from the Intel Performance Counter monitor.
	Libvirt	Collect detailed disk, memory, cpu and network metrics from libvirt-based hypervisors including sKVM.
	Docker	Collect detailed docker metrics covering cpu, memory, bulk IO, and huge page table statistics,
	OSv	Collect over 260 metrics from an OSv instance including detailed cpu, memory and trace data.
Processor	MovingAverage	Calculate a moving average for collected data.
Publisher	OpenTSDB	Publish to an OpenTSDB back end, supporting rich

		time-series visual querying.
	InfluxDB	Publish to an InfluxDB back end, supporting grafana-built dashboards and visualisation.
	PostgreSQL	Publish to a PostgreSQL relational database.
	MySQL	Publish to a MySQL relational database.
	RabbitMQ	Publish to a RabbitMQ messaging queue.

6 Observations & Plans

6.1 Observations

Being able to build on and contribute to the snap open source telemetry framework allows MIKELANGELO resources to focus on implementing just the precise functionality that it needs, rather than attempt to construct and maintain an independent flexible framework.

At the time of writing the following scenarios are all supported.

Data can be collected from

- motherboards
- cpus
- memory
- disks
- operating systems
- hypervisor
- guest operating system

Specific collection tasks can be arbitrarily defined, with resolution, duration, and precise metrics captured easily and dynamically configurable.

Any of this data can have moving averages calculated locally, and soon anomaly detection will allow resolution of data to be automatically adjusted to reduce the burden on network and back end systems, without affecting richness of the data.

This data can be published to a variety of useful tools including

- MySQL
- PostgreSQL
- InfluxDB
- OpenTSDB

Existing tools can be leveraged to construct dashboards and other visualisations of this data.

Plugins are versioned and easily deployable and upgradable. The entire system can be managed via the command line or a RESTful API. Tribe commands are supported, propagating instructions peer-to-peer inside relevant groups, delivering practical management at scale.

Thus, a potential deployment of snap in MIKELANGELO is illustrated in Figure 12.

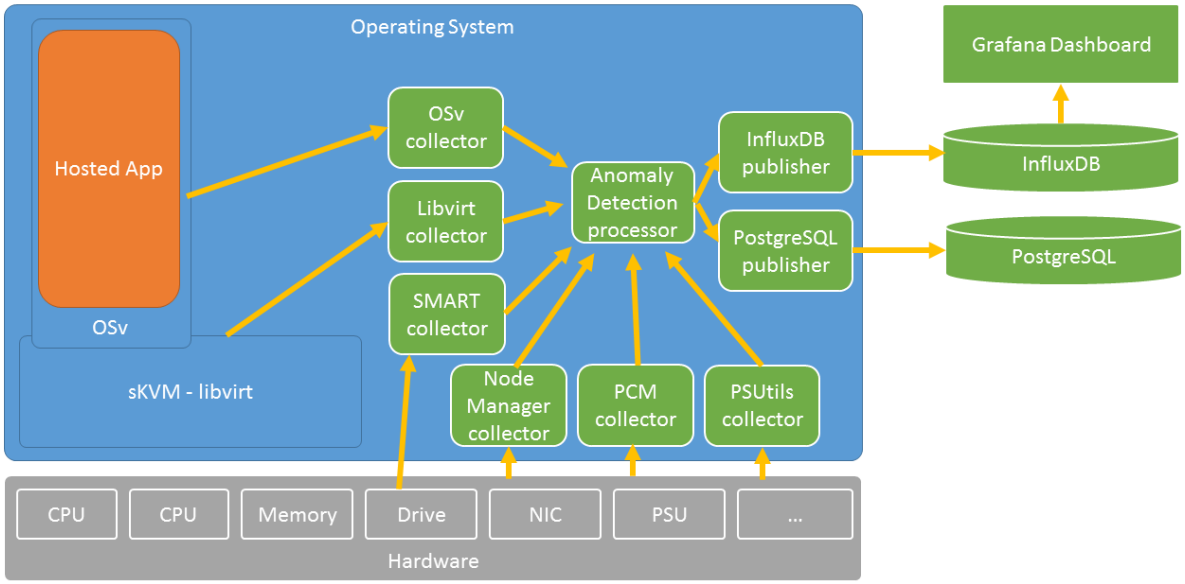

Figure 12: Example arrangement of snap plugins for a MIKELANGELO deployment.

Figure 13 demonstrates a dashboard created using Grafana exposing data captured from an OSv guest instance and published to an InfluxDB database.

Figure 13: Example Grafana dashboard displaying metrics captured from OSv.

All plugins can be signed to verify their authenticity, and telemetry data can be encrypted.

6.2 Plans

Regarding next steps, INTEL will continue to enhance and extend the plugins already developed. Additional short-term plans (early 2016) for the instrumentation and monitoring system include:

- Completing development of the anomaly detection plugin to allow intelligent adjustment of telemetry resolution
- Developing a nova plugin to allow OpenStack metadata to be captured and made available for analysis
- Developing an OpenVSwitch plugin to allow detailed virtual network statistics to be captured

In the medium term, throughout 2016, as the use cases start to deploy on the various testbeds, custom plugins will be developed as appropriate to enable application specific metrics to be gathered and exposed.

For the longer term, across 2016 and 2017, INTEL will explore if there is benefit in developing a powerful performance analysis toolkit integrated with the instrumentation and monitoring system. Such a toolkit could automate the comparison of large volumes of data captured across multiple runs of an experiment, allowing key changes in performance and correlations to be discovered, and optimisations to be identified.

7 Key Takeaways

The key takeaways of this deliverable are:

- A rich instrumentation and monitoring framework - snap - is now available to support the varied, full-stack evaluation needs of the MIKELANGELO project.
- snap is open-source, highly scalable, completely extendable and has significant corporate support.
- The open-source community are being actively encouraged to leverage and contribute to snap. MIKELANGELO will benefit from community contributions going forward, and has already started contributing to this community. This engagement will help raise the awareness of MIKELANGELO in a very relevant community: those interested in optimising the performance of their clouds.
- MIKELANGELO has contributed important functionality to snap: libvirt and OSv collector plugins and a PostgreSQL publisher plugin are already released.
- Other useful MIKELANGELO-developed plugins are in development or in planning.

8 Concluding Remarks

In the first 6 months of this task an initial instrumentation and monitoring framework has been delivered with the extensibility and scalability to support MIKELANGELO's foreseeable telemetry needs. The solution builds upon and contributes to the open-source snap telemetry framework, exposing MIKELANGELO efforts and results to that community and allowing MIKELANGELO to benefit from contributions that that community may make,

INTEL's MIKELANGELO resources have been able to focus on MIKELANGELO-specific needs, successfully implementing plugins such as the OSv Collector plugin, getting them robust enough to be incorporated into the initial snap open-source release.

INTEL look forward to further enhancing this initial instrumentation and monitoring framework over the coming months and years as the needs of the MIKELANGELO use-cases become more explicit. INTEL will also pursue additional opportunities for advancing the state of the art in areas such as performance analysis, open-sourcing our results were possible and appropriate.

9 References and Applicable Documents

[1] The MIKELANGELO project, http://www.mikelangelo-project.eu/
[2] The snap open-source telemetry framework, https://github.com/intelsdi-x/snap
[3] MIKELANGELO Public Deliverables, https://www.mikelangelo-project.eu/deliverables/
[4] The OSv guest operating system for the cloud, http://osv.io/
[5] The GNU General Public License 3, http://www.gnu.org/licenses/gpl-3.0.en.html
[6] The Apache License Version 2.0, http://www.apache.org/licenses/LICENSE-2.0
[7] The IOLANES project, http://www.iolanes.eu/
[8] Wikipedia comparison of network monitoring systems, https://en.wikipedia.org/wiki/Comparison_of_network_monitoring_systems
[9] List of monitoring and metrics tools by Monitoring Sucks, https://github.com/monitoringsucks/tool-repos
[10] Montes, J., Sánchez, A., Memishi, B., Pérez, M. S., & Antoniu, G. (2013). GMonE: A complete approach to cloud monitoring. Future Generation Computer Systems, 29(8), 2026-2040. doi: http://dx.doi.org/10.1016/j.future.2013.02.011
[11] Povedano-Molina, J., Lopez-Vega, J. M., Lopez-Soler, J. M., Corradi, A., & Foschini, L. (2013). DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant Clouds. Future Generation Computer Systems, 29(8), 2041-2056. doi: http://dx.doi.org/10.1016/j.future.2013.04.022
[12] Frank Carden, R. P. J., Robert Henry. (2002). Telemetry Systems Engineering: Artech House.
[13] OpenNMS Opensource Network Management System, http://www.opennms.org/
[14] Zabbix Enterprise Class Network Monitoring Platform, http://www.zabbix.com/
[15] CA Spectrum - Root Cause Analysis Software & Tools, http://www.ca.com/us/root-cause-analysis.aspx
[16] Ganglia Monitoring System, http://ganglia.sourceforge.net/
[17] Prometheus Monitoring System, http://prometheus.io/
[18] Sensu Monitoring System, https://sensuapp.org/
[19] Montes, J., Sánchez, A., Memishi, B., Pérez, M. S., & Antoniu, G. (2013). GMonE: A complete approach to cloud monitoring. Future Generation Computer Systems, 29(8), 2026-2040. doi: http://dx.doi.org/10.1016/j.future.2013.02.011
[20] Oracle database, https://www.oracle.com/database/index.html
[21] Postgresql database, http://www.postgresql.org/
[22] Hadoop, https://hadoop.apache.org/
[23] OpenTSDB, http://opentsdb.net/
[24] Nagios Monitoring System, https://www.nagios.org/
[25] Graphite Graping System, http://graphite.wikidot.com/
[26] D3.js Visualisation framework, http://d3js.org/

[27] Grafana Dashboard and graphing System, http://grafana.org/

[28] IBM Tivoli - IT operations analytics, http://www-03.ibm.com/software/products/en/category/it-operations-analytics

[29] HP SiteScope - Agentless Application Monitoring Software Solutions, http://www8.hp.com/us/en/software-solutions/sitescope-application-monitoring/index.html

[30] VMWare vSphere - Virtual Infrastructure Monitoring & Analytics, https://www.vmware.com/products/vsphere-operations-management/features/vsphere-monitoring

[31] Gulati, A., Shanmuganathan, G., Ahmad, I., Waldspurger, C., & Uysal, M. (2011). Pesto: online storage performance management in virtualized datacenters. Proceedings of the 2nd ACM Symposium on Cloud Computing, Cascais, Portugal.

[32] Metsch, T., Ibidunmoye. O., Bayon-Molino, V., Butler, J.,Hernández-Rodriguez, F., Elmroth, E. Apex Lake: A Framework for Enabling Smart Orchestration, Article No.: 1, Proceedings of the Industrial Track of the 16th International Middleware Conference doi: http://dx.doi.org/10.1145/2830013.2830016

[33] snap plugin catalog, https://github.com/intelsdi-x/snap/blob/master/docs/PLUGIN_CATALOG.md

[34] snap plugin authoring guidlines, https://github.com/intelsdi-x/snap/blob/master/docs/PLUGIN_AUTHORING.md

[35] The Go programming language, https://golang.org/dl/