WHITE PAPER

Intel Corporation

Closed Loop Automation – Telemetry Aware Scheduler for Service Healing and Platform Resilience

Authors

John Browne

Emma Collins

Krzysztof Kepka

Sunku Ranganath

Jabir Kanhira Kadavathu

Swati Sehgal

Killian Muldoon

Michal Kobylinski

1 Introduction

Closed loop automation is the process of continually monitoring real-time network conditions, workload requirements, and resource capabilities and availability to determine the optimum workload placement for faster and more efficient delivery of services. Service Providers are adopting Closed Loop Automation as a strategic priority to address their business objectives [1]. One such objective is to ensure customer Service Level Agreements (SLAs) are met to provide an optimum Quality of Experience (QoE), which is delivered according to operator defined policies. Network Services rely on the infrastructure they run on and so this is a crucial place to start with when trying to ensure an optimal QoE.

This document provides an overview of a platform resiliency prototype that showcases the integration of key Closed Loop Automation components, leveraging a mix of solutions provided by Intel and the open source community.

Traditional Telecom Services Assurance is an Operations Support System (OSS) function carried out offline [1] without automated processes and methods for self-optimization of the network. This paper outlines 'near real-time' automation at a granular platform level. This allows an orchestration system to detect and respond to an issue before a customer is impacted, including integration with Management and Network Orchestration (MANO), using platform telemetry, and analytics. Closed loop solutions integrate components that include platform, telemetry, analytics processing, MANO and polices to create automated processes.

The Intel platform has a wealth of features and resources that provide a rich set of data and control points that can be used for configuration, reporting, monitoring and managing workloads in a network infrastructure. This data can be used as part of a Closed Loop Solution to provide granular insights into the behavior of the system. When these metrics provide actionable alerts, the configuration controls can be used to provide platform resiliency support and ensure minimal unplanned operational downtime.

By collecting Intel® Architecture specific feature telemetry, this prototype will provide host insights to a new Kubernetes* (K8s*) scheduling extension, called Telemetry Aware Scheduling (TAS), to trigger corrective healing actions on a workload and influence workload placement decisions, depending on the reliability of the underlying infrastructure. By gaining these insights into the health of the platform, the services they are running on, and having the control to be able to remediate against faults and errors in an automated way, this prototype provides the methodology to reduce risk of unplanned outages and enable scheduled downtime for maintenance, therefore maximizing service availability.

Along with the video showing this use case in action included in the release, this document is part of the Network Transformation Experience Kit, which is available at: https://networkbuilders.intel.com/network-technologies/network-transformation-exp-kits

Table of Contents

1 Introduction 1

2 Document Overview 3
2.1 Intended Audience 3
2.2 Terminology 3
2.3 Reference Document 3

3 Components of the Intel® Closed Loop Automation Solution 4
3.1 Intel® Architecture Platform Telemetry 4
3.2 Telemetry Analysis 5
3.2.1 Host Health Indicator Calculation Logic 5
3.3 Automated Action – Telemetry Aware Scheduling 6
3.4 Error Injection 8

4 Closed Loop Automation – Platform Resiliency Use Case Description 8
4.1 Method of Operation 9
4.2 Reliability Aware Placement Scenarios 9

5 Conclusion 10

Figures

Figure 1. ETSI NFV Architecture + Analytics Component (Including view of scope of this document) 4
Figure 2. Streaming analytics with Kafka and KSQL 5
Figure 3. Host Health Indicator Sample Threshold Decision Matrix 6
Figure 4. Topology of a Kubernetes system integrated with Telemetry Aware Scheduling 7
Figure 5. High level Prototype Architecture 8

Tables

Table 1. Terminology 3
Table 2. Reference Document 3

2 Document Overview

This document describes the components of a closed loop solution for automating a "platform resiliency" use case. Details are provided on the wealth of Intel® Architecture specific platform features that can play a crucial role in many automated use cases. Combined with solutions for telemetry analysis and orchestrated techniques for intelligent workload placement, the following sections will outline the potential building blocks that can be used to implement their own closed loop solution on an Intel® Architecture platform.

2.1 Intended Audience

This white paper is intended for Service Providers, Telecom Equipment Manufacturers (TEMs), Network Monitoring and Management solution providers or those planning to deploy closed loop automation solutions running on Intel® Xeon® Processors.

2.2 Terminology

Table 1. Terminology

ABBREVIATION	DESCRIPTION
BMC	Baseboard Management Controller
CLA	Closed Loop Automation
CSP	Communication Service Provider
IPMI	Intelligent Platform Management Interface
KSQL	Kafka* Structured Query Language
KPI	Key Performance Indicators
LLC	Last Level Cache
MANO	Management and Network Orchestration
MCA	Machine Check Architecture
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Functions Virtualization Orchestrator
OSS	Operating Support System
QoE	Quality of Experience
OPNFV*	Open Platform for Network Functions Virtualization*
PMU	Performance Monitoring Unit
RAS	Reliability, Availability and Serviceability
RDT	Intel® Resource Director Technology
SLA	Service Level Agreements
TAS	Telemetry Aware Scheduling
TEM	Telecoms Equipment Manufacturer
VNF	Virtual Network Function
VNFM	Virtual Network Function Manager

2.3 Reference Document

Table 2. Reference Document

Ref Num.	REFERENCE	SOURCE
1.	NFV Service Assurance – In Need of Big Data, Small Data or Both?	https://www.lightreading.com/analytics/big-data/nfv-service-assurance---in-need-of-big-data-small-data-or-both/a/d-id/739614

3 Components of the Intel® Closed Loop Automation Solution

This section includes the ETSI NFV Architecture diagram and the scope of this prototype relative to this architecture superimposed over it (refer to Figure 1). There is a Virtual Network Function (VNF) running in a Network Function Virtualization Infrastructure (NFVI) and interacting with the VIM layers. The blue layer details the configuration specific to this document and build. An analytics function has been added also to show its inclusion as part of a closed loop solution.

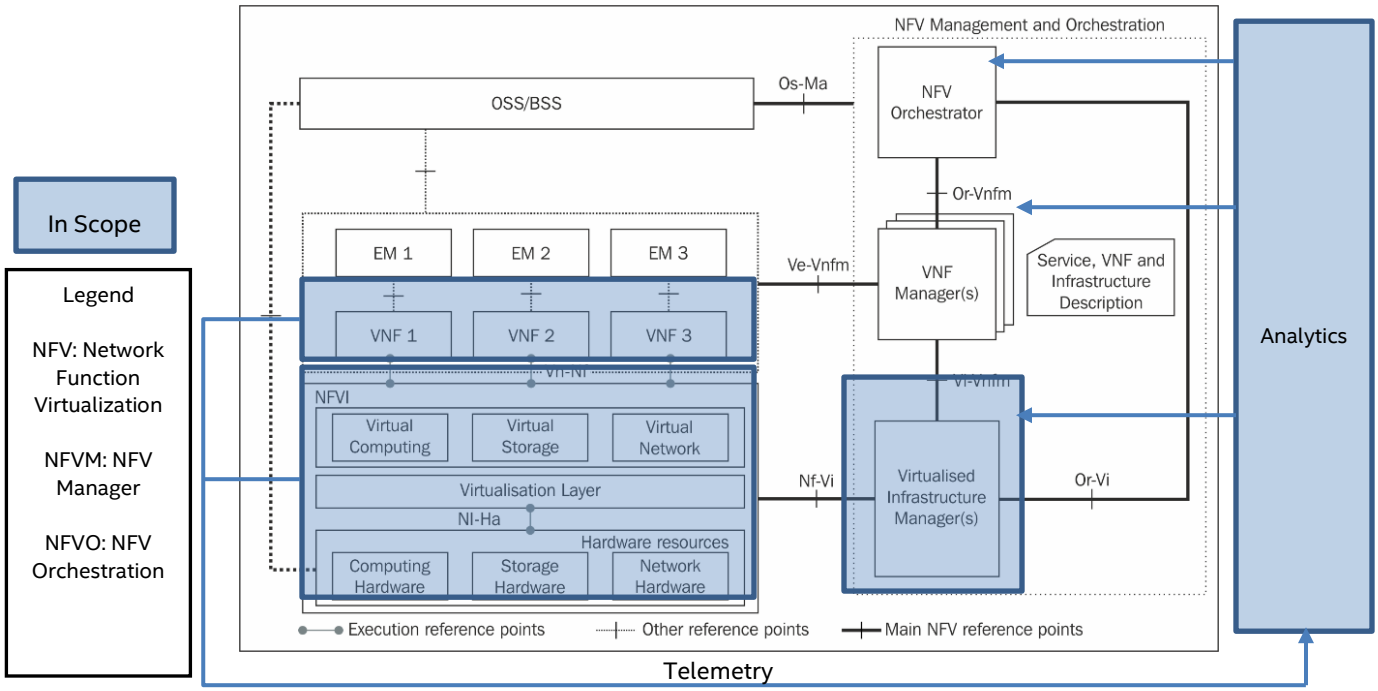

Figure 1. ETSI NFV Architecture + Analytics Component (Including view of scope of this document)

3.1 Intel® Architecture Platform Telemetry

Intel platform features have been instrumented to provide fine grained insights into the performance and health of a platform, as well as other KPI's. This prototype uses collectd* plugins that have been specifically written to retrieve stats and events from these particular Intel Platform Features. collectd, being the data collection mechanism of choice, is an open source daemon that has a pluggable architecture. It uses the concept of south bound plugins, which will read metrics from a platform feature, and north bound plugins, which will make this information available in various formats, so that it can be consumed as necessary (such as through monitoring or analytics solutions). The frequency of metrics gathering from the platform and/or processes is fully configurable, ranging from seconds to minutes, depending on the granularity required.

Metrics from the following features, currently available on most Intel platforms, are the inputs into the Streaming Analytics component that can be used to make a decision as to platform health and provide a "Host Health Indicator". See Section 3.4 for more details on the "Host Health Indicator calculation".

Intel server platforms provide a rich and growing set of data and control points that fall under the umbrella term of Intel's Infrastructure Management Technologies. These features can be used for configuration, reporting, monitoring and managing workloads and data:

- Intel® Resource Director Technology provides the ability to view and control shared resources like the Last Level Cache (LLC) and Memory Bandwidth being used by an application, virtual machine or container running on the platform. For more information, refer to Intel® Resource Director Technology (Intel® RDT)'s resource on Unlocking System Performance In Dynamic Environments.
- The Performance Monitoring Unit provides performance counters through hardware registers that count hardware events such as instructions executed, cache-misses suffered, or branches mispredicted. This information can be used for profiling applications that run on Intel CPU's.
- Intel® Run Sure Technologies, including Reliability, Availability and Serviceability (RAS) features, provide advanced reliability and system resiliency for workloads. They provide the capability to auto-detect and correct transient hardware errors, and advanced CPU and memory disk monitoring and recovery features (which includes the detection and reporting of potentially service impacting faults).
- Intelligent Platform Management Interface (IPMI) collects sensor data from the Baseboard Management Controller (BMC) of a platform which includes data from the internal physical variables such as temperature, humidity, power-supply voltage and fan speeds.

All of these features have an associated collectd* plugin that can be found as part of collectd or Barometer, the Open Platform for Network Functions Virtualization* (OPNFV*) telemetry project.

FEATURE	COLLECTD INFORMATION	BAROMETER INFORMATION
Intel® RDT	https://collectd.org/wiki/index.php/Plugin:IntelRDT	https://wiki.opnfv.org/display/fastpath/Intel_RDT
PMU	https://collectd.org/wiki/index.php/Plugin:Intel_PMU	https://wiki.opnfv.org/display/fastpath/PMU
RAS	https://collectd.org/wiki/index.php/Plugin:mcelog	https://wiki.opnfv.org/display/fastpath/Memory+RAS
IPMI	https://collectd.org/wiki/index.php/Plugin:IPMI	https://wiki.opnfv.org/display/fastpath/IPMI

3.2 Telemetry Analysis

A truly autonomous network and infrastructure require a level of intelligence and synthesis with capability to enforce the decisions taken. This requires the right level of telemetry to be consumed at node level for real time closed loops, Virtual Network Function Manager (VNFM) level for near-real time closed loops and at the Network Functions Virtualization Orchestrator (NFVO) level for overall processing. The platform metrics and application metrics need to be correlated and tailored for specific sets of use cases based on the workload. Analytics play an important role in data visualization, modeling, trending, capacity optimization that are not often easy to correlate without an intelligent agent. The choice of the algorithm or the extent of intelligence would differ heavily based on the use case being chosen to be automated in a closed loop.

Apache Kafka* open source framework provides a distributed streaming platform that helps build near real-time data pipelines to perform stream processing enabling anomaly detection on streaming data before being consumed by a time series database. Kafka's streams library and Kafka Structured Query Language (KSQL*) provides the necessary constructs to build necessary queries on top of streaming data to identify and alert the state of the deployment. This prototype uses KSQL queries built on collectd Kafka topic from each of the nodes, run the queries to calculate 'Host Health Indicator', as detailed in Section 3.2.1, and indicates the red or yellow state as a new Kafka topic to be fed in to Prometheus*.

A custom Kafka to Prometheus Bridge has been implemented to export the data from Kafka topic format to a format that Prometheus can understand. Kubernetes TAS would then take corrective action based on the host health indicator, as detailed in Figure 2.

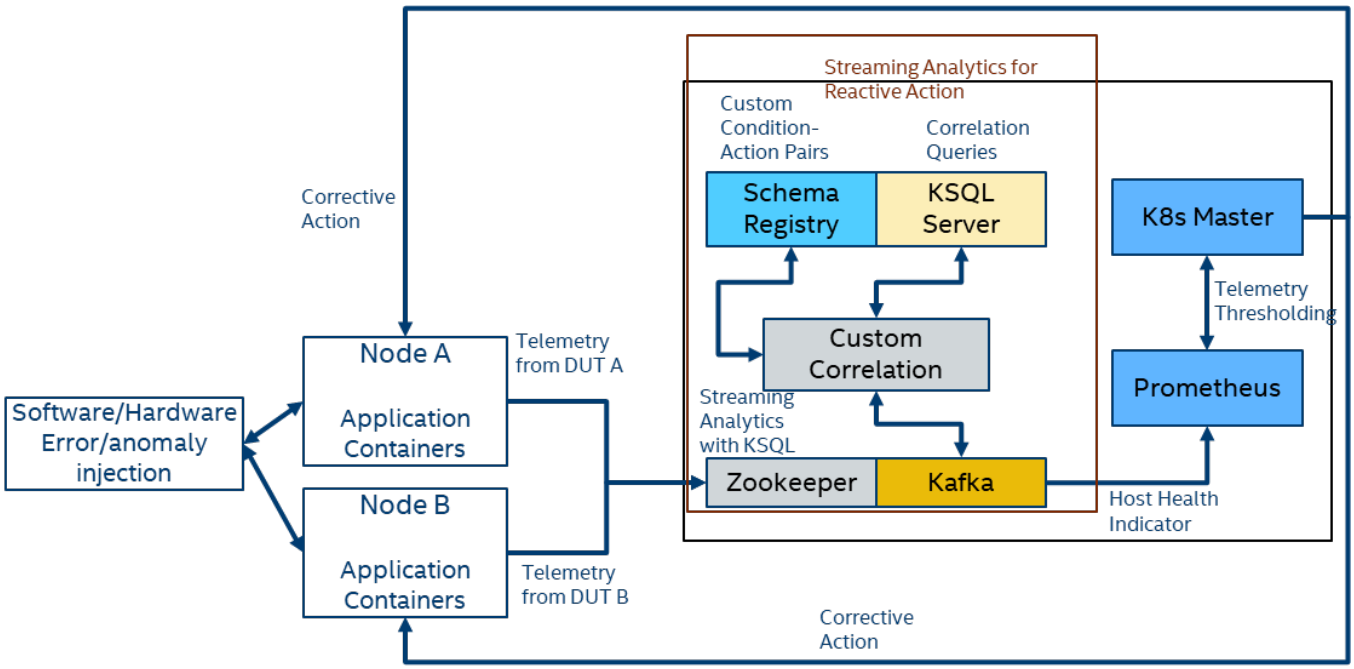

Figure 2. Streaming analytics with Kafka and KSQL

3.2.1 Host Health Indicator Calculation Logic

Figure 3 gives an overview of the approach the telemetry analysis framework took to calculating the individual host health indicators based on the metrics and events coming from the platform. Each measured metric is compared against expected nominal threshold ranges for that metric and an alert is generated when the value is outside the expected range. The ranges can be normal (green), minor (yellow), and critical (red). Each group of metrics can trigger a status change independently.

The following figure is an example of mapping the prototype Host Health Indications to the ITU perceived severities, defined in https://tools.ietf.org/html/rfc3877. The values for the Host Health Indicator are used as examples for the prototype, many other mappings and alternate ranges and values for the host health indications can be used.

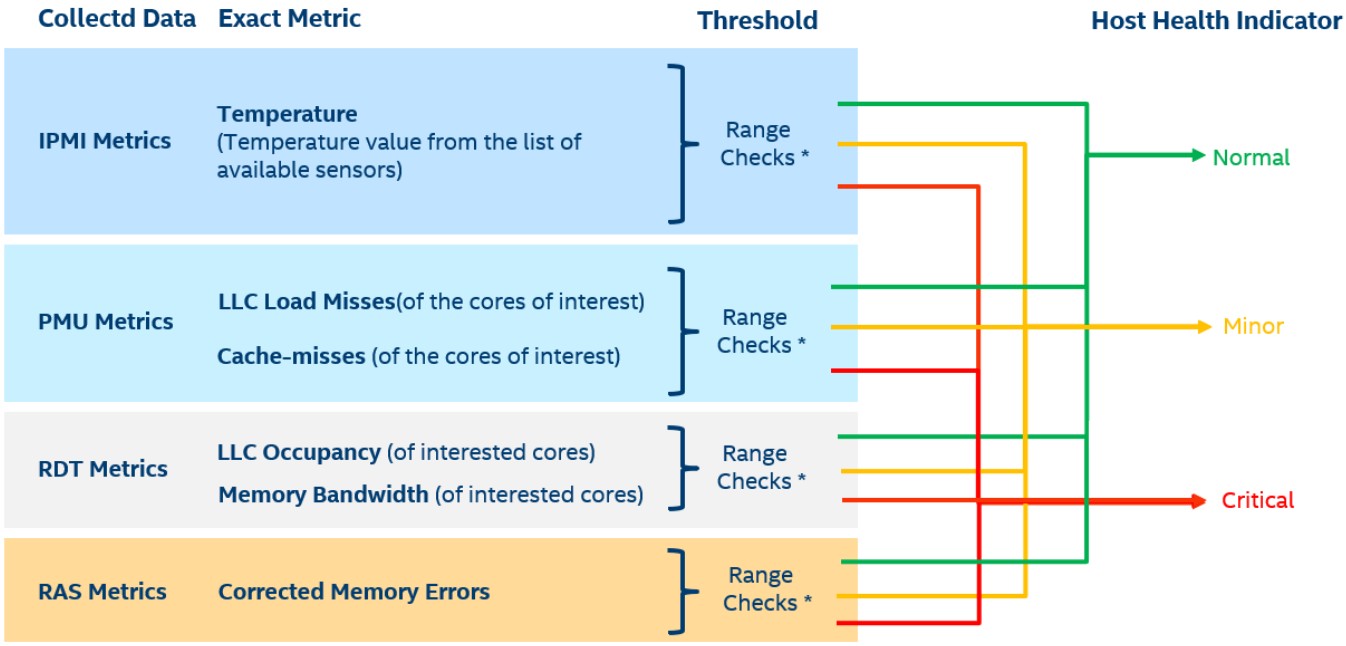

Figure 3. Host Health Indicator Sample Threshold Decision Matrix

3.3 Automated Action – Telemetry Aware Scheduling

Automated action can be carried out in the VNFM, NFVO or VIM depending on the type of automated action required and the appropriate MANO layer. In the prototype, corrective actions are taken within the VIM. In an NFV deployment, the Service Orchestrator and the VNFM will receive insights from analytics and take consequential actions in line with the responsibilities of each layer of MANO (VIM, VNFM, NFVO), since re-deployments will require knowledge of the local service, VNF life cycle, and the end to end service.

The generic Kubernetes Scheduler ignores platform and other telemetry when making placement decisions. TAS is a Kubernetes scheduling extension that takes telemetry into account, enabling automated actions and intelligent placement of workloads to be made based on cluster/platform metrics and the Service Level Agreements of individual workloads.

TAS enforces a rule-based, user defined telemetry policy on the cluster to influence up front pod placement and control pod distribution later in the lifecycle. This policy is implemented as a Kubernetes Custom Resource Definition and can be treated as if it were part of the native Kubernetes API by its creators and consumers.

The TAS telemetry policy takes the form:

```
apiVersion: telemetry.ie/v1
kind: TelemetryPolicy
metadata:
  name: health-policy
  namespace: default
spec:
  strategies:
     scheduleonmetric:
      rules:
        - metricname: health_metric
          operator: GreaterThan
    dontschedule:
      rules:
      - metricname: health_metric
        operator: Equals
        target: 1
      - metricname: health_metric
        operator: Equals
        target: 2
    deschedule:
      rules:
      - metricname: health_metric
        operator: Equals
        target: 2
```

Three scheduling strategies are named in the policy. The first, “scheduleonmetric”, prioritizes nodes based on the metric passed to it. The second, “dontschedule”, excludes nodes which break the declared ruleset from the list of candidate nodes. The third, “deschedule”, removes a pod from a node violating the attached ruleset.

The telemetry data used for policy enforcement are consumed from the Prometheus database through the Kubernetes Custom Metrics API. Metrics can be compared with “Greater Than”, “Equals” and “Less Than” operators.

TAS is deployed as a single pod and sits beside the generic scheduler on the Kubernetes Master. When a pod request with a telemetry policy is created a list of candidate nodes based on default scheduling rules are sent to TAS. TAS then uses the pod’s policy definition combined with up to date platform metrics to find the optimal node for pod placement.

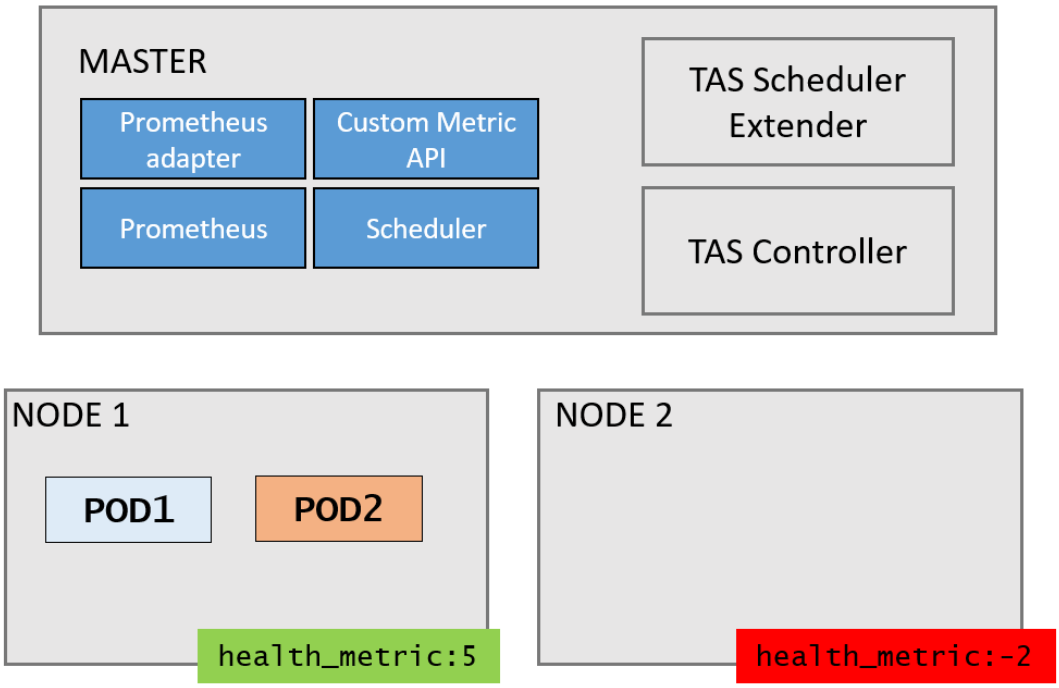

Figure 4. Topology of a Kubernetes system integrated with Telemetry Aware Scheduling

Once a workload is placed TAS continues to monitor metrics specified in the telemetry policy to ensure compliance and trigger redistribution of workloads if required. The "deschedule" policy allows workloads to be descheduled in case of violation and subsequently be placed on a non-violating node.

The above policy, for example, will not schedule affected workloads on any node which has a health metric of 2 or 1. If a node's health metric reaches 2 TAS will run it's critical deschedule policy, removing that workload from the node altogether and causing it to be placed on a non-violating node.

3.4 Error Injection

Intel® Run Sure technology provides set of resilient system and resilient memory technologies that are designed to reduce frequency of downtime while reducing the cost of downtime. Resilient memory technologies provided as part of Intel® Run Sure technologies ensure data integrity by recovering from memory fatal errors and reduces the service cost.

Linux's* mcelog is a daemon for handling and reporting hardware errors. It is able to use trends in corrected error reports to implement specific error prevention algorithms. Errors will be usually reported to the operating system using the standard x86 Machine Check Architecture (MCA) registers. Uncorrected memory errors – that is data corruption – are reported using a machine check exception and handled directly by the kernel (for example, by killing the affected process or shutting down the system down).

Corrected errors are polled by a timer or interrupt handler and reported to the mcelog daemon. The mcelog daemon decodes the error and does accounting and other book keeping.
To stress the server with memory faults and simulate an unhealthy state of host, mce-inject and stress-ng tools are used as below:

1. `mce-inject` allows to inject machine check errors on the software level into a running Linux kernel. This tool is intended to test the machine check handling code. The injection only happens on the software level and does not simulate full machine check handling on the platform level. This tool is useful specifically for Intel RAS.
   https://git.kernel.org/pub/scm/utils/cpu/mce/mce-inject.git/
2. Stress-ng will stress a host in various selectable ways. It can introduce thermal overruns, high memory bandwidth, L2 cache misses, etc. This tool is useful for specific to Intel® Resource Director Technology (RDT)/Performance Monitoring Unit (PMU).
   https://wiki.ubuntu.com/Kernel/Reference/stress-ng

4 Closed Loop Automation – Platform Resiliency Use Case Description

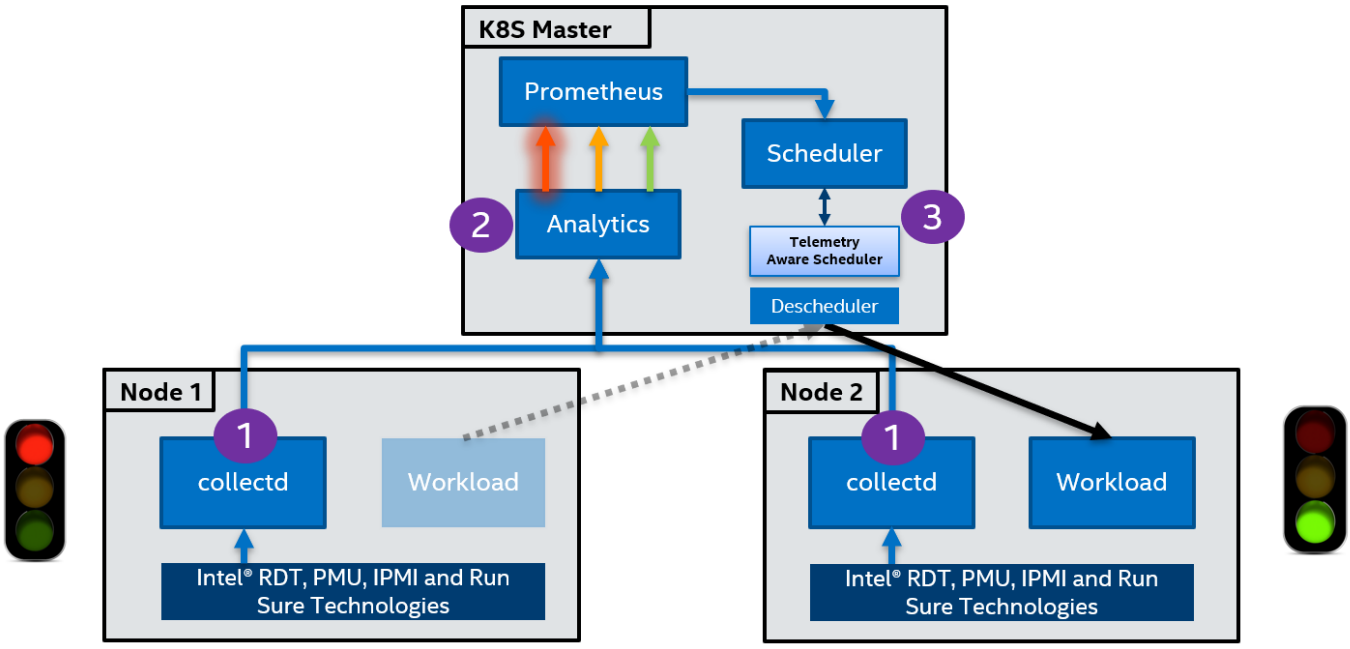

Figure 5. High level Prototype Architecture

4.1 Method of Operation

The stress-ng application is used in this prototype both as application under test scheduled by Kubernetes* and a noisy neighbor scheduled by the user on an isolated set of CPU cores.

1. Using three nodes in the cluster to showcase the functionality, Node 1 and 2 have collectd* running, collecting metrics from the platform. Refer to Section 3.1 for further details. A standard workload is running without issue on Node 1 when various platform errors can be injected to mimic "critical" network infrastructure issues.
2. The Streaming analytics solution aggregates metrics from all nodes and processes the data as outlined in Section 3.2. The errors injected in Step 1 will be picked up through the metrics and once analyzed, will provide a "Critical" Host Health indicator to Prometheus*.
3. Kubernetes Telemetry aware scheduler picks up this indicator from Prometheus and takes immediate remedial action to move the workload from Node 1 to Node 2 to ensure the service can continue with minimum disruption.

Refer to Section 4.2 for all possible "Reliability Aware" placement scenarios used in this prototype.

4.2 Reliability Aware Placement Scenarios

The "Host Health Indicators" provided to the Telemetry Aware Scheduling in this prototype determine the workload placement decisions the scheduler will take.

There are three reliability aware placement scenarios configured here (Refer to Section 3.4 to understand how each indicator is calculated):

1. **Red Scenario**: A platform will provide a "Red" host health indicator to the Telemetry Aware Scheduling if it has been deemed to be in a "critical" state. Once TAS receives this critical indicator it will deschedule workloads linked to the specific policy on that node and schedule on a "healthy" node.
   a. Users can trigger a red scenario by using the stress-ng tool (with specified options) pinned to 4 isolated cores.
   b. One way is to use a few workers exercising the CPU by sequentially working through all the different CPU stress methods combined with workers continuously calling sync to commit buffer cache to disk simultaneously calling mmap/munmap and writing to the allocated memory.
      For example:
      - `stress-ng --cpu 4 --io 10 --vm 40 --vm-bytes 1024M -t 90s --taskset 1-4`
      - `stress-ng --cpu 4 --io 10 --vm 50 --vm-bytes 1024M -t 90s --taskset 1-4`

      The second way is to use stress-ng with a few workers that perform various matrix operations on floating point values, where matrix size is setup on 8192.
      For example:
      - `stress-ng --matrix 10  --matrix-size 8192 -t 90s --taskset 1-4`
      - `stress-ng --matrix 200  --matrix-size 8192 -t 90s --taskset 1-4`
   c. On a Kafka stack, users can define the "RED" scenario using ksql queries with proper threshold conditions. Once the streaming data hits the threshold conditions, Users can define a correlated ksql query as Host health Indicator. The following are example queries that we used for Intel RAS plugin with mcelog errors:

```
                ksql > create stream mcelog as select * from collectd_stream where plugin =
'mcelog' and \
                type_instance = 'corrected_memory_errors' and plugin_instance =
'SOCKET_0_CHANNEL_0_DIMM_any';

                ksql > create stream HOST_HEALTH_DB as select host,type,type_instance,case
when `VALUES`[0]='0' and \
                `VALUES`[0]<'0.1' then 'Green' when `VALUES`[0]>'0.1' and `VALUES`[0]<'0.3'
then 'Yellow' else 'Red' \
                end as HH_Status from mcelog;

                                ksql > create stream Host_Health_Status as select host,case
when hh_status='Green' then 0 when \
                                hh_status='Yellow' then 1 else 2 end as Status from
Host_Health_DB;

            ksql > create table HostHealth_Indicator as select host,MAX(Status) from
Host_Health_Status \
                                window tumbling (size 10 seconds) group by host;
```

   Note: This scenario is not intended for production environment.

2. **Amber/Yellow Scenario**: An amber indicator will identify that a minor fault has been found on the node. All workloads can continue to work as normal, but the scheduler will not schedule any further workloads with this policy on this node until planned maintenance can occur to return the node to a "healthy" status. TAS will place new workloads with this policy on an alternative "healthy" node.
   a. Yellow scenario user can be triggered by running stress-ng tool with workers that perform random wide spread memory read and writes to thrash the CPU. Stress-ng is pinned to 4 cores (isolated).
      Example:
      - `stress-ng --cache 400 --cache-flush -t 90s --taskset 1-4`
      - `stress-ng --cache 500 --cache-flush -t 90s --taskset 1-4`
      - `stress-ng --cache 600 --cache-flush -t 90s --taskset 1-4`
   b. Refer to the queries mentioned above for "RED" scenario that provides the threshold condition for Amber/Yellow scenario as well.
3. **Green Scenario**: A green indicator identifies all the healthy nodes that TAS can use to make a workload placement.

5 Conclusion

We show how rich platform telemetry, combined with analytics and MANO components are used for closed loop automation of a set of resiliency and service healing actions. Closed loops can occur at the VIM layer, as shown with Kubernetes*, using the same components, actions can also be taken further up the MANO stack in the VNFM or NFVO layers, with the decisions informed by platform telemetry and analytics insights. The paper shows how platform telemetry can be used to respond and close the loop to maximize service uptime and customer QoE by proactively responding to advance insight from the platform.

1019/DN/PTI/PDF 341450-001US