SaaS OverCloud 기술 문서 #4

단일 하이브리드 클라우드 인프라/플랫폼 상의 SaaS 응용들의 동작 호환성 검증을 위한 Agent 기반 Resource Visibility 기능 설계/구현

Document No.	SaaS OverCloud #4
Version	1.0
Date	2016-10-11
Author(s)	GIST Team

■ 문서의 연혁

버전	날짜	작성자	비고
초안 - 0.1	2016. 08. 17	김종원, 박선, 김남곤, 한정수	
0.2	2016. 10. 11	Usman Muhammad	
0.3	2016. 10. 14		
0.5			
0.7			
0.8			
0.9			
1.0			

본 문서는 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.R7117-16-0218, 이종 다수 클라우드 간의 자동화된 SaaS 호환성 지원 기술 개발)

The reserach was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.R7117-16-0218, Development of Automated SaaS Compatibility Techniques over Hybrid/Multisite Clouds)

Contents

SaaS OverCloud #4. 단일 하이브리드 클라우드 인프라/플랫폼 상의 SaaS 응용들의 동작 호환성 검증을 위한 Agent 기반 Resource Visibility 기능 설계/구현

표 목차

그림 목차

SaaS OverCloud #4.

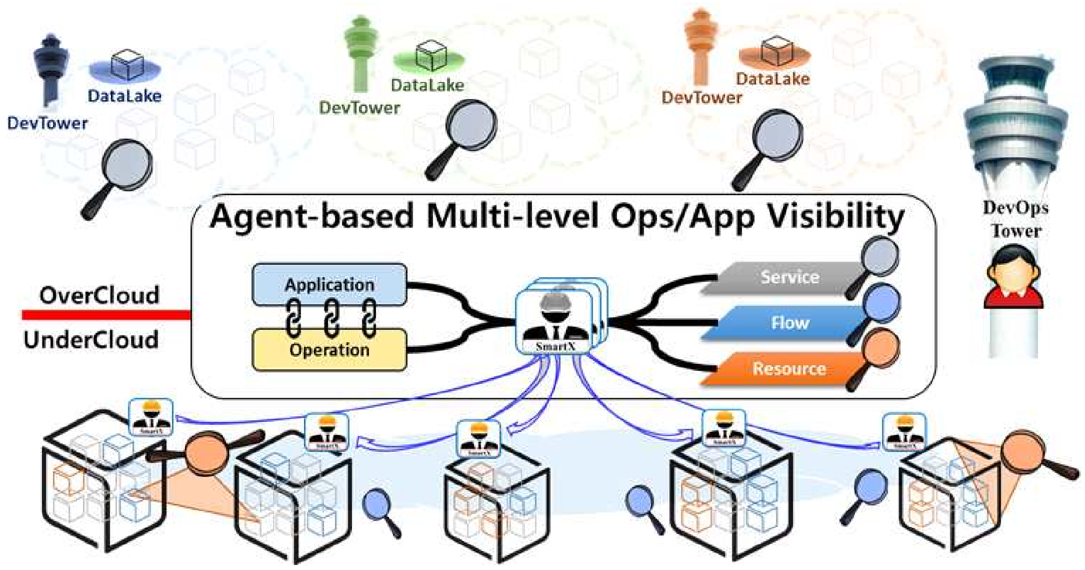
단일 하이브리드 클라우드 인프라/플랫폼 상의 SaaS 응용들의 동작 호환성 검증을 위한 Agent 기반 Resource Visibility 기능 설계/구현

1. Visibility Overview

This chapter discusses overall goal of multi-level visibility in SAAS OverCloud Project. Also, other important concepts like logical clusters, DevOps tower and data lake are covered in this chapter.

1.1. OverCloud Project Multi-level Visibility Goals

Mostly SAAS OverCloud projects provide automation tools for provisioning and orchestration but lacks visibility support. SAAS OverCloud Project, multi-level visibility goal is to provide the visibility support tools to developers, so that they can have complete visibility of their applications. Also at the same time, we need to provide operator with the specialized visibility support tools to monitor and troubleshoot when the developers face problems during their applications executions. With the increased complexity of resources composition it is becoming really critical to identify key visibility sources. This visibility information should not only been captured from single visibility level instead it should be from multi-level e.g. Resource-level, Flow-level and Workload-level. The overview diagram of Agent-based Multi-level visibility is shown in [Fig. 1].



[Fig. 1] Multi-level Visibility Challenges

Agent-based Multi-level Ops/App Visibility should cover following functions: collection of Ops/App visibility data from multiple sources and visibility levels (e.g. resource, flow and workload), reliable delivery of visibility data from multi sites to SmartX Visibility Center, Validation of visibility data at SmartX Visibility Center, Integration of collected visibility information, Storage support for visibility data through SmartX Data Lakes, Finally, providing analysis/visualization tools for operators and developers to utilize this visibility data in analysis/troubleshooting.

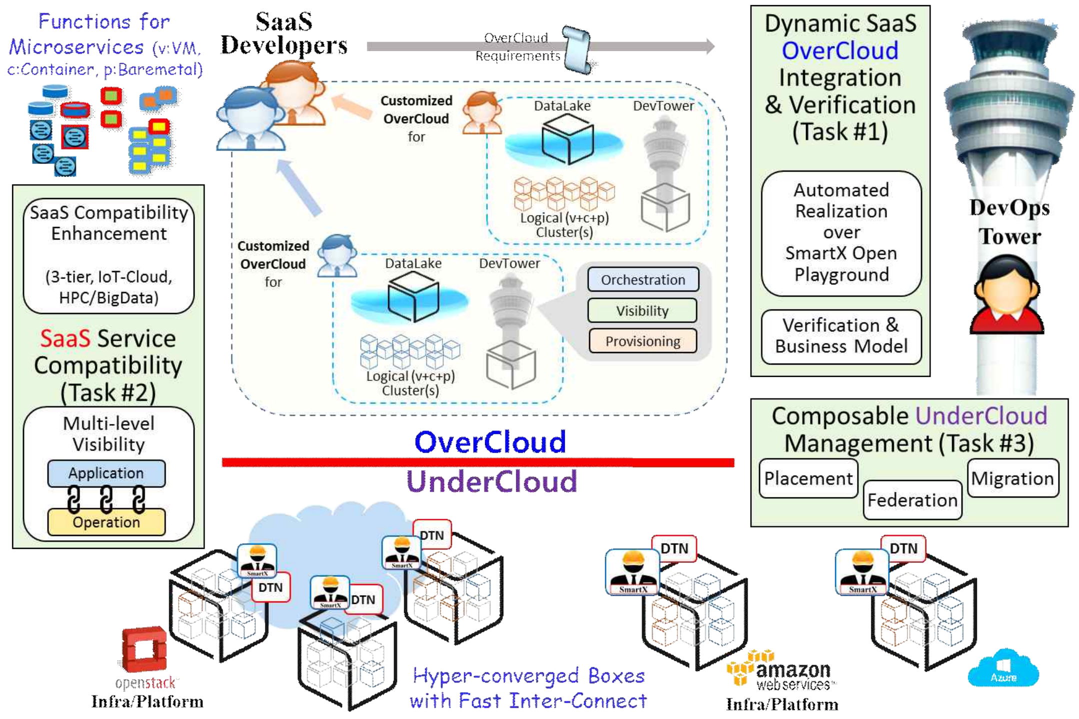
As, a first step towards Multi-level visibility we started with resource-level visibility. In Resource-level visibility, first of all, essential visibility sources are identified to capture decisive performance metrics. Next, step is to deploy visibility tools, which can collect visibility data from identified sources. Next, visibility data is required to be delivered to DevOps Tower for further processing. Then, application/operation visibility data is required to be integrated and stored for analysis and visualization purposes. Finally, this application/operation data is required to be visualized using specialized visualization tools.

1.2. DevOps Tower

DevOps Tower is a centralized location to monitor and control operation of SAAS UnderCloud/OverCloud resources. DevOps Tower is composed of multiple centers like Provisioning Center, Orchestration Center, Visibility Center and Intelligence Center. The role of these centers is to provides specific tools to carry out certain operation tasks like SmartX Box installation and configuration, resources provisioning, flows steering and visibility collection and visualizations.

1.3. Logical Clusters

Logical cluster is an abstract representation of one or more p+v+c (physical, virtual and container) resources. These clusters are used to manage SAAS Developers workload, failover and application requirements. In SAAS OverCloud project goal is to provide each SAAS Developer with DevTower to manage their logical cluster(s) and Data Lake to store visibility/configuration information. In Fig. 2, customized logical cluster(s) for different SAAS Developers are shown.


Functions for Microservices (v:VM, c:Container, p:Baremetal)
SaaS Developers
OverCloud Requirements
Dynamic SaaS OverCloud Integration & Verification (Task #1)
Automated Realization over SmartX Open Playground
Verification & Business Model
DevOps Tower
Customized OverCloud for
DataLake
DevTower
Logical (v+c+p) Cluster(s)
Orchestration
Visibility
Provisioning
SaaS Compatibility Enhancement
(3-tier, IoT-Cloud, HPC/BigData)
SaaS Service Compatibility (Task #2)
Multi-level Visibility
Application
Operation
OverCloud
UnderCloud
Composable UnderCloud Management (Task #3)
Placement
Federation
Migration
DTN
openstack
Infra/Platform
Hyper-converged Boxes with Fast Inter-Connect
amazon web services
Azure

[Fig. 2] SAAS OverCloud Project Overview Diagram

1.4. Data Lake

Data Lake is used to store application/operation visibility data. Data Lake refers to a storage capacity that contains a large amount of data in its native format until it is required (e.g. to be used for visualization or analysis purposes). We may use aggregation of HDFS [1], Elasticsearch [2], MongoDB [3] and InfluxDB [4]. Note that, HDFS is Java-based file system that provides scalable and reliable storage, and it was designed to span large clusters of commodity servers. Elasticsearch is a distributed, open source search and analytic engine using index-based approach for data storage and retrieval. MongoDB is NoSQL, document-oriented database tool. InfluxDB is an open source time series database with no external dependencies. It's very useful for recording metrics, events and performing analytics on operation/application visibility data. We setup SmartX Visibility Center to provide visibility processing and storage capabilities. SmartX Visibility Center is running Ubuntu 14.04.3 LTS Operating System. To provide Data Lake facility in SmartX Visibility Center, we used MonogDB, Elasticsearch and InfluxDB data stores.

```
# MongoDB installation
# sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
# echo"deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2
    multiverse"| sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
# sudo apt-get update
# sudo apt-get install –y mongodb-org

# Elasticsearch Installation
# wget
https://download.elastic.co/elasticsearch/elasticsearch/elasticsearch-1.7.2.deb
# sudo dpkg –i elasticsearch-1.7.2.deb
# sudo update-rc.d elasticsearch defaults


# InfluxDB Installation
# wget https://dl.influxdata.com/influxdb/releases/influxdb_1.0.2_amd64.deb
# sudo dpkg -i influxdb_1.0.2_amd64.deb
```

2. Resource-level Visibility

This, chapter will discuss details about types of resources required to be considered for monitoring in SAAS OverCloud environment. Next, we discuss visibility collection mechanism. Then, we will discuss about delivery of application/operation visibility data from remote sites to DevOps Tower and it's storage to SmartX Data Lake. Finally, we discuss about how to make this data readily available to become useful in analysis and visualizations.

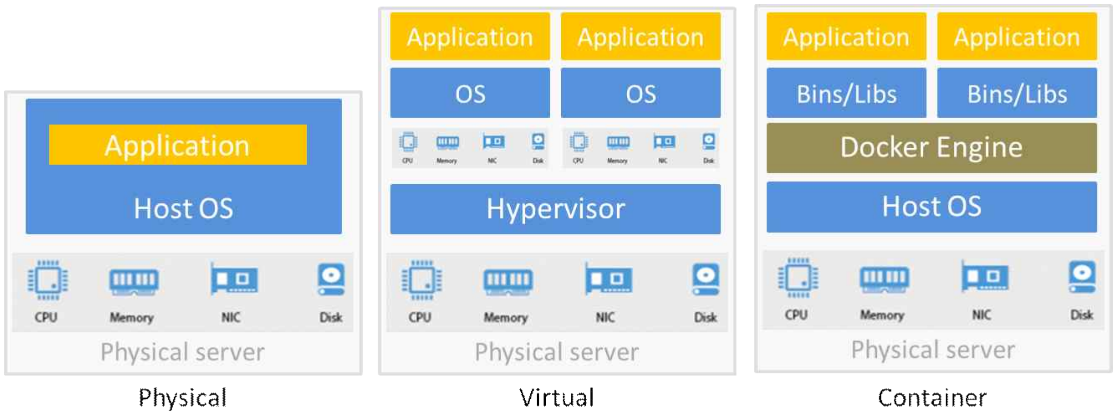
2.1. Resources Types and Key Parameters

First of all, we need to identify, what kind of resources are available to be considered for monitoring. In our setup, we have three kind of resources p+v+c (physical, virtual and container).

Let's, start with most basic type of resource that is physical server. Each physical server has it's own hardware: cpu, memory, storage, processing and network resources [5]. By using this hardware host operating system is loaded. The host operating system provides you facility to run user applications. At physical server level all the things are simple and straight forward. The Figure 4 shows physical server with simplest architecture. We deploy special features to physical servers to fulfil specific SAAS OverCloud requirements. Therefore, from here onwards, we will refer physical server by using name Box. At Box level, we need to measure cpu, memory, network and disk components for health monitoring. We are running Ubuntu 14.04.2 LTS in physical box.

Another physical resource is Physical links. Physical links are used to provide networking capabilities among Boxes which are located and spread across multiple sites. Ensuring Boxes interconnectivity is one of key requirement of SAAS OverCloud operators to ensure smooth execution of developer applications.

Second, type of resources are virtual resources. Virtual Machine (VM) is one of virtual resource. It provides an emulation of real computer that executes programs like a real computer. Virtual machines run on top of pBox using a hypervisor. A hypervisor is piece of software that VM's run on top of. The hypervisor runs on top of pBox. Each Virtual Machine has it's own virtual devices such as cpu, memory, network interface cards and disks. We used OpenStack cloud for virtual machine management. In Fig. 3 physical server with virtualization support is shown. SAAS developers are provided with the virtual machines to host their applications on them. Therefore, virtual machines performance also need to be monitored by using same parameters like pBox. Another type of virtual resource is the virtual switch. This, virtual switch is a software that resides in a software layer that is placed in a pBox that is hosting virtual machines and containers, such as docker, have logical or virtual Ethernet ports. These, virtual ports are connected to virtual switch [6]. So, virtual switch is responsible for providing virtual networking inside the box and it's status should also be monitored to ensure virtual machines have access to the network. We are using Open vSwitch version 2.3.1 to provide virtual switch functionality.



[Fig. 3] Physical vs Virtual vs Container

Finally, most recently containers are introduced. Unlike virtual machine which provides hardware level virtualization; a container provides operating-system level virtualization by abstracting the "user space" [7]. Figure 4 shows that containers are packaged up just the user space and not the kernel or virtual hardware like a virtual machine does. Each container gets it's own isolated user space to allow multiple containers to run on a single pBox. As you can see all operating system level is shared among containers except bins and libs. In Table 1, performance metrics to measure are listed.

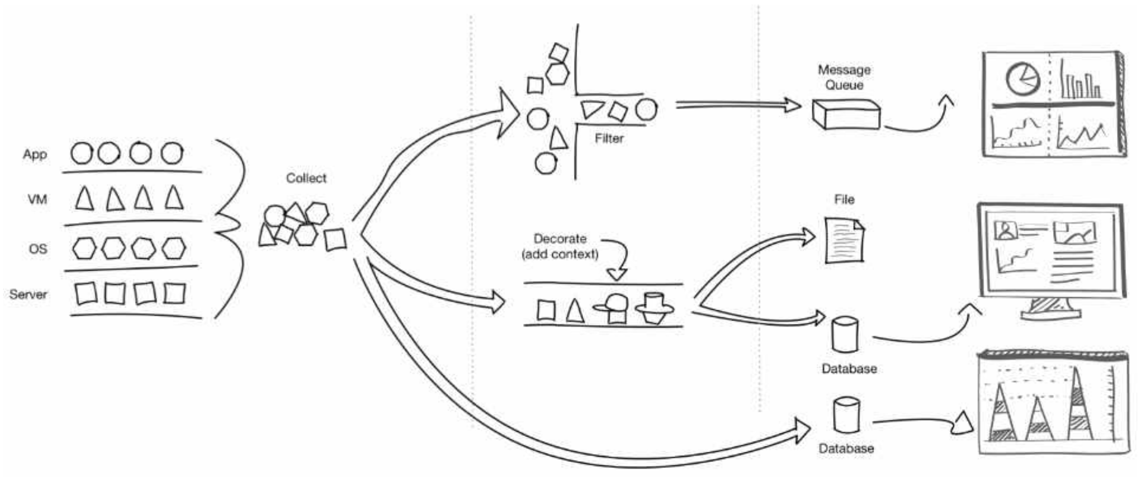
[Table 1] Performance Metrics

Type of Resource	Resource Name	Resource Metric
physical	CPU	load1, load5, load15, utilization_percentage, avg-cpu-/user/nice/system/iowait/steal/idle
	Memory	mem-/free/available/used/total
	Disk	space-/free/reserved/used, space-percentage-/free/reserved/used, inodes-/free/reserved/used device-/name/type
	Network	bytes-/recv/sent, packets-/sent/recv
virtual	CPU	load1, load5, load15, utilization_percentage
	Memory	mem-/free/available/used/total
	Disk	space-/free/reserved/used, space-percentage-/free/reserved/used
	Network	bytes-/recv/sent, packets-/sent/recv

2.2. Visibility Collection

Once, we figured out type of resources required to be monitored, next, we need to develop or utilize existing open source tools to collect visibility information. We used Intel Snap Framework [8] for visibility collection from different systems. Intel Snap is an Open-source telemetry framework written in Glolang for consumption of data center telemetry easier. Snap Framework offers three types of plugins: collectors, processors and publishers. Snap Workflow starts with data collection at all layers by using collector plugins, then visibility data is transformed with basic filtering using processor plugins, and finally visibility data is published to one or more platforms for consumption using publisher plugins. Fig. 4 shows simplified Workflow of Snap Framework.

To collect visibility data from boxes, first, we need to get latest release of Snap Framework from github (https://github.com/intelsdi-x/snap/releases) Next, we need to start Snap daemon. Then, we need to start collector plugins, processor plugins and publisher plugins. A configuration file is used to control

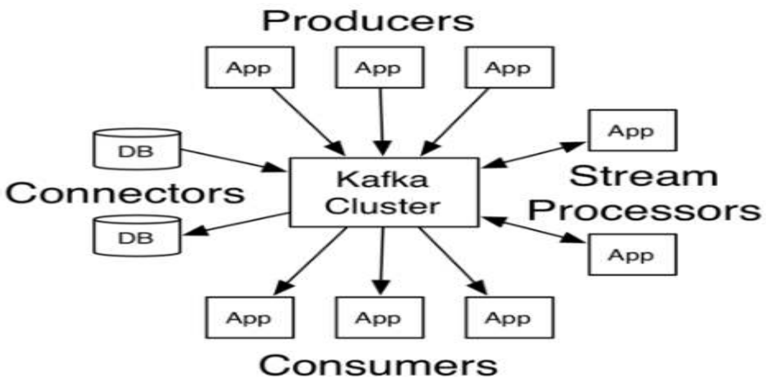


[Fig. 4] Snap Framework Simplified Workflow

which metrics to be measured. After that a task is required to be created. A task is Workflow which uses this configuration file for starting complete process shown in Fig. 4. Also, we created custom shell script to automate the execution of Snap daemon, plugins and tasks processes.

2.3. Visibility Delivery

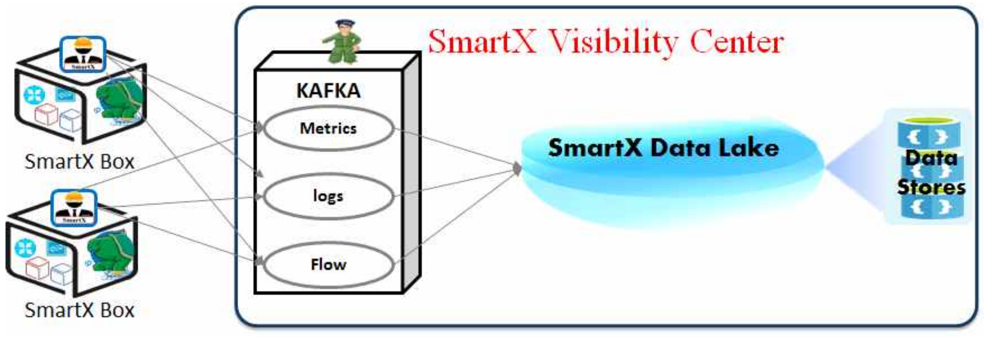
In previous section, we discussed about visibility metrics measurement by using Snap Framework. After collection next, step is to deliver visibility data to SmartX Visibility Center for further processing e.g. analysis or visualization. We used Apache Kafka [9] for visibility data delivery. Apache Kafka is an open-source message broker project developed by Apache Software Foundation written in Scala. Kafka provides a unified, high throughput, low-latency platform for handling real-time data feeds. Kafka has three components namely Producers (writes data to Kafka topics), Kafka Cluster (stores data in partition identified by name “topic“) and consumers (end applications which consumes stored data). Fig. 5 shows Kafka’s overall Workflow. First of all we need to deploy Apache Kafka in SmartX Visibility Center. Kafka uses Apache ZooKeeper [10] for log management and synchronization. So, first we get ZooKeeper version 3.4.6 and start zookeeper server (bin/zookeeper-server-start.sh config/zookeeper.properties). To deploy Apache Kafka, we need to get latest version (0.10.0.1) of Kafka from (https://kafka.apache.org/downloads). After downloading Kafka distribution, we start Kafka broker (bin/kafka-server-start.sh config/server.properties) and create required topics for storing visibility data. By using Snap publisher plugin for Kafka, visibility data is delivered to Kafka topic by using Kafka publisher API.



[Fig. 5] Apache Kafka Workflow

2.4. Visibility Data Storage

In previous section, we discussed about Kafka messaging system and visibility data storage to Kafka topic. Now, in this section, we will focus on moving data from Kafka topics to SmartX Data Lake. Operation data from Kafka topics is consumed using Kafka Consumers API. We wrote custom Java-based consumer to read data from Kafka topic and store to InfluxDB. InfluxDB is an open-source time series database. It is written in Go and optimized for fast, high availability storage and retrieval of time series data in fields such as operations monitoring, application metrics and real-time analytics. In Fig. 6 overall flow of operation visibility data is shown from SmartX Box till SmartX Visibility Center. Custom Java-based consumer is packaged as a jar file and executed by using java –jar kafka-influx-linker.jar command.



[Fig. 6] Flow of Operation Visibility Data from Box to SmartX Visibility Center

We also maintain a separate database that stores updated configuration/status data for p+v resources. This, configuration/status database is maintained using MongoDB collections. Data from separate collections is integrated using specific keys e.g. physical box identifier. In Table 2, name and purpose of each collection is listed. Currently, Java-based plugins are used for collecting latest resources states and updating MongoDB collections on regular intervals. Our visualization tool makes use of these collections for displaying graph-based topology and associated p+v resources states.

[Table 2] Configuration/Status database MongoDB collections

No.	Collection Name	Collection Purpose
1	configuration-multiview-users	Contains list of users
2	configuration-pbox-list	Contains list of physical boxes and their availability status
3	configuration-vbox-list	Contains list of OpenStack vm's and their status
4	configuration-vswith-list	Contains ovs topology information
5	configuration-vswitch-status	Contains status of ovs switches

3. Visualization

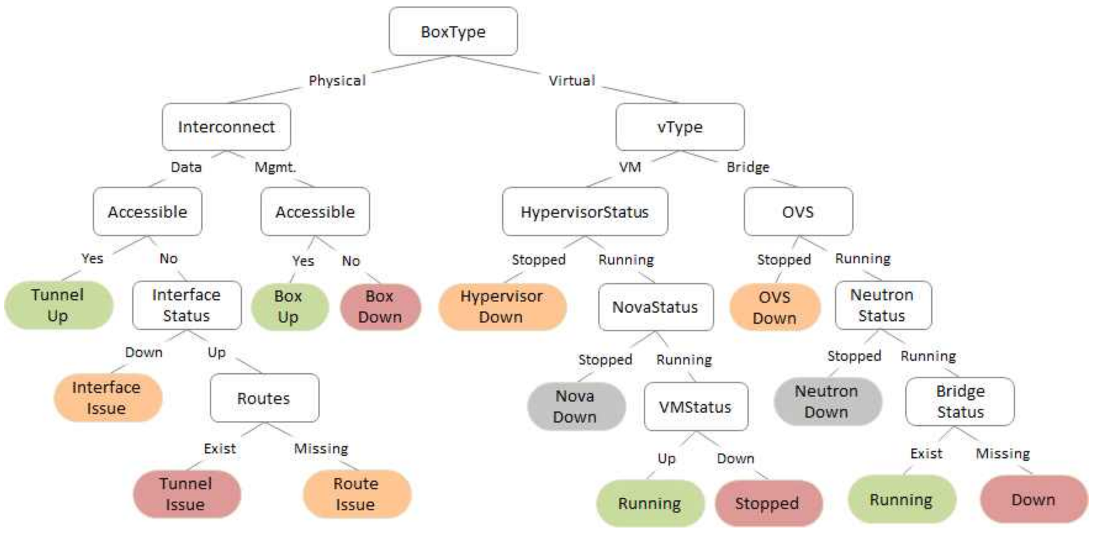
In this chapter, we will discuss about the visualization of visibility data which we captured from SmartX Boxes and stored in SmartX Data Lake.

3.1. Visualization Requirements

In this section, we will discuss about visualization requirements of physical-virtual resources. We, require an innovative solution to offer single view for Cloud and SDN segments by covering both p+v components. Also, this visualization must assure integrated visualization of visibility data collected from multiple sources for faster troubleshooting of SAAS OverCloud resources. p+v topological visualization is one of the approaches to present all the essential information (e.g. p+v resources interconnections, resources status, associated performance metrics etc.) in single view using visual support tools.

3.2. Visualization Design

Initial design of p+v topological visualization is shown in Fig. 7. p+v topological visualization, design can be considered as a decision tree where each node represents attribute, branches represent values of attributes and leafs represents classification. Different colors at leaf nodes represent special status messages to facilitate the developers for faster troubleshooting by providing



[Fig. 7] p+v Topological Visualization Design exact visual information. Let's, follow a particular path in decision tree to understand how decisions are made for enabling p+v topological visualization. Process starts by identifying type of resource. In case of physical resource, physical interconnects are checked. If we follow the "Data" branch and result of attribute accessible is "yes" then tunnel is up data plane is working fine. In case, data plane is not accessible then interface status is checked and based on interface status attribute next decision is made. In case, interface is up then routes are checked and based on result of routes checking appropriate decision is made like "tunnel issue" or "route issue".

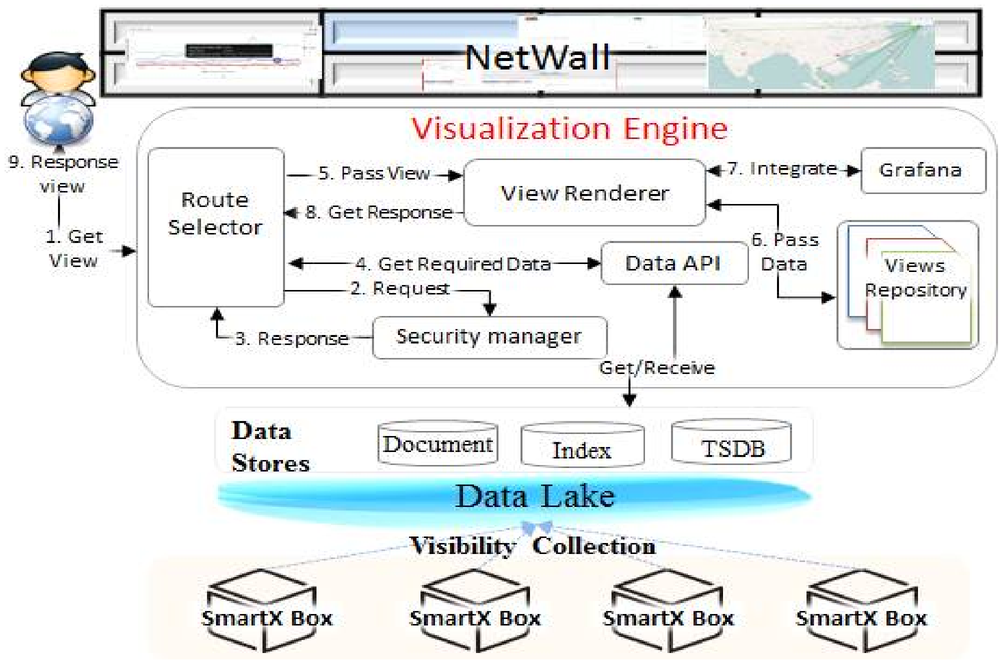
The p+v topological visualization needs several functionality modules for easier processing and flexible integration. Workflow of p+v topological visualization is shown in Fig. 8. First of all, Route Selector module is responsible for managing view requests and interaction with other modules. Next, Security Manger module is responsible for user authentication & authorization. Then, Data API role is to provide required data for visualizations. After that, View Renderer module generates required view using data object send by Route Selector. By using graph-based visualization library Vis.js [9] data object is translated into graphical components like nodes and links.

3.3. Visualization Implementation

We used Grafana [11] for performance metrics dashboard creation. Grafana provides a powerful and elegant way to create, explore and share dashboard and data. Grafana offers support for number of backhand data stores. So, we deployed Grafana for creating custom dashboard and graphs. We deployed Grafana version 3.1.1 using debian package:

```
# wget
https://grafanarel.s3.amazonaws.com/builds/grafana_3.1.1-1470047149_amd64.deb
# sudo apt-get install -y adduser libfontconfig
# sudo dpkg –i grafana_3.1.1-1470047149_amd64.deb
```

Main challenge in p+v topological visualization is to collect information from network and show dynamically on GUI using graph-based style with nodes (representing resources) and edges (representing interconnects). So, we considered Vis.js [12] library for graph-based drawings. Vis.js offers a simple and easy to use graph-based library for interactive network graphs development. Vis.js can also handle large amounts of dynamic data, and



[Fig. 8] p+v Topological Visualization Workflow

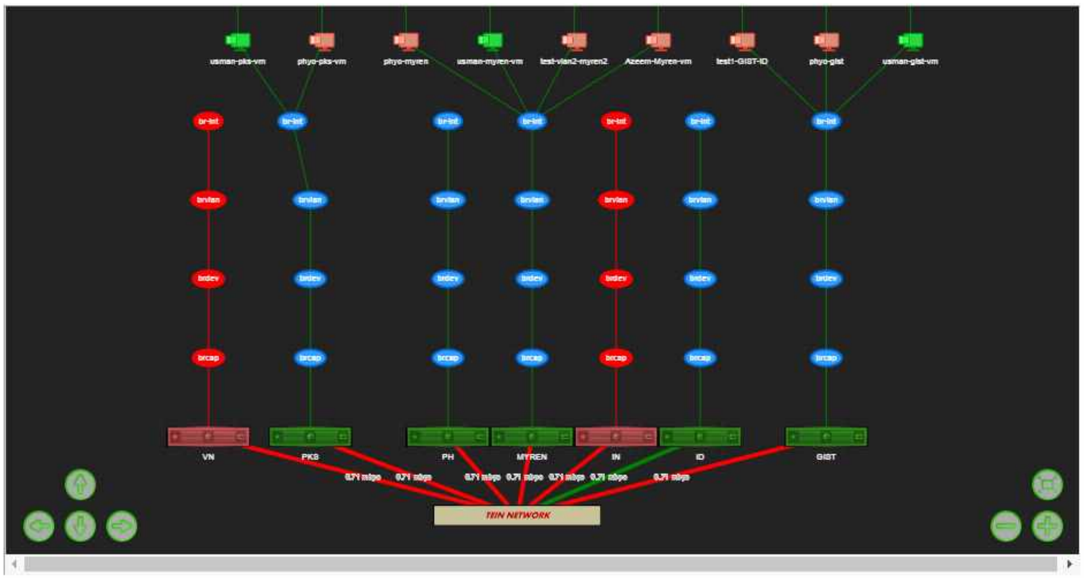
interaction with the data stores. We downloaded Vis.js library using "npm install vis" and used in our Javascript code for creating topology.

We used Node.js [13] to create a server for web application since Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Furthermore, Node.js provides number of plugins, to help users to easily build complex applications. We installed Node.js server version 4.x using commands:

```
# curl -sL https://deb.nodesource.com/setup_4.x |sudo -E bash -
# sudo apt-get install –y nodejs
```

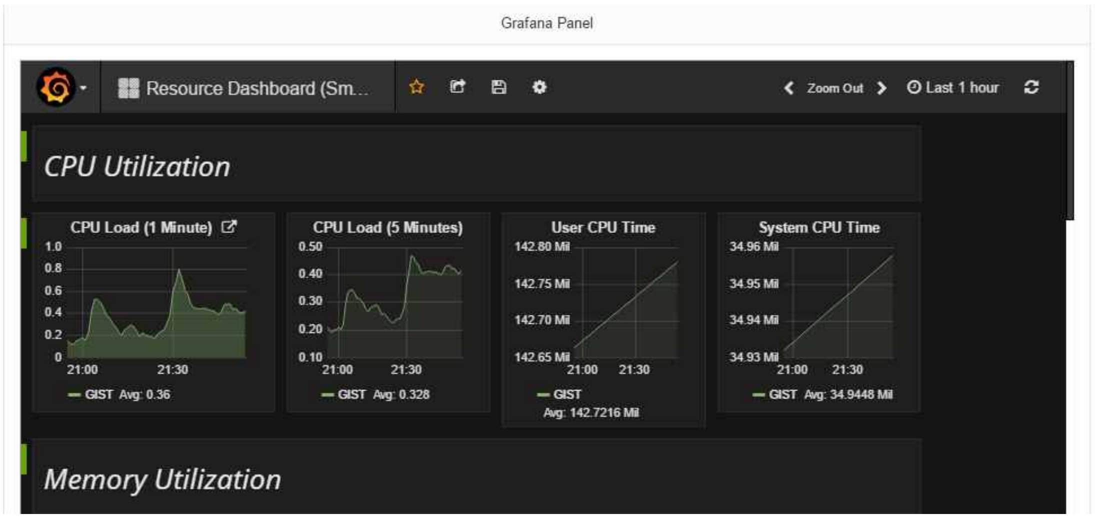
Snippet of Javascript code for server.js is shown below which is used for dependencies inclusion, routes redirection and data api calls.

```
/**
 * Module dependencies.
 */
var express = require('express');
var ArticleProvider = require('./resource-mongodb').ArticleProvider;
var RouteProvider = require('./route-mongodb').RouteProvider;
var BoxProvider = require('./multiview-hierarchical-mongodb').BoxProvider;
var UserProvider = require('./user-mongodb').UserProvider;
var http = require("http");
var app = module.exports = express.createServer();
var client = require('socket.io').listen(8080).sockets;
// Configuration

app.configure(function(){
  app.set('views', __dirname + '/views');
  app.set('view engine', 'jade');
  app.use(express.bodyParser());
  app.use(express.methodOverride());
  app.use(require('stylus').middleware({ src: __dirname + '/public' }));
  app.use(app.router);
  app.use(express.static(__dirname + '/public'));
});

app.configure('development', function(){
});
app.configure('production', function(){
  app.use(express.errorHandler());
});

//Define Application Routes
//MultiView Data Integration & View Access
var resourceProvider = new ResourceProvider();
app.get('/resourcecentricviewops', function(req, res){
    var boxList          = null;
    var switchList       = null;
    var instanceList     = null;
    var serviceList      = 0;
    var ovsBridgeStatus = null;
    var pPathStatus      = null;
    resourceProvider.getpBoxList( function(error,boxobj)
    {
        boxList = boxobj;
        console.log( boxList);
        showView();
    })
```



[Fig. 9] p+v Topological Visualization

In Fig. 9, snapshot of topological visualization is shown. Square boxes are showing the physical box and their availability status. Oval shape shows the topology and status of virtual switches inside the physical box. At the top of br-int virtual switch, monitor image shows the deployed OpenStack virtual machines in particular box. Lines shows the interconnectivity status of those resources. When, we click on any of physical box, we get metrics dashboard as shown in Fig. 10.



[Fig. 10] Performance Metrics Visualization using Grafana

References

[1] Apache Hadoop, HDFS, http://hortonworks.com/apache/hdfs/
[2] Elasticsearch, https://www.elastic.co/
[3] MongoDB, https://www.mongodb.com/
[4] InfluxDB, https://www.influxdata.com/
[5] Physical Servers vs Virtual Machines, https://www.veeam.com/blog/why-virtual-machine-backups-different.html
[6] Virtual Switch, https://www.sdxcentral.com/cloud/open-source/definitions/what-is-open-vswitch/
[7] Containers, https://medium.freecodecamp.com/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b#.o0t1prt6b
[8] Intel Snap Framework, http://snap-telemetry.io/
[9] Apache Kafka, http://kafka.apache.org/documentation.html
[10] Apache ZooKeeper, https://zookeeper.apache.org/
[11] Grafana, Grafana, http://grafana.org/
[12] Visjs, http://visjs.org/
[13] Node.js, https://nodejs.org/en/

SaaS OverCloud 기술 문서

작성기관: 광주과학기술원
작성년월: 2016/10