Indian Journal of Science and Technology, *Vol 12(4), DOI: 10.17485/ijst/2019/v12i4/140768, January 2019*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

AAAS - Framework in Large Virtualized Environment

M. B. Bharath[1] and D. V. Ashoka[2]

[1]Solution Architecture, Dell EMC Software and Service Bangalore - 560048, Karnataka, India; Bharath.Basavarajappa@emc.com
[2]Department of Information Science and Engineering, JSS Academy of Technical Education, Bangalore - 560060, Karnataka, India; dr.ashok_research@hotmail.com

Abstract

Objective: To define template architecture for large virtualized environment automation ecosystem. **Methods/Statistical Analysis:** This study proposes self-sustaining, highly scalable automation architecture with multiple services components supporting each other's to deliver a reliable and scalable service in self-contained virtualized environment. This new proposed template architecture has been evaluated by building Proof of concept built using few readily available tools and few property application developed for this project. Finally we evaluated the architecture with varying load of storage, SAN networking and Backup and recovery workflow tasks. Stability, sustainability and self-load balancing capabilities are measure over time, by injecting dynamic work task to this framework. **Findings:** Large virtualized environment gives unlimited access to compute and storage resource on demand. One of the key enabler for this functionality is the robust automation framework. Due to limited technical expertise and lack any open source standard shinder migration of any existing mid-size virtualized IT assets into fully automated eco-system. This study tries to address this challenge by proposing new template open-ended architecture. This takes care of any organizational scalability, security, and compliance and maintainability issues. **Application/Improvements:** The proposed architecture can be used for implementing highly automated virtualized ecosystem in large virtualized environment.

Keywords: Automation, Integrated Automation Framework, Orchestration, Scalable Architecture, Virtualized Environment

1. Introduction

Virtualization was evolutionary change that altered the landscape of how organizations visualize their IT assets. Concepts like Infrastructure as-a-Service (IaaS) gives better resource utilization and reduce the operational cost of managing and maintaining IT infrastructures. There are good amount of highly automated large commercial public cloud providers like Amazon EC2[1], Microsoft Azure, Google cloud platform, VmWarevCloudAir, Rackspace are few such examples, which provides IaaSservice. These solutions address the commercial public cloud space, but there is a gap in terms of medium to large scale organization which cannot move to public cloud due to policy constrains and security risk, like large financial institutes, defense and defense manufactures, public health care institutes etc. Most of them are using virtualization but limited by their ability to move to the highly automated IaaS space. Because of lack of technology and limited resources, which hinder their efforts. We are trying to address this unexplored space with template architecture. The large scale virtualization orchestration involves the creation, management and manipulation of resources, i.e., of the compute, storage and network, in order to realize user requests in an environment and or to realize operational objectives of the service provider. User requests are driven by the service abstraction and service logic that any environment exposes to them. Any automation and orchestration built should address issues like bulk concurrent user request, organizational policy enforcement

*Author for correspondence

and security and compliance assessment, while servicing this request.

However, existing techniques for orchestration are rudimentary to meet the new very large virtualized environment. Its ability to scale and sustain is hamper by the multiple functionality and features it support. This paper proposes a new alternative model with capability of self-monitoring, scalable, secure and compliance driven engine which can scale horizontally along with orchestrator engine. Before that we will go through some of the related research work done in this area. Not all of them related to virtual environment, but they try to address similar issues in different domain. Virtualization automation and Service-oriented paradigms in industrial automation. In[2] proposed talk about intelligent device networking based on service-oriented high-level protocols in Service Infrastructure for Real-Time Embedded Networked Applications (SIRENA) project. In have worked on the Learning Automata (LA)-based QoS (LAQ) framework[3], which address the challenges and demand of various cloud applications, hence making efficient use the computing resource. This proposed framework also make sure that the on-demand request are serviced with minimum service level as prescribed by the Service Level Agreement (SLA). COOLAID[4] proposes a data-centric network configuration management. COOLAID manages router configurations and adopts the relational data model and Data log style query language. There are several related frameworks proposed for management and orchestration for large scale systems. Autopilot[5] is a data center software management infrastructure from Microsoft for automating software provisioning, monitoring and deployment. It has repair actions to deal with faulty software and hardware. Its periodic repair procedures maintain weak consistency between the provisioning data repository and the deployed software code. Similarly in open source community Puppet[6] is configuration management tool which can orchestrate the datacenter administrative task through easy declarative statements. On similar line Chef[7] is the configuration management tools written in Ruby, which uses the domain specific language (DSL) for the writing recipes (aka system configuration details). This is wildly used in DevOp process to integrate with cloud service providers like Amazon EC2[1], Google Cloud Platform, Oracle Cloud, OpenStack, SoftLayer, Microsoft Azure and Rackspace.

2. Materials and Methods

2.1 Limitation of Existing System

Most of the current day automation solutions are around orchestrator or running a group of job scripts. This method helps things done in quick way and gets the job done. But it's not sustainable on a long run. After couple of quarters in production, maintenance and support issues crop up. Along with this, as more devices are added to automation system, scalability becomes a limitation of the system. One of the most difficult part to handle with this solution is the, issue related to security and compliance (this is critical aspect in managed service business) and not easy to plug-in this to any of the legacy system on the brownfield solutions. Along with this any organization level policy enforcement is a challenge, since system is not build to accommodate these needs.

In summary following is the few critical limitation of the existing system

- Automations are done in silos
- Not easy to monitor and scale
- Policy enforcement and compliance adherence.
- Availability and Predictability of automation system
- No plug & play framework for automation as service
- Most of the automation are specific to domain and fails in long run
- Most of the current automation solution are Orchestrator centric
- Not maintainable in long run

So there is a need for the radical new solution which take into consideration of security, compliance, plug and play etc, in to consideration at the conceptualization stage rather than dealing with them as a afterthought process. Next section describes one such solution.

2.2 Domain Model

This new proposed model has four base components services as listed below.

- Infra as Service
- Management System

- Backbone System
- End point Adapters

Each of these base components is a service group,which logically group the related components. Where Management System includes components like self-service portal, Health check dashboard and Policy engine portal. All users facing Graphical User Interface (GUI) is grouped under this component. In other words these are interface component with automation eco system from end user perspective. Whereas the Infra as Service include core of the automation eco system, that is orchestrator and load balancer. These two forms the foundation of the framework on top of which rest of the eco system is build. Most of the other services coordinate and support this base component to achieve sustainable eco system. Figure 1 shows these details in abstract representation. Backbone and End point adapters provide support functionaries like policy enforcement, Monitoring internal resource utilization and health, also provide API connectivity to external world through adapters. All these components are represented in Figure 1.

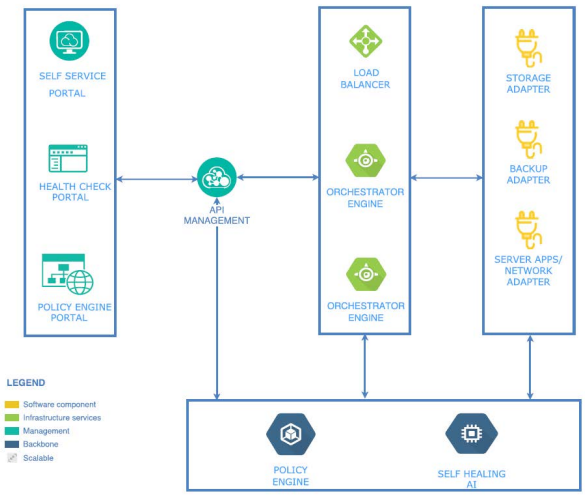

Figure 1. Four main services of Data Model and their logical relationship.

2.3 Proposed Solution

The new proposal for scalable automation and orchestration platform for data center operations, which is a self-sustaining, highly scalable architecture with multiple services supporting each other's to deliver a reliable and scalable service. This is an open ended architecture with eight supporting components (as listed). This is a template framework for which, one can easily identify the products readily available in the market to build this setup. Here are the eight independent service components in this eco system. Figure 2 represents this template architecture with their logical interconnection between eight services. Following section briefly describes these eight components in detail:

- Orchestrator Engine.
- Integration Gateway.
- Controller Engine.
- Policy Engine and Inventory database.
- Health checks Engine.
- Security and complacence Engine.
- Common End point Adapters.
- Load Balancer.

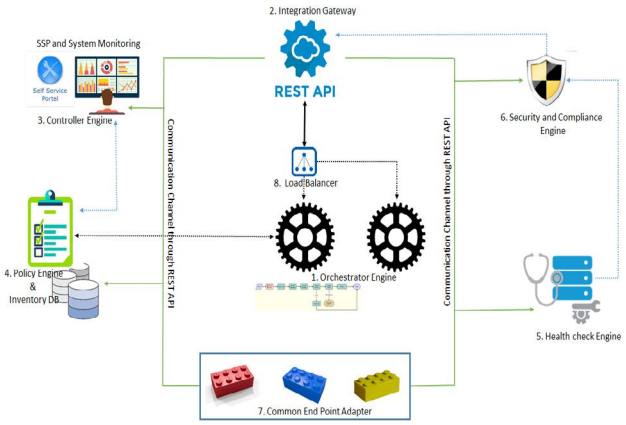

Figure 2. AAAS Automation architecture.

2.3.1 Orchestrator Engine

Orchestrator is the core component of the automation platform, which will host the business workflows and other logic to execute any datacenter admin or operational tasks. Basic recommendation is to have two Orchestrator Virtual Machines (VMs), one as a primary and other as a secondary - to provide basic High Availability (HA) service. Based on the demand, one can horizontally scale this by adding additional orchestrator engine(s) as need arises. Its primary task is to execute and orchestrate the automation jobs submitted by end users.

2.3.2 Integration Gateway

This component provides north bound interface access to architecture. Any IT Service Management (ITMS) tools

or ticketing tools can be coupled with the automation framework through REST API service provided by the integration gateway (or using Self-service portal embedded within Controller Engine service).

This service will also host load balancer to pass on the incoming jobs/tasks to available Orchestrator engine(s). These also update the results (success or failure details) back to its original requestor. All the service entry point to Integration gateway is defined in terms of REST API endpoint to allow easy and flexible integration with any external third party tools.

2.3.3 Controller Engine

Controller Engine provide following two primary service.

1. Self-service portal- gives direct invocation access to automaton workflows
2. Health check reports- gives the utilization and health status of the automation assets and eco system elements.

2.3.4 Policy Engine and Inventory Database

This service provides an ability to add any customer specific polies like Max size of a file system on NAS array or Default backup policy for any filesystem etc. Most of these policies will be owned by the individual account or customer and they can change them over a period of time. Most of these policies are global in nature. But next level of granularity can be provided by creating group/tagged elements and applying policy on that group. Most of these policies will be store in database which also hosts the customer Inventory data along with device admin credentials. Array level utilization metrics will also be stored in this database, which will be used to find the least utilized array for next provisioning request. Utilization data is feed by the Health check Engine collector, which collects hourly capacity utilization data for most of the inventory device entry in the database. Orchestration Engine queries the Inventory database to find array/device for allocation request. This request will be honored by the inventory database service using policy engine as selection logic. All the communication between the services like orchestration engine, inventory database and policy engine happens through REST API.

2.3.5 Health Check Engine

This is one of the critical components in existing architecture, which check the availability of inventory devices with their current utilization numbers. Health check engine has a collector which can talk to each device/array through API layer and collects its availability and utilization details. It also shows the basic discovery of each of those devices/arrays. Along with this it looks at the other service engine in this architecture to give quick health check of the eco system. It has built in event correlation & self-recovery of some of predefines failures. Like restarting service or API endpoint tomcat etc. Event's data and availability details will be shared with controller engine health check reports. These reports show the current running state of the automation framework and its utilization details.

2.3.6 Security and compliance Engine

This component keeps track of all the security logs and auditing details for the other entire service engine in this architecture. It provides both syslog integration and SNMP trap receiver service of any security or audit events. This also hosts some of the reporting capability for security compliance service.

2.3.7 Common End-Point-Adapter

These are interface service which will communicate with any device in its native API layer. These are primary interface to talk to device/external components, which enable automation capabilities in this framework.

2.3.8 Load Balancer

This will distribute the incoming task to multiple orchestrator engine(s) based on its availability. This will also take care of the "High Availability" aspect of automation eco system.

3. Results and Discussion

3.1 Prototype Implementation

This template architecture has been implemented using some of the off the self-products readily available (open source) in market and few components are proprietary

code developed for this project. Our primary orchestration engine is from VMware vRealize Orchestrator[8] (vRO). Integration gateway which allows API layer access to the framework is written in Python using Flask library module. Similarly Load Balancer service is java based code which distribute the task to multiple orchestrator engine(s). The choice of java is primarily because vRO has few native java API for communication and task assignment, which simplify the interface coding. Health check and security compliance part of this framework is done by EMC Storage Resource Monitor[9] product. Policy engine and inventory db is built using PostgreSQL server with static HTML page, which can be accessed through REST API. Figure 3 shows the high level implementation diagram of this prototype.

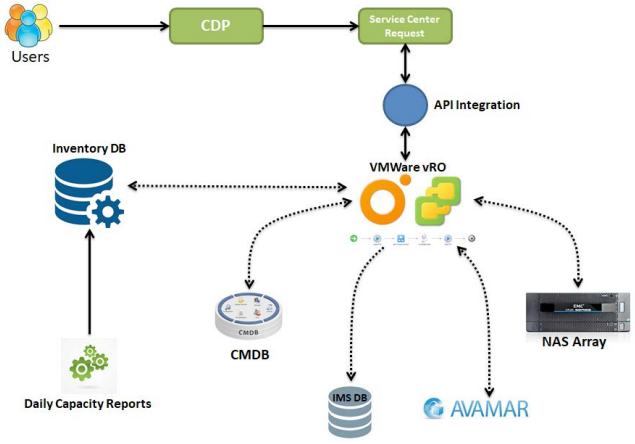

Figure 3. AAAS Prototype implementation details (abstract level).

3.2 Preliminary Evaluation

This section will explain two sample workflow and its basic flow and life cycle (of automation task or job) in this new framework. This shows two use-cases as example to explain the complete flow. It starts with explaining the generic flow of job entry to the eco system through REST API request with required input parameters to run the automaton. Any automation task go through following seven stage or process in this ecosystem as shown in Figure 4. Each of this stage is explained briefly in the following section.

- Load Balancer will automatically assign it to one of the free orchestrator engine, based on the current load.
- Task/Job input parameters are validated against the policies and threshold to enforce any organization level policy.
- Communication end point adapter select the appropriate device/Configuration Item (CI) to complete this task/job. This selection will be assisted by the utilization collector and inventory db.
- Health check collector will ensure required device/CI is available and accessible through API interface.
- Finally required changes are executed on the target device/CI.
- Job status is actively monitored if it fails to execute any one of the required change(s), complete task will be rolled back, by invoking recovery steps.
- If task/Job complete with success it will be reported back to Integration gateway service, which intern update to its caller (ITSM tool or Self Service Portal).

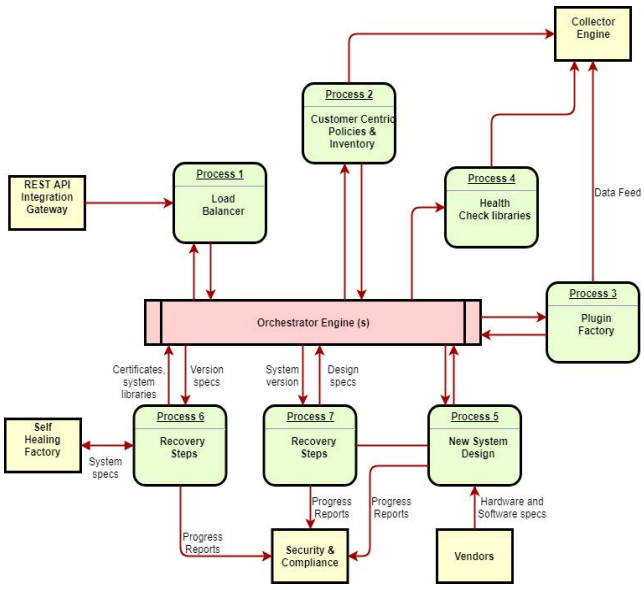

Figure 4. Automation task life cycle within eco system.

Let's consider the use-case of configuring the backup on Avamar server. This automation use-case has 6 common stages and 3 specific additional stages, if that machine is a virtual server. Each of this stage/process is executed within the orchestrator as workflow step and passing on the result to next subsequent stage finally complete the task in case of success, or role backing the tasks done in

case of failure. It's important to make sure that any automation as atomic state, either it complete in its totality or nothing will be done. This is critical to keep the production setup in the consistent state.

Finally it updates the results back to its requestor. Figure 5 explains the different stages in the use-case workflow. Similarly, Figure 6 shows the workflow stage for any Configuration Management DataBase (CMDB) update. First it checks if that device/CI is available in CMDB, if so it updates the details if not it will try to create that entry and then update the details. Figure 7 shows the sample JSON input object sent to this CMDB update request.

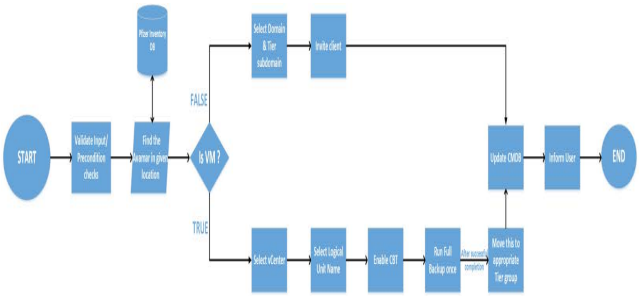

Figure 5. Configuring backup for server automation use-case flow chart.

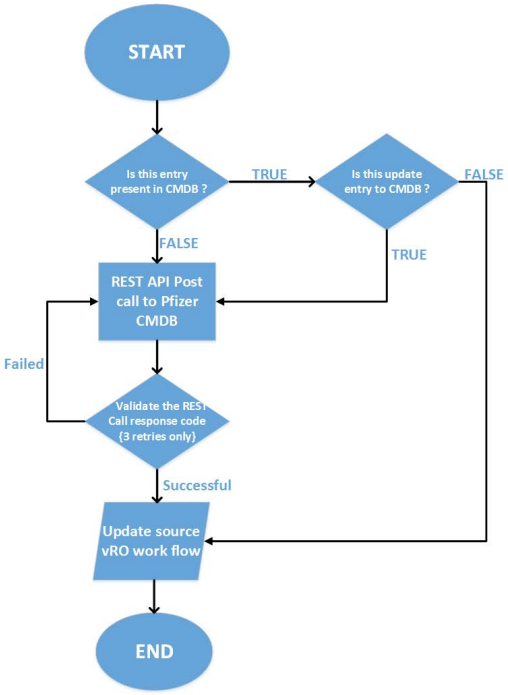

Figure 6. Updating the CMDB after automation task completion use-case.

```
/cmdb/brs/v1/update  [POST]  ->  Translates to updating CMDB details
{
    "CI Name" : USNESQ5601
    "CI ID" : SC2927426
    "TypeofChange" : INFRASTRUCTURE HARDWARE
    "RFCType" : HARWARE
    "RFCSubtype" : CMDB UPDATE ONLY
    "WorkflowType" : PROD ONLY
    "RFCCategory" : PRE-APPROVED
    "RequesterName" : Person Requested orignal ticket
    "CIFieldAttribute" : Backup Master Server
    "JournalActions" : Planning has been completed
    "StartTime" : Start time of this change
    "EndTime" : End time of this change
    "CompletionCode" : Successful
    "CompletionComments" : completed successfully
}
```

Figure 7. Sample JSON input for the updating CMDB use case.

Figure 8. Top 30 workflows and their execution frequency count.

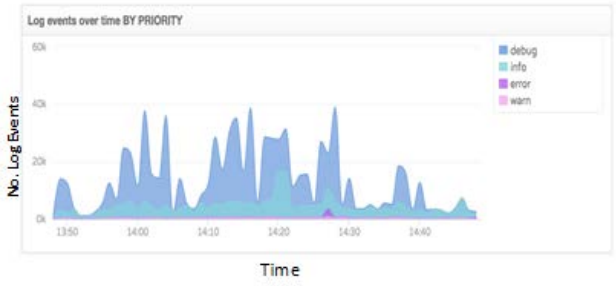

Figure 9. Events grouped by severity.

3.3 Validation and Results

The prototype of the proposed framework is implemented using 54 workflows, which include both storage and backup product related use-cases along with SAN Networking admin tasks. Most of these workflows are related to operational task like provisioning, adding or deleting backup configuration, creating filesystem, giving access, mapping to server etc. These vRO workflows

are trigged by ServiceNow tickets configured using integration gateway service. Once the system is up and running health monitoring and success ratio is captured in VmWare Log Insight, which is system health monitoring tool. Here are the few sample report dashboard related to vRO workflows and their relative state as the system runs for couple of weeks to monitor the behavior of this new architecture. Figure 8 shows that top 30 workflows with their execution frequency. Similarly Figure 9 shows the different logging event over a time group by their priority (or severity) level. Figure 10 show the relative quantity of different events grouped by their type.

This section shows the self-stabilization of the system over period of type. Figure 11 and Figure 12 shows, how the number of event (also unique events) and their relative quantity decrease over time period. This provides positive result that system self-correction feature work as expected, making it more suitable for log running self-contained automation workload.

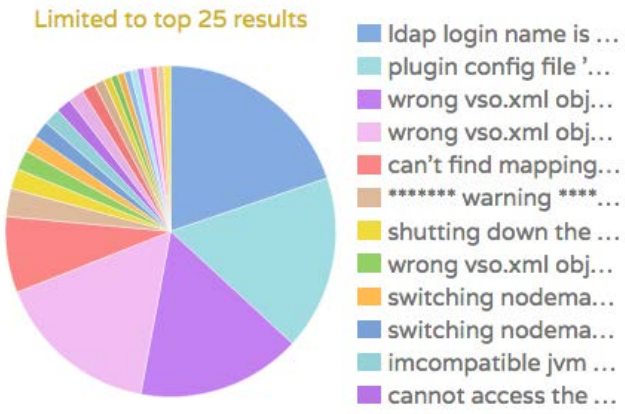

Figure 10. Event group by their type.

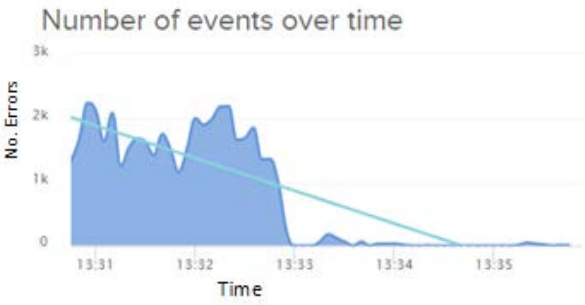

Figure 11. Event group by their type.

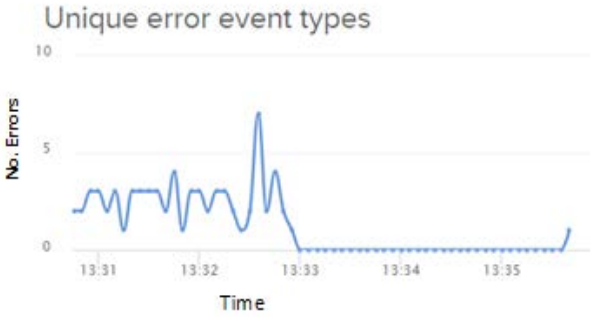

Figure 12. Event group by their type.

4. Conclusion

Virtualization is an omni present technology in most of the commercial organizations. It allows the IT assets to be managed as a commodity. Even then migrating to next level of hyper automated eco system is a challenge. This is primarily because of limited expertise and technology in small and medium scale organization. Our proposal will help them to migrate to highly automated eco system, which can be easily scaled up on demand. AAAS is a first attempt in adopting a scalable and self-sustaining approach that combines the different aspect of production ready automation eco system. This template allows easy adaptation in wider range of scenarios as this is a generic open ended architecture. We were able to successfully implement this architecture using combination of readily available products and few self-developed service. Hence this is a relatively a viable option to consider,for anyone looking for large scale virtualization automation system, with varying dynamic load characteristic.

5. Reference

1. Amazon Elastic Compute Cloud (Amazon EC2). Available from: http://aws.amazon.com/ec2/
2. Jammes F, Smit H. Service-oriented architectures for devices- the SIRENA view. International Conference on Industrial Informatics; 2005. p. 140–7. https://doi.org/10.1109/INDIN.2005.1560366
3. Misra S. Learning automata-based QoS framework for cloud IaaS. IEEE Transactions on Network and Service Management. 2014; 11(1):15–24. https://doi.org/10.1109/TNSM.2014.011614.130429

4. Chen X, Mao Y, Mao ZM, Van der Merwe J. Declarative configuration management for complex and dynamic networks. Proceedings of 6th International Conference on Emerging Network Experiments and Technologies (CoNEXT); 2010. p. 1–12. https://doi.org/10.1145/1921168.1921176
5. Isard M. Autopilot: Automatic data center management. ACM SIGOPS Operating Systems Review. 2007; 41(2):60–7. https://doi.org/10.1145/1243418.1243426
6. Puppet: A data center automation solution. Available from: http://www.puppetlabs.com/
7. Chef: Automation for web-scale IT. Available from: http://chef.io
8. vRealize Orchestrator. Available from: https://www.vmware.com/in/products/vrealize-orchestrator.html
9. Dell EMC Storage Resource Manager. Available from: https://www.emc.com/collateral/data-sheet/h12350-storage-resource-management-suite-ds.pdf