International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 05 | May 2020 www.irjet.net p-ISSN: 2395-0072

Server Firmware Management using DMTF Redfish REST API's

Muteeb Akram Nawaz[1], Veena Gadad[2]

[1]*Student, Dept. of Computer Science and Engineering, R V College of Engineering, Karnataka, India*
[2]*Professor, Dept. of Computer Science and Engineering, R V College of Engineering, Karnataka, India*

Abstract - *Periodic firmware updates are needed to improve the functionality of servers, fix any existing software deficiency, and to provide security from vulnerabilities. This paper shows a simple and best approach for firmware upgrades of server components with an industry-standard protocol Distributed Management Task Force (DMTF) Redfish and REST API. DMTF is a non-profit organization that provides industry standards for emerging infrastructures such as servers, network, cloud, storage, and virtualization. Redfish is one of the standard technologies offered by DMTF for server management which leverages common web services and Internet standards such as HTTP to expose information directly to the modern tool chain such as REST APIs. API is a set of functions and procedures that allow one application to access the features of other applications whereas REpresentational State Transfer (REST) is a software architectural style that defines a similar set of rules or guidelines to build a client-server API for web services*

Key Words: **Firmware, Server, BMC, REST APIs and DMTF's Redfish**

1.INTRODUCTION

Data centers are the location where huge computational power, networking devices are equipped for processing, sharing, storing large amounts of data. These data centers play a critical role in running government organizations, telecommunication industries, businesses, enterprises, and many more. Data centers consist of several generations of components from legacy main-frame systems to advanced rack-based, modular blade servers. Thousands of such servers make up a data center. Servers have complex hardware and software structures. Managing such a diversity of servers and components is a serious problem for the server industry. To solve this problem, the Distributed Management Task Force (DMTF) has come up with Redfish.

Redfish is a set of standards that deliver an industry-standard protocol by using RESTful API architecture for easy management of server systems such as storage, networking, and firmware management. Many server industries and their customers are pushing this technology as it provides a standard common platform server management. It aims to provide server vendors and customers a uniform platform for easy management of servers.

Firmware is a set of instructions or software program that is programmed on hardware components of a server. It provides the necessary instructions for how the component communicates with the other components in the server. Firmware is typically stored in the flash ROM of a component. Because flash ROM can be erased and rewritten since it is a type of flash memory. Firmware updates bring some changes in the instruction or program, which are essential for corresponding component performance and feature improvements as well as to fix the software deficiency for better security.

The following are the main objectives of this paper:

1. To promote industry standard technologies in the domain of Server Management by using DMTF's Redfish.
2. To provide a simple, efficient, and common platform for server vendors and their customers for server management.
3. To enhance the current Firmware Management in Server by providing modern tool chains such as REST API that adheres to Redfish Standard.

1.2 Organization of the paper

The initial part of the paper gives a brief introduction and explanation of Servers in the Data center and diversity of components present in the server, an industry standard protocol called Redfish provided by DMTF organization, and an open source REST APIs to create Redfish APIs for firmware management. The rest of the paper provides a detailed explanation of designing APIs for discovering firmware components, updating firmware components and monitoring status of firmware components.

1.2 Literature Review

At its simplest, a data center is a physical building used by companies to store sensitive software and records. The architecture of a data center is focused on a network of computing and storage resources that enables the delivery of shared applications and data[1]. The rapid growth of cloud computing, both in terms of the spectrum and volume of cloud workloads, necessitates re-visiting the traditional rack-mountable servers based datacenter design.

The paper provides a set of modification to be made on existing rack server for making it ready for current cloud industry[2]. A client-server architecture for creating, managing API's for managing components of the server. The

Rest API's provides security, robust, flexible to user and more [3].

Data center managers, standardization bodies and hardware / software manufacturers are joining forces to develop and promote Redfish as the main hardware management standard for data centers and beyond. The authors hope that this article will be used as a starting point for understanding how Redfish and its extensions are being pursued as the central management framework for next generation data centers. The white paper on Redfish gives a brief about all the API's provided by the organization[4 - 5].

Dell Computer Company used Redfish Schema and developed RESTful API's and deploy them on Dell PowerEdge Server. A end-to-end implementation and deployment of Redfish on Server was proposed[6]. A set of standard API design rules, drawn primarily from best practices that stick close to the Web's REST architectural style. Along with rules for URI design and HTTP use, one will learn guidelines for media types and representational forms. The steps needed to be followed for designing, implementing and deploying firmware for hardware components of the server[8]. The Baseboard Management Controller (BMC) is a specialized software processor that controls the physical state of a computer, network server or other hardware unit using a sensor and interacts with the system administrator via an independent interface[9].

2. Methodology

2.1 System Setup

Each server has a dedicated component that can monitor the host and other components present in the server called Baseboard Management Controller (BMC). It is usually an ASIC – Application Specific Integrated Controller (aka SoC) that sits on the motherboard of the server. BMC has its own processing power, and one can run an operating system, applications and more on it. Original Equipment Manufacturer (OEM) vendors provide a dashboard/console to monitor the server and its components from the console itself. The motherboard of the server is designed such that BMC has multiple connections to the host system and other firmware components of the server. OEM vendors leverage these connections to monitor the status of components present in that server. BMC can also control the power cycle of the server, that is it can shut down and reboot the host system. It also provides remote access to host's via virtual Kernel-based Virtual Machine (KVM) so engineers/administrators need not visit the server. In short, BMC is a small computer that sits on the main computer (host) which can monitor components of the main computer.

It has a dedicated network connection to host via Network Interface Card (NIC) most vendors mainly use USB - NIC as shown in Fig 1. USB - NIC is a high speed wireless network card that uses the USB port of the host system to provide a network connectivity to BMC.

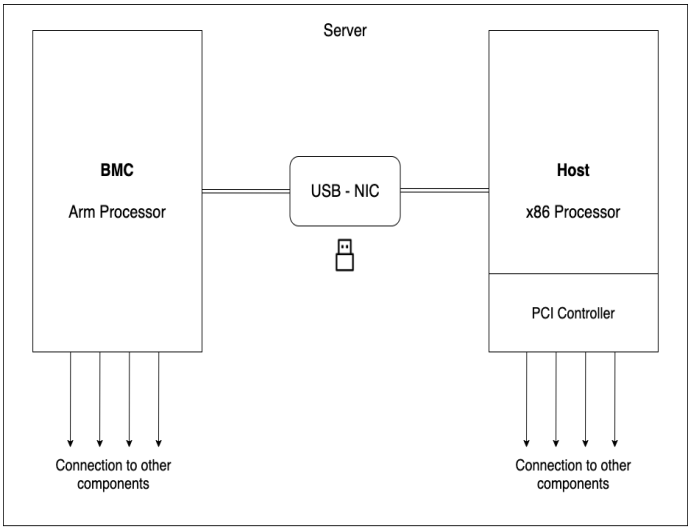

Fig – 1: BMC connected to Host via dedicated USB-NIC

2.2 System Design

To provide a REST APIs support for firmware management the system starts a lightweight python web server at the host system such as flask. Flask is a micro web framework written in Python. This micro web framework is perfect for building REST APIs. Once the host operating system boots up the web server can run like a demon process in the background. The web server needs to run on the USB-NIC interface IP of the host system.

Therefore, BMC will become the client to this web server called BMC Agent as shown in Fig 2. BMC can make an API request to the web server and the web server can return the response back to BMC. Now, for firmware upgrades of the component present in the server this project designs API that adheres to Redfish standard.

This paper proposes the following APIs for firmware management of the server.

1. To provide an API to discover components, present in the server.
2. To provide an API that pulls the new firmware image and flash it on the component.
3. To provide an API to monitor the status firmware flash.

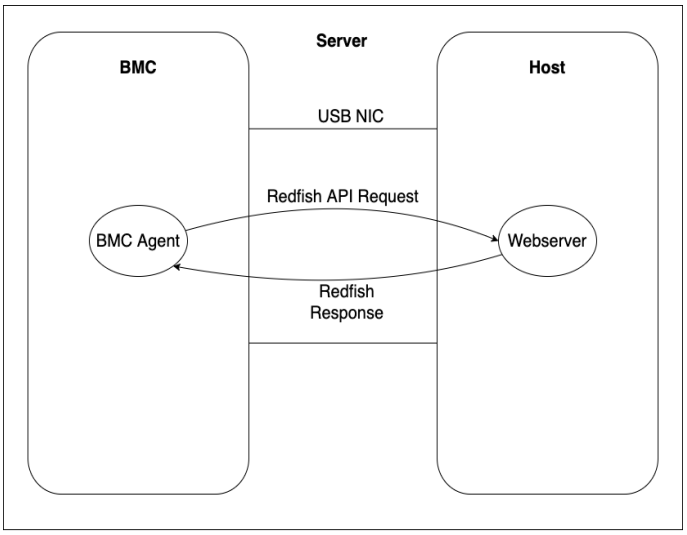

Fig – 2: System design

2.2.1 Discover Firmware Components

Many server vendors provide a BMC console for the server through which one can mount an Operating System (OS) to host usually it is virtual KVM mapping of OS to host. For Firmware management of components, the system first needs to discover the components present in the server. The most common approach for discovering firmware components is while booting an OS. OS needs to know all the components/resources present in the server to be functional. One can run a python script while booting the host to collect the information about the component, its current running version, health and etc. from the SMBIOS table and cache the results into a database called firmware inventory in the host system. Therefore, the system has a full firmware inventory that is the system is aware of installed hardware components and also has information about every component present in that server. The Firmware Inventory database is passed to the web server.

When a client makes a Redfish firmware inventory API request to the web server. The web server can use the inventory database generated during the boot-up process and send a response to the client request. Client can also request the information about a particular component by calling Redfish firmware component API web server pulls the information of the requested component from the firmware inventory database and sends the Redfish response back to the client as shown in Figure 3.

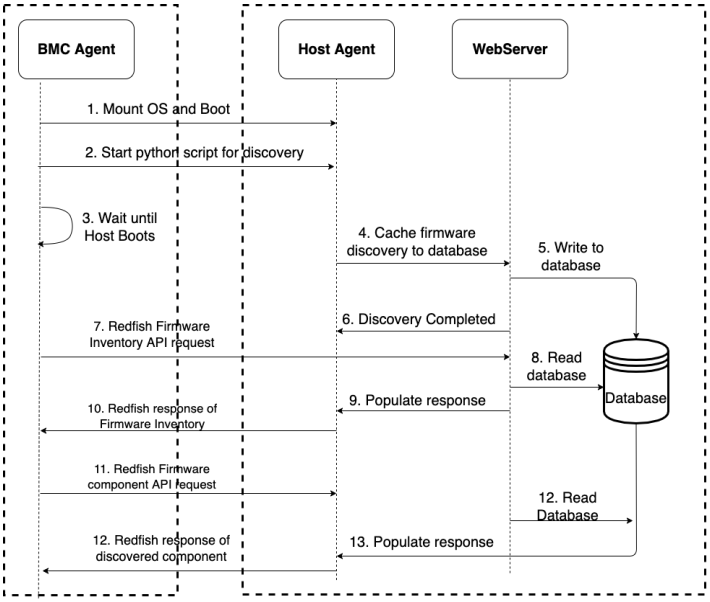

Fig – 3: Sequence diagram for the discovery of components in Server

2.2.2 Firmware Update Components

Firmware upgrade is a two-step process. First, update the component with a new firmware image and second is to activate the component by flashing the new firmware image and reboot the component. Most of the server vendors bundle the new firmware image with the operating system or they place the firmware image in a remote server and the image is pulled from the remote server by an HTTP request or so during firmware update. Server vendors can design the parameters of the API as per the needs of their customers to facilitate firmware upgrade. To make firmware updates seamless the upgrade process needs to be done in the background as firmware upgrades take time. Redfish provides an Update Service API which is an asynchronous HTTP POST request to run firmware upgrades in the background.

When the client makes a Redfish update API request to the web server, the API parameters are verified, new firmware image is pulled and new Redfish task is started with a unique Task ID and the firmware upgraded of the component is started in the background like a daemon process by calling the corresponding component hook for firmware upgrade. This hook is responsible for updating the status database as shown in Fig 4. It is up to server vendors how they upgrade the firmware internally when the client sends an update request.

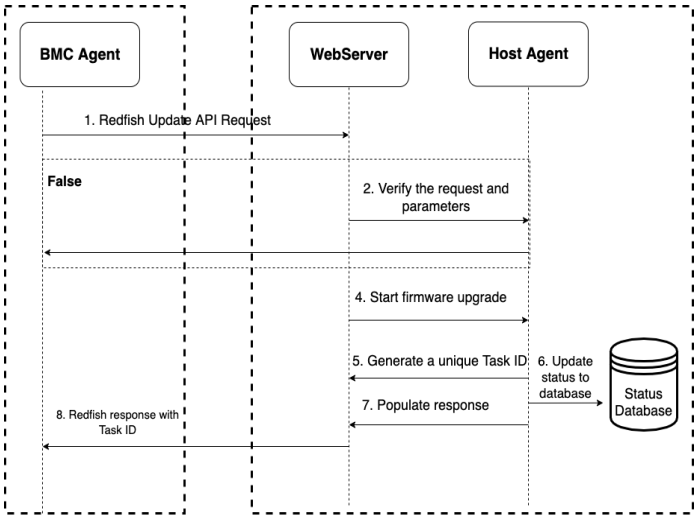

Fig – 4: Sequence diagram for the firmware update of components

2.2.3 Monitor Firmware Upgrade

As mentioned in section 2.3, the firmware upgrade is a background process in order to view the status of the upgrade Redfish provides another API called Task Service. When the client makes Redfish Task Service API for a particular task with its Task ID. The status of that task is read from the status database and a JSON response is sent to the client as shown in Fig 5.

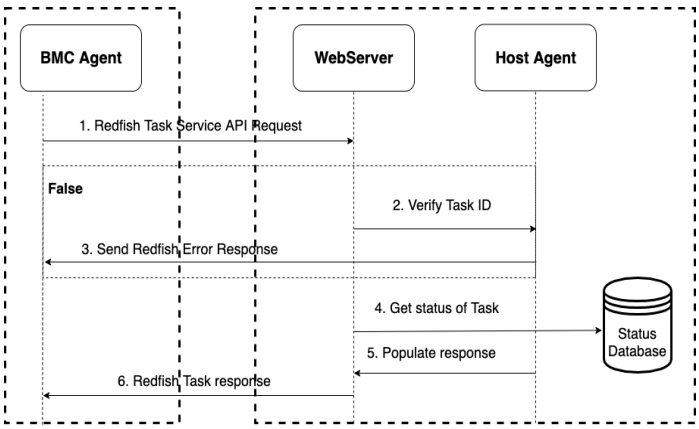

Fig – 5: Sequence diagram for monitoring firmware upgrade of components

3. CONCLUSIONS

This paper shows a simple yet an easy way for firmware management using Redfish REST APIs. The proposed system successfully provides firmware inventory of the server that is all the components present in the server. After firmware inventory, the client can select the firmware to upgrade or upgrade all the components by making a valid request to the web server. As the firmware upgrade is a background process the system provides another API to monitor the status of the firmware upgrade. To any system, there is always room for improvisation. The proposed system mainly focuses on firmware management of the standalone server. The system needs to be extended to provide firmware management of the entire chassis.

ACKNOWLEDGEMENT

Any accomplishment, whether scholastic or otherwise, is not based solely on individual actions, but on scholars, elders and friends' support, encouragement and cooperation. A variety of personalities have helped us in carrying out this project work within their own capacities. I want to take this opportunity to thank every one of them. I express our sincere gratitude to our mentor Prof. Veena Gadad, Professor at the Department of Computer Science and Engineering, R.V.C.E, Bengaluru for his skillful guidance, frequent source of motivation and support during this project. We would also like to commend Dr. Ramakanth, Head of Department, Computer Science and Engineering, R.V.C.E, Bengaluru, for his valuable suggestions and expert advice.

REFERENCES

[1] D. H. Georgios Karagiannis, "Cloud Central Office Reference Architectural Framework", 2018

[2] C.-S. Li et al., "Composable Architecture for Rack Scale Big Data Computing", Future Generation Computer Systems, vol. 67, pp. 180-93, 2017.

[3] J. Baker, S. Savino, 'The role of client/server computing technology in the management of global enterprises', 2002, Innovation in Technology Management – IEEE

[4] Glauco Gonçalves, Daniel Rosendo, Leylane Ferreira, Guto Leoni Santos, 'A Standard to Rule Them All: Redfish', 04 September 2019, IEEE Communications Standards Magazine.

[5] 'Redfish Composability White Paper', 2018, DMTF Redfish DSP2086,

[6] S. S. A. Jonas Werner, P. S. S. E. P. Raveendra Reddy and S. P. M. Paul Rubin, 'Implementation of the DMTF Redfish API on Dell PowerEdge Servers', 2016,

[7] Mark Masse, 'REST API Design Rulebook: Designing Consistent RESTful Web Service', "O'Reilly Media, Inc.", 18-Oct-2011

[8] Veit B. Kleeberger, Stefan Rutkowski, Ruth Coppens, 'Design & verification of automotive SoC firmware', 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)

[9] Jing Lin, BO Xie, Jian Xu, Binqi Zhang, 'Server management using a baseboard management controller to establish a wireless network'