January 28–29, 2014 San Jose
Infrastructure Management

• Data Analysis in the Data Center
• Infrastructure Management Requirements
• A Way Forward

Jeff Carr
Data Center Manager, Product Development
Data Analyst, System Administrator, Database Programmer
DATA ANALYSIS IN THE DATA CENTER
Data Center Case Study ~50,000 square foot

1. Independent Software
   - Asset Tracking/DCIM: Rackwise
   - Modbus Power Monitoring: Aperture
   - Infrastructure Management: Infrastructure Manager
   - Motorola Scanning software
   - Manufacturer Specific Firmware Management: Servertech, Raritan, APC, etc
   - Remote Management: UCS, ILO, DRAC, Various Blade Center Manager Software
   - Server Management: Altiris
   - Patch Management: Shavlik
   - Proprietary Custom Software

2. Software Sharing Information
   - Ticketing/Discovery: Remedy
   - Proprietary Configuration Management Database
Data center processes requiring data sharing

1. Asset/Audit Management
2. Change Management
3. Workflow
4. Incident Management
5. Resource Management
6. Capacity Planning
7. Cost/Depreciation Tracking and Assignment
8. Server Management
Advantages to a common API

1. Consistency
   - A common method makes it easy to store data in common or compatible locations

2. Accuracy
   - Single input of information with fewer conversions makes for more accurate data

3. Analysis
   - Analysis that is extremely difficult like predictive capacity management is simplified

4. Efficiency
   - More comprehensive and timely access to sensor data and cooling controls

5. Flexibility
   - Compatible software from a variety of vendors, and vendor lock in is limited greatly

6. Security
   - A flexible API with role based permissions that can be granted or revoked as needed
Disadvantages to a common API

1. Yet another management layer
   - If the management layer isn’t robust enough to replace existing tools, it will simply add yet another tool, another unused API and another protocol

2. Security
   - The API will be a large attack surface for infrastructure information and/or control

3. Flexibility
   - Asking OCP certified hardware to work with the API could slow down development if the API wasn’t developed quickly enough
INFRASTRUCTURE MANAGEMENT REQUIREMENTS
What makes RESTful API?

• The term RESTful has come to really mean Restful HTTP:
  - Uses standard http transport and methods, and stateless communication
  - Encodes data using JSON or XML
  - User can execute using normal OS http APIs (eg curl…) – no client code required

• Early RESTful APIs
  - Often just re-implemented commands using http transport
  - URI path, query string, http headers, body data was used various ad-hoc ways

• More recent REST APIs (“V2”) embrace several ‘commonly accepted’ RESTful API principles and best practices---ex: OpenStack V2 API
  - URI points to the resource or collection--not the action or command.
  - Uses IDs in URI to identify resources or collections (eg sleds, fans, servers)
  - Uses links to associated resources (eg serverNode to sled, dependent PSUs…)
  - Uses standard http methods appropriately (http GET, POST, PUT, CREATE, DELETE)
  - Supports multiple data representations (json, xml)
  - Uses HTTP headers to negotiate capabilities or program versions
Existing state of agent vs agentless management

1. Reliability
   - Agentless management is always installed, there is almost always exceptions with installation of agents
   - Agentless management can monitor internal computer systems invisibly and cleanly
   - Agentless management is normally on and functioning when the system is not.

2. Flexibility
   - Agentless management can rarely be customized, upgraded, or expanded
   - Agentless management is often limited when not using vendor specific hardware

3. Security
   - Agentless management is resistant to attacks from its host
   - Multiple external attacks against BMC’s were discovered in 2013
   - Most BMCs must store passwords in clear text
   - Most agentless management has passwords that cannot be easily be changed en mass, often leading to duplicate passwords unchanged through staff changes
Multiple management protocols will be required

1. **IPMI**
   - IPMI is commonly deployed across multiple vendors
   - BMCs will remain in common use in the near term due to advantages in reliability

2. **SNMP**
   - SNMP is commonly deployed in power and monitoring systems that will remain in place for much of the life of the data center

3. **Modbus**
   - Craah and Crac units are commonly monitored and controlled with Modbus

4. **WBEM**
   - Microsoft WMI is an implementation of WBEM that will be widely deployed for the foreseeable future
A WAY FORWARD
Options for a common API

1. Create a new API from scratch
   - No problems with backwards compatibility
   - Complete control of the API
   - Extremely difficult to do correctly out of the box

2. Adopt an existing open source API
   - Experience gained since the creation of the API is immediately available
   - Must work with existing standards body
   - Inherit limitations due to backwards compatibility

3. Fork an existing API
   - Benefits of adopting with some of the flexibility of creating your own
   - Possible merge headaches in the future

4. Open source an existing proprietary API
   - Benefits of forking, but without the problems and benefits of an existing community
Rack Management

- Models for Rack Management in Use Today
- Example of Rack Management: Dell DCS G5
- REST API Analysis

Paul Vancil
Dell Data Center Solutions, Architecture Group
Systems Management Architect
Topics

• Three Models for Rack Management in Use
• Example Rack Management – Dell DCS G5 Rack Management
• Rack Management RESTful API Analysis
DIFFERENT RACK MANAGEMENT MODELS
Three models for shared Infrastructure management

1. Shared Infrastructure managed via IPMI BMCs
   - Chassis or rack has shared power supplies, and shared fans
   - Servers have BMCs managed via shared or dedicated NIC on each blade
   - Chassis controller “pushes” shared fan / PSU sensor data to BMCs (via internal chassis bus)
   - User monitors and manages servers and shared resources via BMC (IPMI++)

2. Central Management – via rack-level management MC
   - Rack has shared power supplies, shared fans
   - Rack has a “Rack Management Controller”
   - All servers and shared components can be managed from the single Rack MC. (including console redirect)

3. Central Management + 1:1 management to BMCs
   - Central Mgt used for automation control and monitoring (model-2)
   - Direct connect to BMC (model-1) used for 1:1 server node debug
     - 1:1 management functions like iKVM, serial console redir
Variants to models 1 and 3—IPMI interfaces to BMCs

• Internal Rack Management Network
  – Rack infrastructure has internal L2 switch that connects to dedicated management port for each blade BMC.
  – One uplink out of the rack/chassis provides mgt access to all BMCs in rack.
  – Experience:
    ▪ Addresses model-1 cable issues, but
    ▪ Most customers don’t like to expose BMCs behind an integrated L2 switch on their DC networks

• Dedicated vs Shared Network Interfaces:
  – Some customers require dedicated Mgt Network
  – Some customers require shared Mgt Network
    ▪ to minimize cabling or TOR port usage
    ▪ a lot of problems and workarounds with 10GbE shared interfaces
EXAMPLE OF RACK-LEVEL MANAGEMENT

▪ DELL DCS G5
G5 Rack Physical Concept

- 1-10 blocks with multiple compute or storage sleds
  - 4 (full-width) sleds/block to 12 (1/3rd width) sleds/block

- 1-2 Power Bays w/ shared PSUs for rack/domain

- Shared fans in each block

- Rack Management Controller (RMC)
  - Located in PowerBay
  - Single point of management interface for each rack/domain
  - Rack/domain level management features
    - Rack power-on/off, power capping
    - Rack Power capping (rack has a budget)
    - Access to AC sockets, TORs.. From RMC
Top-Level G5 Management Architecture

- **Three Level Management Architecture**

  - **Rack/Domain Level** – *provides central point of management for rack/domain*
  - **Block Level** – *per-block fan control + interface to sleds within the block*
  - **Sled Level** – *sleds may have a BMC for individual per-sled management via front-end network – or sleds can be managed from the central management MC*
G5 Rack Level Management Functions

- RMC provides basic management of sleds and shared power/fan infrastructure:
  - **Sled** Power-on/off/reset, serial console redirect, power consumption, FRU inventory, sensor data
  - **Shared** fan and power supply monitoring and control
  - **Rack-level** power capping (rack level budget)
- RMC Interfaces via: serial port CLI, telnet/SSH CLI, SNMP, and future REST API
G5 RMC Software Stack

RMC User APIs and rack-level SW Apps - Future REST API

Base embedded Linux platform
- Embedded Linux apps
- Linux kernel
- Drivers
- Boot from eMMC or SD
- RMC hardware

G5 RMC Software Stack

NCLI, G5 DCS CLI, SNMP G5 MIB, Power capping, REST API

G5 Device Manager, availability features

Other services: ssh, telnet, busybox, BASH, etc...

SNMPv3 "agent"

http server

eLinux kernel with Network stack, filesystem

Enet, Serial, I2C, eMMC, SDcard

Boot FW + UBOOT boot ldr

Rack Mgt Cntlr HW
1000 MIPS uC, 512MB DDR, eMMC, SDcard

Engineering Workshop
RACK MANAGEMENT REST API ANALYSIS

Topics:
• Motivation / Objective
• What is a RESTful API
• Example APIs
Motivation—Addressing Customer Needs

• Need “Open” API with **Rack-level features** not in IPMI
  - Customers currently using proprietary APIs (cli and oem ipmi commands)

• Need Better API to interface with automation infrastructures
  - Customers currently using CLIs mostly (G5CLI, NCLI, ipmitool)

• Customer’s prefer a RESTful http API vs SMASH2.0, or proprietary HTTP-based API
Dell DCS RESTful API Objective

• Embrace Well-accepted RESTful http principles
• Make it easy to understand and use -- everything in one online spec
• Support across multiple platform types and rack mgt models
  – Model-1: BMCs in shared infrastructure
  – Model-2: G5 RMC rack-level management (all rack-level mgt features)
  – Model-3: Consistent resource IDs/links between BMC and RMC interfaces
  – Monolithic servers via BMC
• Easily extended and customized for OEM features
• Open, and industry standard
# Example APIs – similar to OpenStack v2 APIs

**BaseURI** = \(^=\) http[s]://<ipAddr>[::<port>]/OcpRest/[<oem>/]v1/Server/1

### APIs:

<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>^/servers</td>
<td>lists server collection: IDs, names, status</td>
</tr>
<tr>
<td>GET</td>
<td>^/server/&lt;n&gt;</td>
<td>Get server-&lt;n&gt; details (status/properties)</td>
</tr>
<tr>
<td>POST</td>
<td>^/server/&lt;n&gt;/action</td>
<td>execute server action eg power-on</td>
</tr>
<tr>
<td>RqData:</td>
<td></td>
<td>PowerOn, PowerOff, PowerCycle, Reseat, ...</td>
</tr>
<tr>
<td>PUT</td>
<td>^/server/&lt;n&gt;</td>
<td>set config data</td>
</tr>
<tr>
<td>RqData:</td>
<td></td>
<td>config data</td>
</tr>
<tr>
<td>GET</td>
<td>^/fans</td>
<td>list fan collection: IDs, name, status</td>
</tr>
<tr>
<td>GET</td>
<td>^/fan/&lt;n&gt;</td>
<td>Get fan&lt;n&gt; details</td>
</tr>
<tr>
<td>GET</td>
<td>^/psus</td>
<td>list Power Supply collection: IDs, status,...</td>
</tr>
<tr>
<td>GET</td>
<td>^/psu/&lt;n&gt;</td>
<td>Get PSU&lt;id&gt; details</td>
</tr>
<tr>
<td>GET</td>
<td>^/rmc/1</td>
<td>Get RMC properties</td>
</tr>
<tr>
<td>PUT</td>
<td>^/rmc/1</td>
<td>Set config data</td>
</tr>
<tr>
<td>RqData:</td>
<td></td>
<td>config data</td>
</tr>
<tr>
<td>GET</td>
<td>^/sleds</td>
<td>List Sled collection: IDs, type, status</td>
</tr>
<tr>
<td>GET</td>
<td>^/sled/&lt;id&gt;</td>
<td>Get sled details</td>
</tr>
</tbody>
</table>
Options for a API adoption or fork

1. IPMI
   - IPMI should be fast to develop, and RESTful capabilities would be valuable in IPMI
   - IPMI is widely supported and currently cheap to implement
   - IPMI may not be able to be expanded to suit all use cases.
   - Current BMC’s require backwards compatibility for a long period of time, slowing implementation

2. SNMP
   - Wide adoption in embedded systems
   - Limited capabilities, MIBs are currently problematic
   - Slow implementation of changes due to embedded systems

3. DMTF CIM compatible standards
   - DMTF standards are paywalled
   - DMTF is slow to implement changes
   - DMTF has a set of APIs that cover a large portion of what we are trying to accomplish
Advocacy

1. Fork DMTF standards and expand to serve our needs
   - CIM is an established standard that can be made RESTful
   - CIM compatible standards WBEM, SMI-S, VMAN, have already done much of the work that we will need to accomplish
   - There are already established adapters for IPMI, SNMP, and other protocols
   - WBEM is already supported in both Windows and Linux/Unix.

2. Submit the OCP Infrastructure API changes upstream to DMTF
   - The standard could be available openly on the Open Compute site and licensed freely
   - Changes could be made quickly, and DMTF adoption wouldn’t slow development
   - Increased compatibility with alternative DMTF standards

3. Create several discrete compatible APIs for varied functions
   - API is easier to understand and faster to develop
   - Libraries are faster and smaller for embedded systems
OPEN
Compute Summit
January 16-17, 2013
Santa Clara