RedDrum: an open source Redfish service

What is RedDrum?

It is an open source, python-based implementation of a Redfish service, and is the latest contribution from Dell EMC to the open community.

Named “Red Drum” after the Texas Redfish

Open source code includes:

- A recipe for backend integration with OpenBMC, and
- A full feature, 100% DMTF conformant simulator to help accelerate development

RedDrum is derived from the currently shipping Dell EMC DSS 9000 Rack Manager Redfish service.

RedDrum is available on GitHub at org:
https://github.com/RedDrum-Redfish-Project

What does RedDrum deliver?

Standard DMTF Redfish APIs to access common Hardware Management features

Easy-to-modify Python-based source code For accelerated development of new functionality.

A 100% compliant simulator that allows developers to confidently produce new code even without access to the target hardware.

Introducing the RedDrum Redfish Service

- A python-based Redfish Service
  - Built on Flask
  - Named for the most common type of “Redfish” in Texas—the Red Drum
  - Open Sourced now at: github.com/RedDrum-Redfish-Project
    - RedDrum-Frontend ---- the common Frontend code that implements the protocol
    - RedDrum-Simulator ---- a backend that implements the Simulator
    - RedDrum-OpenBMC ---- a backend for OpenBMC, w/ yocto recipes
    - RedDrum-Httpd-Configs --- httpd config for common httpd used including centos-httpd, and Apache2 for OpenBMC
- RedDrum was Designed to Support Three key use cases
  1. An upgraded full-feature simulator for DMTF (Profile Simulator V2)
     - Currently simulates 2 configs: 1) a monolythic server with the OCP Base Server Profile, and 2) a 4-node DSS9000 mini-rack
  2. A Multi-node Rack-Level Redfish service for DSS9000
     - Serves up to 100 nodes in rack from Linux low-end Atom server
     - Easy to understand, Easy to extend, Easy for customers to customize
     - All of data cached locally—for blazing performance and fast restart from cache
  3. A python easy to customize Open Source BMC Redfish Service for OpenBMC

Red Drum General Architecture

HTTPD-config

(httpd front-end for https)

- Apache reverse proxy
- Flask builtin http for test
- Could use Gevent or Ngnx

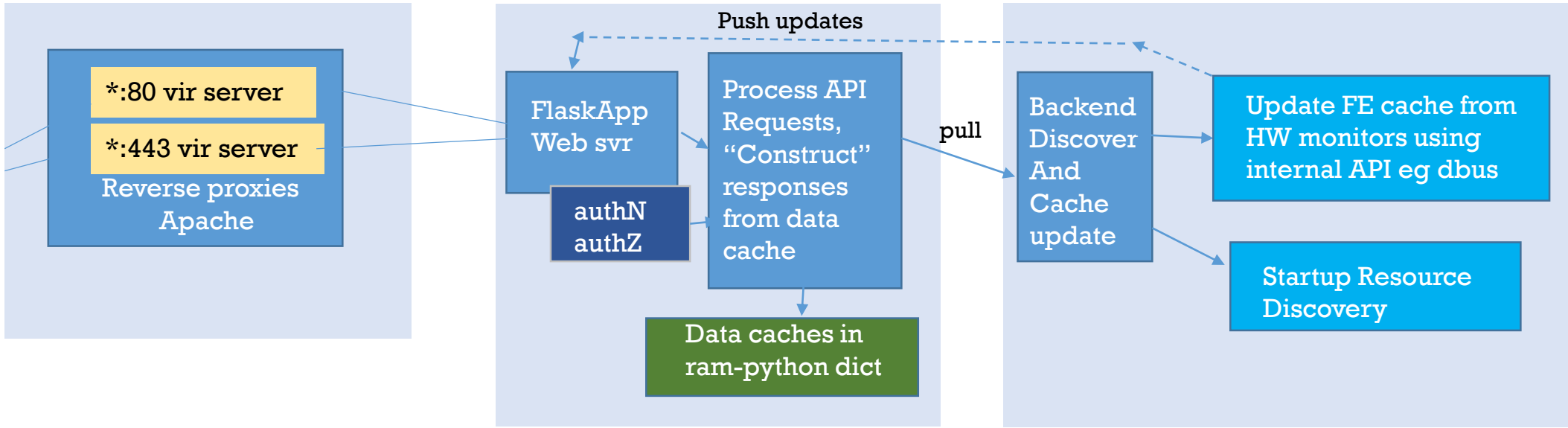
RedfishService

(the Redfish REST engine)

- FlaskApp
- Fully implements: Authentication, Authorization, ServiceRoot, AccountService, SessionService, EventService, JsonSchemas, Registries…

Backend (implementation dependent)

- Dell DSS9000 Rackmanager – multi-threaded backend to get data from up to 96 nodes and 8 fan controllers…
- OpenBMC -- calls Dbus APIs for update/disc
- Simulator – simple discover from json files



Multi-threaded https processing

Single-threaded but very fast

run from front-end thread if getting the data is fast, or else run separate Hwmon/action threads if slow

RedDrum Architecture points

- Relies on a separate HTTPD to implement SSL
  - Currently runs 4 Apache worker threads – SSL processing is a lot of the per-API delay
  - Connects to Front-end Flask service via localhost on port 5001, but could use WSGI interface
- Front-end based on Flask and implements the Redfish protocol
  - Implements a cache in RAM as python dict (grew from initial simulator model)
  - Single threaded because of cache --- but design is that this executes fast
    - If backend is slow, the backend needs to cache or start a thread to implement action
    - Structural hypermedia APIs (eg GET Chassis collection) are fast since no internal IPC is require
    - Service APIs eg AccountService, SessionService… are also optimized
  - Normal use case we see:
    - 1 client polling often in a single-threaded manner walking the hypermedia tree, and
    - every now and then another user queries – again single-threaded walking hypermedia tree
    - But In testing, we hit service w/ 10000 requests from multiple threads, which executed in 10-20 sec out of order on the Dss9000 rackmanager
- Backend – is where implementation-specific code is.
  - It provides the “data” for the frontend (quickly)
  - It it cant get the data quickly on demand, it needs to poll and cache the data and run action threads so that it doesn’t block the frontend

Red Drum Simulator

HTTPD-config

(httpd front-end for https)

- Centos Apache httpd

  reverse proxy with https

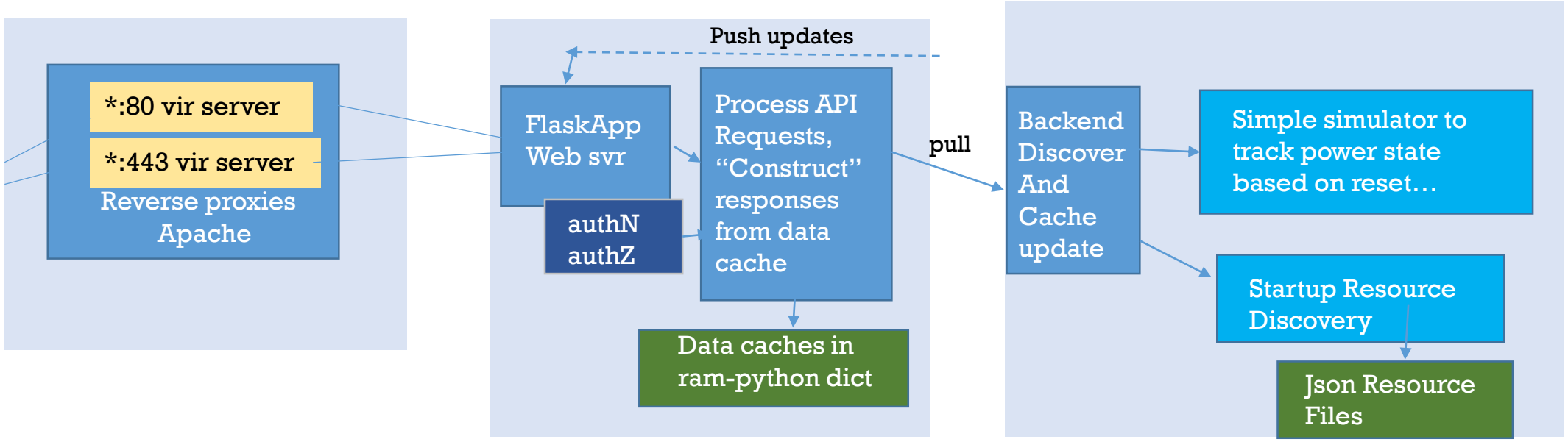
RedfishService

(the Redfish REST engine)

- FlaskApp
- Fully implements: Authentication, Authorization, ServiceRoot, AccountService, SessionService, EventService, JsonSchemas, Registries…

Simulator Backend

- Discovers resources from static json files that describe the resources for the specific profile/config
- Simulator tracks volatile state like power state so that power state changes based on reset
- Other writes are remembered in theh front-end cache
- Persistent front end cache enabled so state is remembered across re-starts of the simulator



Multi-threaded https processing

Single-threaded but very fast

run from front-end thread since fast

Red Drum use on Dell DSS9000 RackManager

HTTPD-config

HTTPD-config

(httpd front-end for https)

- Centos Apache httpd

reverse proxy with https

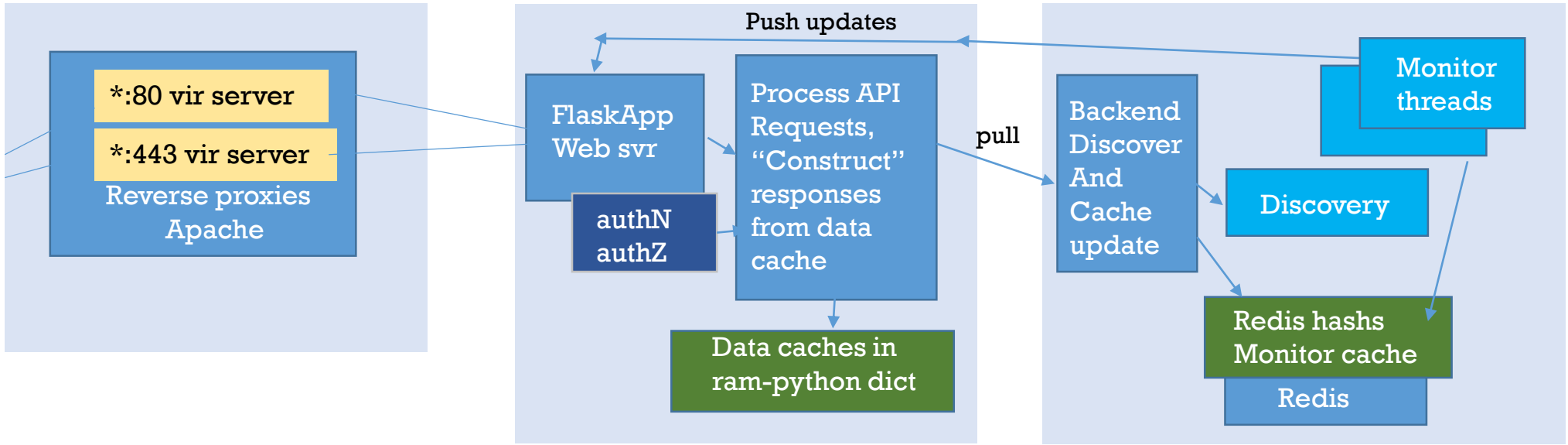
RedfishService

(the Redfish REST engine)

- FlaskApp
- Fully implements: Authentication, Authorization, ServiceRoot, AccountService, SessionService, EventService, JsonSchemas, Registries…

Dss9000 rack level Backend

- Has up to 100 separate threads to monitor each node and cache the node data
- Has several separate threads to monitor fans and powerSupplies from fan controller and powerBay controllers
- Uses a backend Redis RAM cashe to hold data for frontend to get data from the monitors



Multi-threaded https processing

Single-threaded but very fast

64 node HW monitor threads
4 powerbay/fan controller threads
Pushes hot plug changes to frontend
Some low-use APIs are on-demand

Red Drum OpenBMC integration

HTTPD-config

(httpd front-end for https)

- Apache2 httpd

  reverse proxy with https

Used port 5050 since 443 was claimed by Gevent

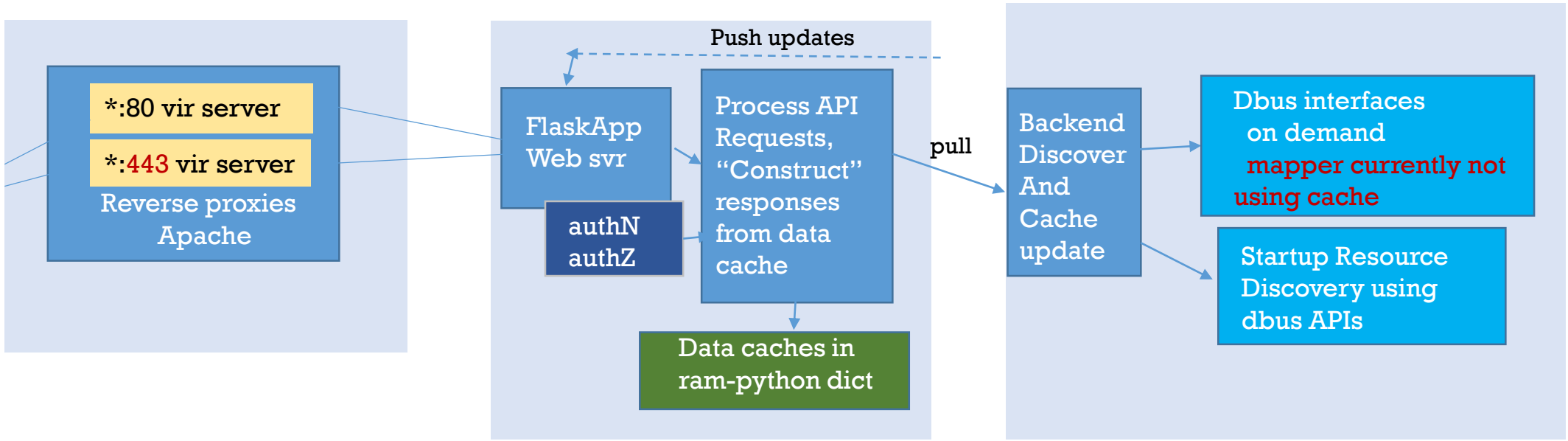
RedfishService

(the Redfish REST engine)

- FlaskApp
- Fully implements: Authentication, Authorization, ServiceRoot, AccountService, SessionService, EventService, JsonSchemas, Registries…

OpenBMC Backend

- Dbus calls to get data from HWMonitor
- Uses some Linux APIs



Multi-threaded https processing

Single-threaded but very fast

Running from front-end thread on-demand
(we thought it would be fast)
Can cache data if we need to

Performance

- Flash filesystem usage: ~8.5MB (32-64MB small mem BMCs, 4.3GB eMMC on iDrac)
  - 8.5MiB for Python3+Flask+other required modules
  - 0.9MiB Apache but we can run it behind GEvent
- RAM usage: ~32MB (typ 256MB)
  - 12MB for Redfish Service (caches, python)
  - 4MB for base Apache + 16MB for 4 worker threads
- Execution Performance:
  - Aspeed 2400:
    - 0.15 (flask) + .3sec (SSL) + .3sec(Auth hash) + 1-8sec(dbus uncached mapper)= .15 - 8 sec
      - Dbus calls slow because we were calling the dbus mapper uncached so it was discovering on ea call
    - http no auth) GET /redfish/v1 **0.15 sec**
    - **https: basicAuth GET /redfish/v1/Chassis/1 2.25 sec**
    - GET /redfish/v1/Chassis/1/Power w/ SSL and BasicAuth: **8.5 sec**
  - Nuvaton Poleg running simulator:
    - 0.02 (flask) + .11sec (SSL) + .07sec(Auth hash) + ?sec(dbus r)= ? (0s on simulator)
      - Without using the HW acceleration for the SSL and hashing. The HW accel will drop these to < .01
    - http no auth) GET /redfish/v1 **0.02 sec**
    - **https: basicAuth GET /redfish/v1/Chassis/1 .20 sec w/o dbus calls**
    - GET /redfish/v1/Chassis/1/Power w/ SSL and BasicAuth: **.21 sec w/o dbus calls**

Performance Comparisons

Redfish Service	Root Svc (no auth, http)	Typical large GET response (auth+https)
RedDrum On OpenBMC Aspeed 2400	0.15 sec	2 – 2.5 sec w/o dbus mapper cache
Estimated RedDrum on OpenBMC Nuvaton	0.02	Est .5-1sec
iDrac 13G	0.5 sec	2 – 6 sec
iDrac 14G	0.11 sec	0.4 – 0.65 sec
RackManager (Atom)	0.06 sec	0.06 – 0.08 sec
Typical BMC at last yrs DMTF plugfest	.2 sec	.5-2

RedDrum Code

- Repo github.com/RedDrum-Redfish-Project
- Repos:
  - ./RedDrum-Frontend – common frontend,
    - could move to openbmc repo, or stay here as upstream code
  - ./RedDrum-OpenBMC – yocto recipes, httpd cfg, OpenBMC backend
    - This will move to github/openbmc
  - ./RedDrum-Simulator – simulator backend
    - This could move to DMTF site