MegaRAC Open Redfish Framework (MORF)

Presented by:
March 28, 2018

Features

- Asynchronous, Non-blocking, event-driven framework
- Stable stack shipping on multiple platforms/solutions
- Easily customizable and reusable for any Redfish based service (RSD, Swordfish etc)
- Multi-arch compatible
- Event Service
- Asynchronous Task Service
- Simplified OEM extensions model

Technologies

- LuaJIT
- Redis
- TurboLua Framework

Why LuaJIT?

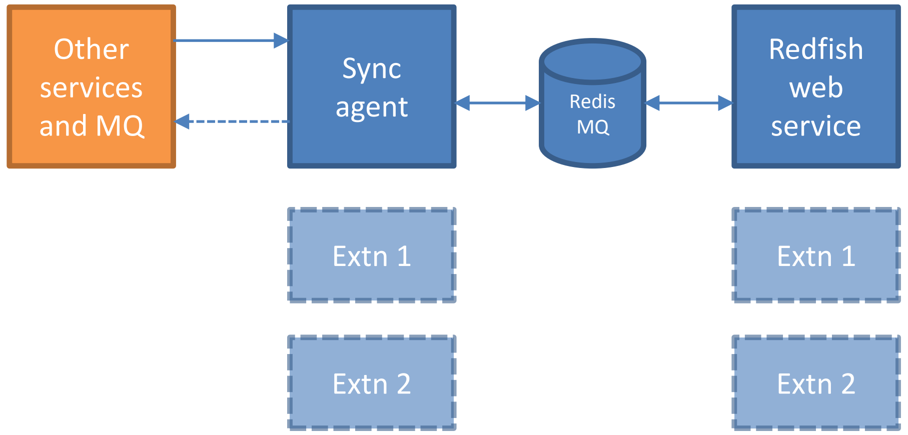
- Fast scripting language
- Small interpreter size ~300KB
- Foreign Function Interface – Direct C library invocation
- Coroutines
- Garbage collector
- Easy to learn with any scripting language experience
- Allows bytecode distribution

Why Redis?

- Lightweight in-memory db
- Key-value pair data model – JSON like
- Pub/Sub feature
- Key Space Notification
- Persistent
- Advanced scripting in db with built-in lua interpreter

Redfish Framework

- **Consists of:**
  - Redfish frontend web service
  - Intermediate MQ on Redis
  - Redfish backend service (sync agent)
- **Asynchronous, Non-blocking, Event-driven**
- **Authentication Models: Basic, Redfish Session (X-Auth-Token), Certificate based**
- **SSL**
  - Our solutions run behind web server like Lighttpd/Nginx
  - Also supports SSL by itself
- **Dynamic Routes**
- **Extensible frontend and backend services**
- **OEM extension on properties as well as URIs**
- **User privilege**



Redfish web service

- Base handler – turbo.web.RequestHandler
- RedfishHandler > ODataHandler > turbo.web.RequestHandler
- All other handler derives RedfishHandler

```
local ChassisHandler = class("ChassisHandler", RedfishHandler)
```

- Handlers override HTTP methods to implement functions. If not they are handled by default functions implemented in RedfishHandler/ODataHandler.
- Simple JSON to Redfish key mapper

```
Redfish:Chassis:Self:Name                                    "Computer System Chassis"
Redfish:Chassis:Self:ResourceExists                     true
Redfish:Chassis:Self:ChassisType                             "StandAlone"
```

- Simple GET handler db query

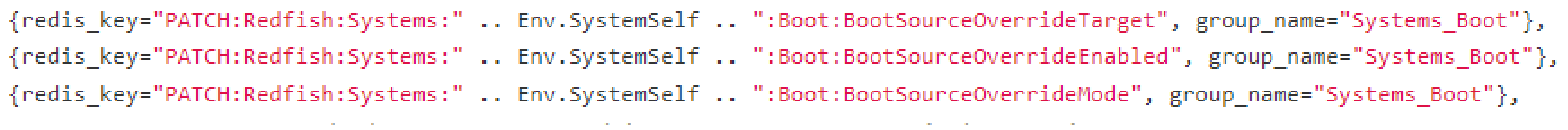
```
local pl = redis:pipeline()

local prefix = "Redfish:Chassis:" .. id

pl:mget({
                prefix .. ":Name",
                prefix .. ":Model",
                prefix .. ":PartNumber",
                prefix .. ":AssetTag",
                prefix .. ":IndicatorLED",
```

Sync agent

- Everything is an extension
- Init-agent (config inits, discovery etc)
- Redis watcher
- inotify watcher
- Redfish event generator based on Log service
- Short-lived task orchestrator
- Live reload for dynamic extension

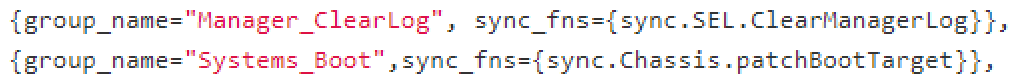
Sync agent

- **Intermediate key – group map**



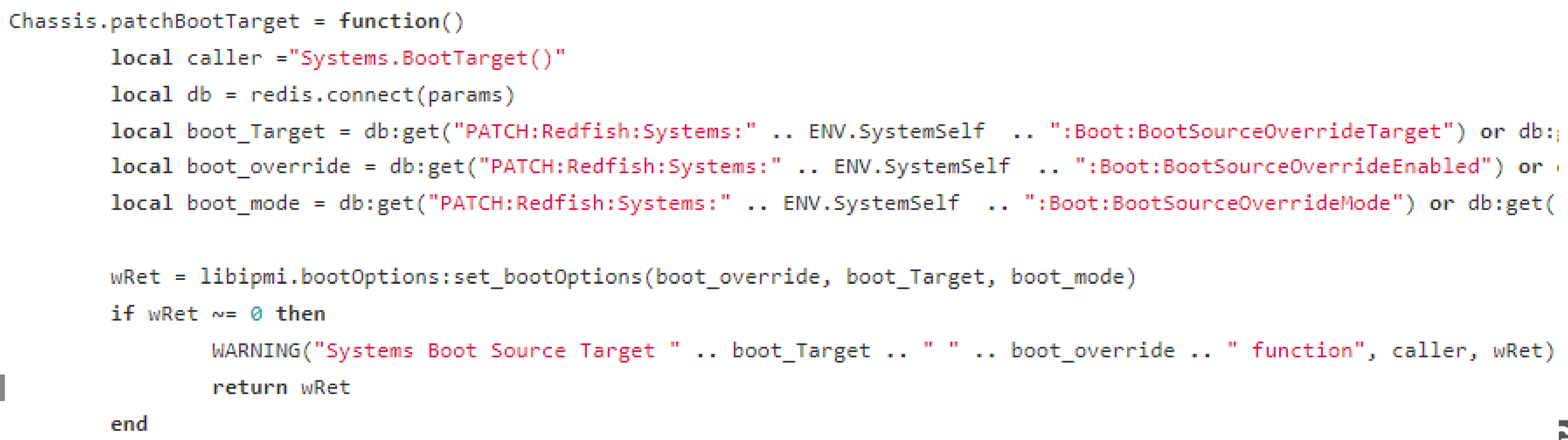
```
{redis_key="PATCH:Redfish:Systems:" .. Env.SystemSelf .. ":Boot:BootSourceOverrideTarget", group_name="Systems_Boot"},
{redis_key="PATCH:Redfish:Systems:" .. Env.SystemSelf .. ":Boot:BootSourceOverrideEnabled", group_name="Systems_Boot"},
{redis_key="PATCH:Redfish:Systems:" .. Env.SystemSelf .. ":Boot:BootSourceOverrideMode", group_name="Systems_Boot"},
```

- **Group - function map**



```
{group_name="Manager_ClearLog", sync_fns={sync.SEL.ClearManagerLog}},
{group_name="Systems_Boot",sync_fns={sync.Chassis.patchBootTarget}},
```

- **Example function**



```
Chassis.patchBootTarget = function()
        local caller ="Systems.BootTarget()"
        local db = redis.connect(params)
        local boot_Target = db:get("PATCH:Redfish:Systems:" .. ENV.SystemSelf  .. ":Boot:BootSourceOverrideTarget") or db:
        local boot_override = db:get("PATCH:Redfish:Systems:" .. ENV.SystemSelf  .. ":Boot:BootSourceOverrideEnabled") or
        local boot_mode = db:get("PATCH:Redfish:Systems:" .. ENV.SystemSelf  .. ":Boot:BootSourceOverrideMode") or db:get(

        wRet = libipmi.bootOptions:set_bootOptions(boot_override, boot_Target, boot_mode)
        if wRet ~= 0 then
                WARNING("Systems Boot Source Target " .. boot_Target .. " " .. boot_override .. " function", caller, wRet)
                return wRet
        end
```

Services, Testing & Documentation

- Postman based test collections for API (can run automated - headless)
- Busted and Lunit based unit tests
- Coverage report through luacov
- Automated code generation based on schema
- Literate style documentation