OpenDCRE Documentation

Release 1.1

Vapor IO

March 06, 2016

Contents

Vapor IO Software and Hardware documentation, reference and downloads.

CHAPTER 1

OpenDCRE (Open Data Center Runtime Environment)

1.1 Introduction

OpenDCRE provides a securable RESTful API for monitoring and control of data center and IT equipment, including reading sensors and server power control - via power line communications (PLC) over a DC bus bar, or via IPMI over LAN. The OpenDCRE API is easy to integrate into third-party monitoring, management and orchestration providers, while providing a simple, curl-able interface for common and custom devops tasks.

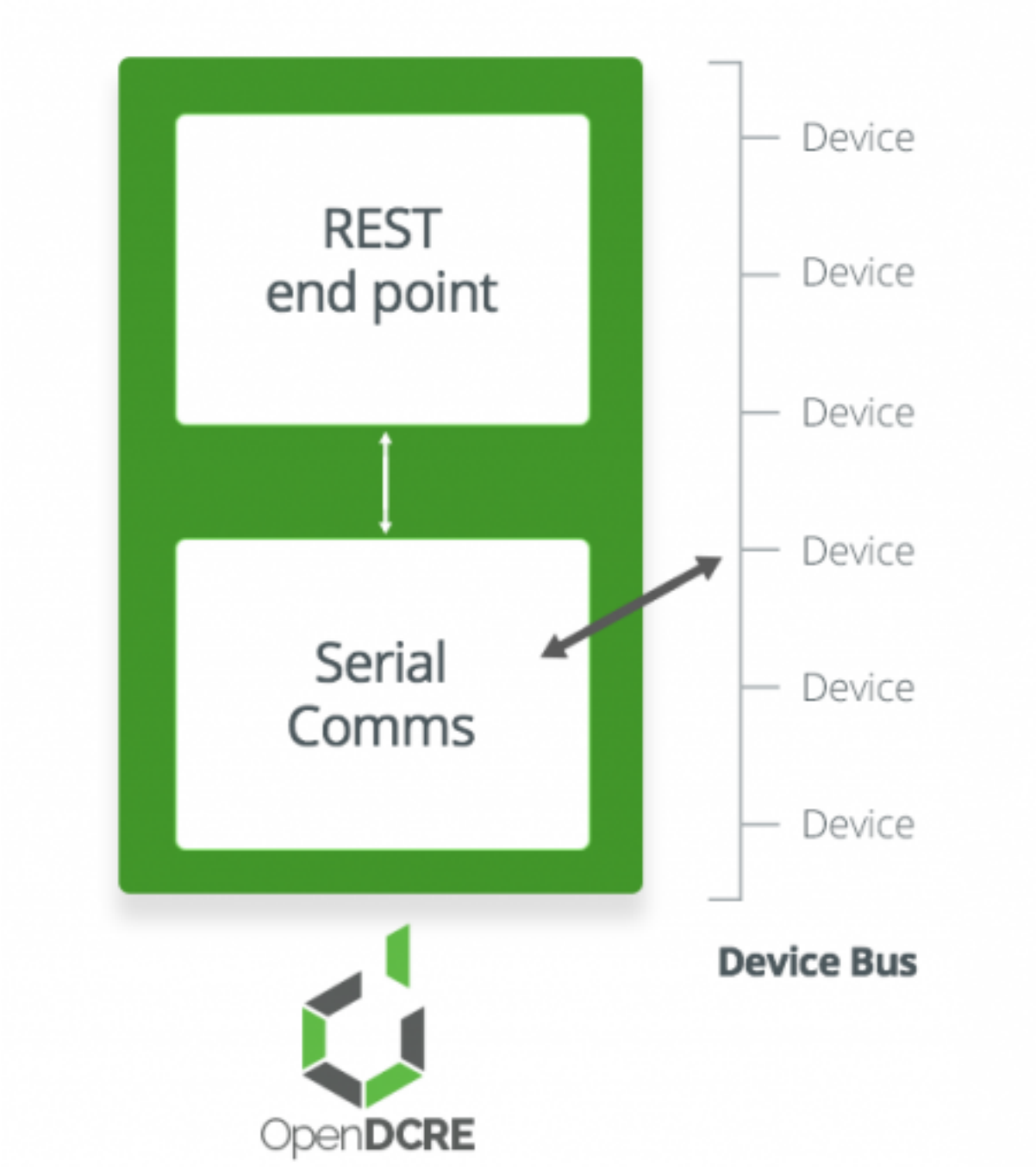
1.1.1 Features

- Analog sensor support (Thermistor)
- Power control and status via power-line communications (PLC)
- IPMI Bridge - power control and status via IPMI
- Asset information for servers (`power` devices - PLC and IPMI)
- Simple curlable RESTful API
- Securable via TLS/SSL
- Integration with existing Auth providers (OAuth, LDAP, AD, etc.)

1.1.2 Architecture

OpenDCRE is part of the OpenMistOS Linux distribution that runs on the Raspberry Pi 2 Model B. OpenMistOS includes a custom Docker package, compiled for ARMv7 (armhf), and OpenDCRE is packaged as a Docker container.

OpenDCRE exposes a RESTful API via an HTTP endpoint in the OpenDCRE container. The HTTP endpoint is comprised of nginx as the front-end, with uwsgi as a reverse proxy for a Python Flask application. Within Flask, OpenDCRE uses a text-based serial protocol to communicate with the OpenDCRE device bus, which is the primary communications channel between API users and the device bus.



The OpenDCRE device bus is comprised of a set of boards and devices, individually addressable, and globally scannable for a real-time inventory of addressable devices attached to the bus. The OpenDCRE device bus allows devices to be read and written, and for various actions to be carried out, such as power control (on/off/cycle/status). Additionally, when a physical OpenDCRE device bus is not present, a software emulator can be used to simulate OpenDCRE API commands and functionality.

All included components of OpenDCRE can be customized, integrated and secured via configuration file (nginx, uwsgi), and output their logs to a common location (/var/log/opendcre).

1.1.3 Applications

OpenMistOS and OpenDCRE can be used as an open platform for monitoring and managing data center hardware, software and environmental characteristics. Given the small form-factor of the Raspberry Pi plus its HAT board, there are a wide variety of possible applications, deployments, physical mounting strategies, and network connectivity options. Community support helps OpenDCRE grow, and enables new functionality.

1.1.4 OpenMistOS

OpenMistOS (OMOS) is an open-source operating system distribution for the Raspberry Pi, featuring OpenDCRE, the Open Data Center Runtime Environment. OMOS was developed for the purpose of using a small, single-board computer like the Raspberry Pi to perform data center sensorfication and control, particularly in concert with OpenCompute-based open hardware.

OMOS is Debian (wheezy)-based, featuring Docker capabilities baked into the OS, with OpenDCRE running as a Dockerized service of the OS. OpenMistOS was developed by a team with decades of experience in data centers large and small, and is working to take the pain out of proprietary and arcane technologies.

1.2 Getting OpenDCRE & OpenMistOS

1.2.1 Requirements

Hardware Requirements

- Raspberry Pi 2 Model B (other Raspberry Pi versions are not supported).
- 4GB Micro SD card - 8GB or larger recommended.
- 5V Micro USB power source.
- Wired ethernet connection (wireless supported but not recommended).
- OpenDCRE HAT (optional)
- DC Bus Bar for power line communications (optional)
- IPMI 1.5-compliant BMC for IPMI bridge (optional)
- HDMI/HDMI-VGA video cable & monitor (optional)

Software Requirements

- OpenMistOS v1.0.0 or later.

1.2.2 Download and Install

Download

- Download OpenMistOS , and decompress the .img file. OpenDCRE Source on GitHub.

Install

- **Insert Micro SD card into card reader, and determine the SD card device:**
  - **MacOS:**
    * `sudo diskutil list`
  - **Linux:**
    * `sudo fdisk -l`
- **Use `dd` to write image to card:**
  - **MacOS:**
    * `sudo dd if=<.img file> of=<sd card device> bs=4m`
  - **Linux:**
    * `sudo dd if=<.img file> of=<sd card device> bs=4M`
  - **Note:**
    * `<.img file>` is the path and filename of the decompressed OpenMistOS .img downloaded above.
    * `<sd card device>` is the SD card device determined in the previous step. (e.g. -/dev/disk1)

When executing the above commands, if an error is returned similar to: `dd: <sd card device>: Resource busy` then the SD card must be unmounted.To do this, identify the SD card partition (can use `df -h` for this, or the results from determining the SD card device, above), then unmount the partition:

- ***MacOS*:**
  - `sudo diskutil unmount <sd card device>`
- ***Linux*:**
  - `sudo umount <sd card device>`

When `dd` is complete, OpenMistOS is ready to run from the SD card. Plug the Raspberry Pi into the wired network, insert the Micro SD card, and power up the Raspberry Pi.

At completion of the boot process, the OpenMistOS device IP address is displayed on screen (if video connection is used); alternately, check DHCP or router logs to determine the IP address of the OpenMistOS device.

Login

Ssh into the OpenMistOS device:

- *Username*: `openmistos`
- *Password*: `0p3ndcr3!`

The openmistos user has sudo rights on OpenMistOS. It is recommended to **immediately** change the openmistos password to a new, secure, password.

Note: OpenMistOS, like other Raspberry Pi OSes, uses only the space required for the OS on the SD card. It is recommended to change this behavior so that the entire space on the SD card is used. To do this, enter the configuration menu on first login:

```
$ sudo raspi-config
```

In the configuration menu, there should be an option to use the entire disk. Once selected and confirmed, OpenMistOS will restart and the entire SD card will then be used.

Verification

There are several methods for verifying that OpenDCRE is running properly.

Browser

Navigate to:

```
http://<openmistos ip address>:5000/opendcre/1.1/test
```

Output should be similar to:

```
{
    "status": "ok"
}
```

Command-Line

Running: `$ docker ps`

produces output similar to:

```
CONTAINER ID        IMAGE                       COMMAND                  CREATED             STATUS
a9419ff86502        vaporio/opendcre:latest     "./start_opendcre.sh   4 days ago          Up 4 days
```

(when using the HAT)

or:

```
CONTAINER ID        IMAGE                       COMMAND                  CREATED             STATUS
2281101f6a60        vaporio/opendcre:latest     "./start_opendcre_em   4 days ago          Up 4 days
```

(when using the emulator)

The OpenDCRE service starts automatically at boot; it can be started/stopped via the init.d daemon:

```
$ sudo /etc/init.d/opendcre <start|stop>
```

Logs

By default, OpenDCRE logs are placed in /var/log/opendcre . Access, error and daemon logs are available for troubleshooting and analytics.

1.3 Configuring OpenDCRE & OpenMistOS

1.3.1 OpenDCRE Emulator

Emulator Configuration

In the absence of HAT hardware, OpenDCRE automatically switches to using a software emulator, to simulate various capabilities of OpenDCRE. The OpenDCRE emulator is configured by means of a JSON file (the default file

is `simple.json`) which is stored in the `/opendcre/opendcre_southbound` directory of the OpenDCRE Docker container.

To modify the emulator output, users may clone the OpenDCRE GitHub repository, and change the contents of `simple.json`, then rebuild the OpenDCRE container.

Example emulator configuration from `simple.json` is below, followed by an explanation of the contents.

```
{
  "boards": [
    {
      "board_id": "00000001",
      "firmware_version" : "OpenDCRE Emulator v1.1.0",
      "devices" : [
        {
          "device_id": "01ff",
          "device_type": "thermistor",
          "read":
            {
              "repeatable": true,
              "responses": [
                656,
                646,
                636,
                625,
                615,
                605,
                594,
                584,
                573,
                563,
                553,
                542,
                532,
                522,
                512,
                502,
                491,
                482,
                472,
                462,
                452
              ]
            }
        },
        {
          "device_id": "02ff",
          "device_type": "none",
          "read":
            {
              "repeatable": true,
              "responses": [ ]
            }
        },
        {
          "device_id": "03ff",
          "device_type": "thermistor",
          "read":
            {
```

```
            "repeatable": true,
            "responses": [
              656,
              646,
              636,
              625,
              615,
              605,
              594,
              584,
              573,
              563,
              553,
              542,
              532,
              522,
              512,
              502,
              491,
              482,
              472,
              462,
              452
            ]
          }
      },
      {
        "device_id": "04ff",
        "device_type": "none",
        "read":
          {
            "repeatable": true,
            "responses": [ ]
          }
      },
      {
        "device_id": "05ff",
        "device_type": "none",
        "read":
          {
            "repeatable": true,
            "responses": [ ]
          }
      },
      {
        "device_id": "06ff",
        "device_type": "none",
        "read":
          {
            "repeatable": true,
            "responses": [ ]
          }
      },
      {
        "device_id": "07ff",
        "device_type": "none",
        "read":
          {
```

```
            "repeatable": true,
            "responses": [ ]
          }
        },
        {
          "device_id": "08ff",
          "device_type": "thermistor",
          "read":
            {
              "repeatable": true,
              "responses": [
                656,
                646,
                636,
                625,
                615,
                605,
                594,
                584,
                573,
                563,
                553,
                542,
                532,
                522,
                512,
                502,
                491,
                482,
                472,
                462,
                452
              ]
            }
        },
        {
          "device_id": "09ff",
          "device_type": "thermistor",
          "read":
            {
              "repeatable": true,
              "responses": [
                656,
                646,
                636,
                625,
                615,
                605,
                594,
                584,
                573,
                563,
                553,
                542,
                532,
                522,
                512,
                502,
```

```
            491,
            482,
            472,
            462,
            452
          ]
        }
      },
      {
        "device_id": "0aff",
        "device_type": "thermistor",
        "read":
          {
            "repeatable": true,
            "responses": [
              656,
              646,
              636,
              625,
              615,
              605,
              594,
              584,
              573,
              563,
              553,
              542,
              532,
              522,
              512,
              502,
              491,
              482,
              472,
              462,
              452
            ]
          }
      },
      {
        "device_id": "0bff",
        "device_type": "none",
        "read":
          {
            "repeatable": true,
            "responses": [ ]
          }
      },
      {
        "device_id": "0cff",
        "device_type": "none",
        "read":
          {
            "repeatable": true,
            "responses": [ ]
          }
      },
      {
```

```
            "device_id": "0dff",
            "device_type": "power",
            "power":
              {
                "repeatable": true,
                "responses": [
                  "0,0,0,0"
                ]
              }
          }
        ]
      }
  ]
}
```

The OpenDCRE emulator simulates a single OpenDCRE HAT board with 13 devices. The JSON document file is structured around a collection of boards and devices.

Boards

Each board must have a `board_id` and `firmware_version` field. Each `board_id` must be a unique 4-byte value, encoded as a hex string between "00000000" and "00FFFFFF" (the upper byte is reserved, and must always be 00), and `firmware_version` must be a string value (including empty string).

The `board_id` is used in the OpenDCRE API to address a given board, while the `firmware_version` field is used to populate the `firmware_version` field of the response to the OpenDCRE "version" command for a given board (e.g.

```
http://<ipaddress>:5000/opendcre/1.1/version/1
```

gets the version information for board 1).

As with all commands in OpenDCRE, if a board or device does not exist in the emulator configuration, then a 500 error is returned as the result of a given command.

Devices

A given board also has a collection of devices. Each device is identified by a `device_id`, used to indicate a given device in an OpenDCRE command - e.g.:

```
http://<ipaddress>:5000/opendcre/1.1/read/thermistor/00000001/01ff
```

The `device_id` field is a 2-byte value represented as a hexadecimal string that is unique to a given board.

Device Types

The `device_type` field must be present, and must contain a string value that corresponds to an OpenDCRE-supported device type. This list includes:

- `thermistor`
- `power`
- `humidity`
- `pressure`
- `led`

- ipmb
- door_lock
- current
- temperature
- none

A device type of none indicates that no device is present at a given device_id on the given board, and may be ignored.

Other device types (e.g. for additional sensors and actions) will be added in future revisions of OpenDCRE, or may be added by developers wishing to add support for other device types.

Finally, a field corresponding to the action supported for a given device type is required. A map of device types to supported actions is below:

Device Type	Action Supported
thermistor	read
power	power
humidity	read
pressure	none (may be added in future OpenDCRE release)
led	none (may be added in future OpenDCRE release)
ipmb	none (may be added in future OpenDCRE release)
door_lock	none (may be added in future OpenDCRE release)
current	none (may be added in future OpenDCRE release)
temperature	none (may be added in future OpenDCRE release)
none	none

Read

For the read action's field in the OpenDCRE emulator configuration, two fields may be configured, relating to the responses returned from a read command for the given device.

First, the repeatable field may be set to true or false, depending on whether it is desirable for the list of responses set in the responses field to repeat in a round-robin fashion, or if a device should stop returning data after its response list has been exhausted.

The responses field is a list of zero or more raw (integer) values that may be returned for a given read command. The raw values are converted by the built-in OpenDCRE conversion functions, based on the given device_type. Some examples are given for the thermistor sensor device type in the simple.json file.

When a list of values is provided for responses, the emulator iterates sequentially through the items in that list, until the list is exhausted (if repeatable is set to "true", then the emulator returns to the beginning of the list).

An empty responses list means the device returns no data, which translates to a 500 error for the read command at the OpenDCRE REST API level (useful for simulating errors). To always return the same single value, a responses list with a single element, and repeatable set to "true" will suffice.

Power

For the power action's field in the OpenDCRE emulator configuration, similar fields are present - repeatable and responses.

For every power command (e.g. on/off/cycle/status) issued to a power device in the OpenDCRE emulator, a response is returned from the responses list, which may be repeatable or non-repeatable. The values in the responses list correspond to power status values returned over PMBUS from the hot swap controller on an OCP server, and

are expressed as an integer value in the emulator configuration (see example above). OpenDCRE converts the raw response to a friendly power status result using its built-in conversion functions.

Other Notes

The emulator configuration in `simple.json` is designed to provide a simple view and demonstration of how OpenDCRE works. The OpenDCRE emulator is also used for testing purposes, and additional emulator configurations may be found under the `/opendcre/opendcre_southbound/tests` directory of the OpenDCRE Docker container.

An invalid emulator configuration will cause the OpenDCRE emulator to fail to start or function properly.

Hardware

When using the OpenDCRE HAT board with OpenMistOS and OpenDCRE, no software or configuration changes are necessary. The "scan" command (e.g.

```
http://<ipaddress>:5000/opendcre/1.1/scan
```

) provides a real-time list of devices present on the HAT board.

To switch between the HAT and emulator, simply power-down the OpenMistOS Raspberry Pi, and add or remove the HAT. On power-up, the HAT's presence or absence will be detected, determining whether or not the emulator should be used.

1.3.2 IPMI Bridge

About

The Vapor IPMI bridge allows users of OpenDCRE to use both bus-bar-based power line communications, and LAN-based IPMI communications for equipment monitoring and management. The IPMI bridge is included with OpenDCRE v1.1.0 and later, and supports power control and status via IPMI using the OpenDCRE REST API.

Requirements

- OpenMistOS must be connected to a wired LAN network that can reach all BMCs configured to be managed over OpenDCRE.
- Knowledge of BMC IP addresses, authentication types and usernames and passwords (where applicable) required.
- **Authentication Types supported:**
  - NONE (no username or password, not recommended)
  - PASSWORD (username and password, sent in clear text, not recommended)
  - MD2
  - MD5

Configuration

The Vapor IPMI bridge is configured via the `bmc_config.json` file, located in the top level of the OpenDCRE distribution. An example file, `bmc_config_sample.json` is included with OpenDCRE, and may be modified to one's environment.

All IPMI BMCs successfully configured will show up on a `scan` command result as devices under `board_id` 40000000.

```
{
  "bmcs": [
    {
      "bmc_device_id": 1,
      "bmc_ip": "192.168.1.118",
      "username": "username",
      "password": "password",
      "auth_type": "MD5",
      "asset_info": "example BMC info"
    }
  ]
}
```

For each BMC supported, an entry is added to the `bmcs` list above. Each entry must include:

- `bmc_device_id` - a numeric value corresponding to the `device_id`, must be unique.
- `bmc_ip` - the IP address (as a string) corresponding to the BMC to be managed. IP address must be reachable by OpenMistOS.
- `username` - the username to use in connecting to the BMC - may be an empty string if no username is used.
- `password` - the password to use in connecting to the BMC - may be an empty string if no username is used.
- **`auth_type` - the type of authentication to use in connecting to the BMC, supported values:**
  - `NONE`
  - `PASSWORD`
  - `MD2`
  - `MD5`
- `asset_info` - a string (up to 127 bytes) containing asset information about the given BMC/server, returned via the read-info command for the device.

Once the configuration file has been successfully edited, rebuild the OpenDCRE Docker container, and verify the configured BMC devices are returned via a `scan` command under board 40000000.

BMC devices will show up as `power` devices, and all `power` commands (`on`, `off`, `cycle`, `status`) are supported.

Reading asset information about a given BMC can be carried out via the `read` command for the info field for the given BMC device. BMC `asset_info` is readable via the OpenDCRE endpoint, but is not writeable via the endpoint.

If the configuration file contains errors or is missing, no devices will show up under the IPMI `board_id` on a `scan` command.

1.3.3 Configuring OpenDCRE

Customization

OpenDCRE may be customized in a variety of ways, most commonly by changing the HTTP endpoint port, adding TLS certificates for HTTPS support, or by integrating OpenDCRE with a site-specific authentication provider.

Port

To change the port OpenDCRE listens on, edit the `opendcre_nginx.conf` file, and rebuild the OpenDCRE docker container. Be sure to also update the `DOCKER_RUN` variable in the `/etc/init.d/opendcre` init.d script to indicate the correct port mapping for the Docker container, as well.

```
server {
    listen 5000;
    server_name localhost;
    charset utf-8;
    access_log /logs/opendcre.net_access.log;
    error_log /logs/opendcre.net_error.log;

    location / {
        add_header 'Access-Control-Allow-Origin' '*';
        uwsgi_pass unix://var/uwsgi/opendcre.sock;
        include /etc/nginx/uwsgi_params;
    }
}
```

TLS/SSL

TLS/SSL certificates may be added to OpenDCRE via Nginx configuration. Refer to Nginx documentation for instructions on how to enable TLS.

Authentication

As OpenDCRE uses Nginx as its reverse proxy, authentication may be enabled via Nginx configuration - see Nginx documentation for instructions on how to enable authentication.

1.3.4 Running and Testing OpenDCRE

Normally, OpenDCRE may be started and stopped via its built-in OpenMistOS init.d script (`/etc/init.d/opendcre {start|stop|restart}`).

When starting OpenDCRE manually, the following steps may be followed.

- First, OpenDCRE expects a volume to be exposed for logs (`/logs` is the location within the container, which should be mapped externally).
- Additionally, OpenDCRE, by default, uses TCP port 5000 to listen for API requests.
- In cases where the OpenDCRE HAT is used with the OpenDCRE container, the /dev/ttyAMA0 serial device is also required.

With HAT

To start OpenDCRE with the HAT device attached:

```
docker run -d -p 5000:5000 -v /var/log/opendcre:/logs --device /dev/ttyAMA0:/dev/ttyAMA0 opendcre ./s
```

With Emulator

To start OpenDCRE in emulator mode:

```
docker run -d -p 5000:5000 -v /var/log/opendcre:/logs opendcre ./start_opendcre_emulator.sh
```

Run Tests

To run the OpenDCRE test suite:

```
docker run -ti -v /var/log/opendcre:/logs opendcre ./opendcre_southbound/bus-test.py
```

1.3.5 Building OpenDCRE Docker Container

Building a modified OpenDCRE container is a fairly straightforward process. First, clone the OpenDCRE GitHub repository to a location on OpenMistOS.

Next, to build a custom distribution of OpenDCRE (for example, to include site-specific TLS certificates, or to configure nginx to use site-specific authn/authz), the included Dockerfile can be used to package up the distribution.

In the simplest case, from the opendcre directory:

```
docker build -t vaporio/opendcre:custom-v1.1.0 .
```

Apply whatever tag is most descriptive for the custom image.

From this point, test and run OpenDCRE to ensure the changes were successful.

1.3.6 Updating OpenDCRE

OpenDCRE Updates

In OpenMistOS, upgrades to OpenDCRE may be carried out by updating the OpenDCRE docker container. To do this, first stop OpenDCRE:

```
$ sudo /etc/init.d/opendcre stop
```

Then, log in to Docker Hub (assumes OpenMistOS has Internet access):

```
$ docker login
```

(enter your Docker Hub username, password and email address)

```
$ docker pull vaporio/opendcre
```

If an update is available, the latest version of opendcre will be pulled down to OpenMistOS.

Finally, start OpenDCRE again:

```
$ sudo /etc/init.d/opendcre start
```

OpenMistOS Updates

To update OpenMistOS:

```
$ sudo apt-get update && sudo apt-get upgrade
```

1.4 API Reference

The examples below assume OpenDCRE is running on a given <ipaddress> and <port>. The default port for OpenDCRE is TCP port 5000. Currently, all commands are GET requests; a future version will expose these commands via POST as well.

1.4.1 Scan

Description

- The `scan` command polls boards and devices attached to the board. The `scan` command takes a `board_id` as its argument, and returns an array of board and device descriptors. If no `board_id` is provided, the `scan` command scans all boards on the device bus.

Notes

- It is likely a good idea for applications to scan for all boards on startup, to ensure a proper map of boards and devices is available to the application. Mismatches of board and device types and identifiers will result in 500 errors being returned for various commands that rely on these values mapping to actual hardware.

Request Format

- Scan devices on a specific `board_id`:

```
http://<ipaddress>:<port>/opendcre/<version>/scan/<board_id>
```

- Scan all boards on the device bus:

```
http://<ipaddress>:<port>/opendcre/<version>/scan
```

Parameters

- `board_id` (optional) : Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper 3 bytes of `board_id` are reserved for future use in OpenDCRE v1.1. IPMI Bridge board has a special `board_id` of 40000000.

Request Example

```
http://opendcre:5000/opendcre/1.1/scan
```

Response Schema


```
{
    "$schema": "http://schemas.vapor.io/opendcre/v1.1/opendcre-1.1-boards-devices",
    "title": "OpenDCRE Boards and Devices",
    "type": "object",
    "properties": {
        "boards": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "board_index": {
                        "type": "string"
                    },
                    "devices": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "device_id": {
                                    "type": "string"
                                },
                                "device_type": {
                                    "type": "string",
                                    "enum": [
                                        "temperature",
                                        "thermistor",
                                        "humidity",
                                        "led",
                                        "ipmb",
                                        "power",
                                        "door_lock",
                                        "current",
                                        "pressure",
                                        "mone"
                                    ]
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```


Example Response


```
{
  "boards": [
    {
      "board_id": "00000001",
      "devices": [
        {
          "device_id": "01ff",
          "device_type": "thermistor"
```

```
        },
        {
          "device_id": "02ff",
          "device_type": "none"
        }
      ]
    },
    {
      "board_id": "00000002",
      "devices": [
        {
          "device_id": "01ff",
          "sensor_type": "thermistor"
        },
        {
          "device_id": "02ff",
          "device_type": "none"
        }
      ]
    }
  ]
}
```

Errors

- Returns error (500) if scan command fails, or if `board_id` corresponds to an invalid `board_id`.

1.4.2 Version

Description

Return version information about a given board given its `board_id`.

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/version/<board_id>
```

Parameters

`board_id` : Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper 3 bytes of `board_id` are reserved for future use in OpenDCRE v1.1. IPMI Bridge board has a special `board_id` of 40000000.

Request Example

```
https://opendcre:5000/opendcre/1.0/version/00000001
```

Response Schema

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.1/opendcre-1.1-version",
  "title": "OpenDCRE Board Version",
  "type": "object",
  "properties": {
    "api_version": {
      "type": "string"
    },
    "firmware_version": {
      "type": "string"
    },
    "opendcre_version": {
      "type": "string"
    }
  }
}
```

Example Response

```
{
  "api_version": "1.1",
  "firmware_version": "OpenDCRE Emulator v1.1.0",
  "opendcre_version": "1.1.0"
}
```

Errors

Returns error (500) if version retrieval does not work or if `board_id` specifies a nonexistent board.

1.4.3 Read Device

Description

- Read a value from the given `board_id` and `device_id` for a specific `device_type`. The specified `device_type` must match the actual physical device type (as reported by the `scan` command), and is used to return a translated raw reading value (e.g. temperature in C for a thermistor) based on the existing algorithm for a given sensor type. The raw value is also returned.

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/read/<device_type>/<board_id>/<device_id>
```

Parameters

- **`device_type`: String value (lower-case) indicating what type of device to read:**
  - thermistor
  - temperature (not implemented yet)
  - current (not implemented yet)

  - humidity (not implemented yet)
  - led (not implemented yet)
  - ipmb (not implemented yet)
  - door_lock (not implemented yet)
  - pressure (not implemented yet)
  - none (**Note**: reading a "none" device will result in a 500 error)

- `board_id` : Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper 3 bytes of `board_id` are reserved for future use in OpenDCRE v1.1. IPMI Bridge board has a special `board_id` of 40000000.
- `device_id` : The device to read on the specified board. Hexadecimal string representation of a 2-byte integer value - range 0000..FFFF. Must be a valid, existing device, where the `device_type` known to OpenDCRE matches the `device_type` specified in the command for the given device - else, a 500 error is returned.

Request Example

```
http://opendcre:5000/opendcre/1.1/read/thermistor/00000001/01FF
```

Response Schema

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.1/opendcre-1.1-thermistor-reading",
  "title": "OpenDCRE Thermistor Reading",
  "type": "object",
  "properties": {
    "sensor_raw": {
      "type": "number"
    },
    "temperature_c": {
      "type": "number"
    }
  }
}
```

Example Response

```
{
  "sensor_raw": 755,
  "temperature_c": 19.73
}
```

Errors

- If a sensor is not readable or does not exist, an error (500) is returned.

1.4.4 Read Asset Info

Description

- Read asset information from the given `board_id` and `device_id` for a specific `device_type`. The specified `device_type` must match the actual physical device type (as reported by the `scan` command), and is used to return asset information (e.g. IP address, MAC address, Asset Tag, etc.) about a given device. Only devices of `device_type` of `power` support retrieval of asset information; IPMI `power` devices support read of asset information, but do not support write of asset information.

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/read/<device_type>/<board_id>/<device_id>/info
```

Parameters

- **`device_type`: String value (lower-case) indicating what type of device to read:**
  - power (**Note**: all other device types unsupported in this version of OpenDCRE).
- `board_id` : Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper 3 bytes of `board_id` are reserved for future use in OpenDCRE v1.1. IPMI Bridge board has a special `board_id` of 40000000. IPMI BMC asset information is readable, but not writeable.
- `device_id` : The device to read asset information for on the specified board. Hexadecimal string representation of a 2-byte integer value - range 0000..FFFF. Must be a valid, existing device, where the `device_type` known to OpenDCRE matches the `device_type` specified in the command for the given device - else, a 500 error is returned.

Request Example

```
http://opendcre:5000/opendcre/1.1/read/power/00000001/01FF/info
```

Response Schema

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.1/opendcre-1.1-asset-info-reading",
  "title": "OpenDCRE Asset Info Reading",
  "type": "object",
  "properties": {
    "board_id": {
      "type": "string"
    },
    "device_id: {
      "type": "string"
    },
    "asset_info: {
      "type": "string"
    }
  }
}
```

Alternately, for IPMI devices:

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.1/opendcre-1.1-asset-info-reading",
  "title": "OpenDCRE Asset Info Reading",
  "type": "object",
  "properties": {
    "board_id": {
      "type": "string"
    },
    "device_id: {
      "type": "string"
    },
    "asset_info: {
      "type": "string"
    },
    "bmc_ip: {
      "type": "string"
    }
  }
}
```

Example Response

```
{
  "board_id": "00000001",
  "device_id": "01FF",
  "asset_info": "example asset information"
}
```

Alternately, for IPMI devices:

```
{
  "board_id": "00000001",
  "device_id": "01FF",
  "asset_info": "example IPMI asset information",
  "bmc_ip": "123.124.10.100"
}
```

Errors

- asset info is not readable or does not exist, an error (500) is returned.

1.4.5 Write Asset Info

Description

- Write asset information from the given `board_id` and `device_id` for a specific `device_type`. The specified `device_type` must match the actual physical device type (as reported by the `scan` command), and is used to set asset information (e.g. IP address, MAC address, Asset Tag, etc.) for a given device. Only devices of `device_type` of `power` support set and retrieval of asset information; IPMI `power` devices support read of asset information, but do not support write of asset information. Attempting to write asset information for an IPMI device will result in a 500 error.

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/write/<device_type>/<board_id>/<device_id>/info/<value>
```

Parameters

- **`device_type`: String value (lower-case) indicating what type of device to write asset info for:**
  - power (**Note**: all other device types unsupported in this version of OpenDCRE).
- `board_id` : Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper 3 bytes of `board_id` are reserved for future use in OpenDCRE v1.1. IPMI Bridge board has a special `board_id` of 40000000. IPMI BMC asset information is readable, but not writeable.
- `device_id` : The device to read asset information for on the specified board. Hexadecimal string representation of a 2-byte integer value - range 0000..FFFF. Must be a valid, existing device, where the `device_type` known to OpenDCRE matches the `device_type` specified in the command for the given device - else, a 500 error is returned.
- `value` : The string value to set for asset information for the given device. Max length of this string value is 127 bytes. Overwrites any value previously stored in the `asset_info` field.

Request Example

```
http://opendcre:5000/opendcre/1.1/write/power/00000001/01FF/info/192.100.10.1
```

Response Schema

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.1/opendcre-1.1-asset-info-response",
  "title": "OpenDCRE Asset Info Response",
  "type": "object",
  "properties": {
    "board_id": {
      "type": "string"
    },
    "device_id: {
      "type": "string"
    },
    "asset_info: {
      "type": "string"
    }
  }
}
```

Example Response

```
{
  "board_id": "00000001",
  "device_id": "01FF",
  "asset_info": "example asset information"
}
```

Errors

- If asset info is not writeable or does not exist, an error (500) is returned.

1.4.6 Write Device

Description

- Write to device bus to a writeable device. The write command is followed by the `device_type`, `board_id` and `device_id`, with the final field of the request being the data sent to the device.

Status

- Not yet implemented.

1.4.7 Power

Description

- Control device power, and/or retrieve its power supply status.

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/power/<command>/<board_id>/<device_id>
```

Parameters

- `board_id` : Hexadecimal string representation of 4-byte integer value - range 00000000..FFFFFFFF. Upper 3 bytes of `board_id` are reserved for future use in OpenDCRE v1.1. IPMI Bridge board has a special `board_id` of 40000000.
- `device_id` : The device to issue power command to on the specified board. Hexadecimal string representation of 2-byte integer value - range 0000..FFFF. Must be a valid, existing device, where the `device_type` known to the OpenDCRE HAT is `power` - else, a 500 error is returned.
- **`command:`**
  - `on` : Turn power on to specified device.
  - `off` : Turn power off to specified device.
  - `cycle` : Power-cycle the specified device.
  - `status` : Get power status for the specified device.

For all commands, power status is returned as the command's response.

Request Example

```
http://opendcre:5000/opendcre/1.1/power/on/00000001/01ff
```

Response Schema


```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.1/opendcre-1.1-power-status",
  "title": "OpenDCRE Power Status",
  "type": "object",
  "properties": {
    "input_power": {
      "type": "number"
    },
    "input_voltage": {
      "type": "number"
    },
    "output_current": {
      "type": "number"
    },
    "over_current": {
      "type": "boolean"
    },
    "pmbus_raw": {
      "type": "string"
    },
    "power_ok": {
      "type": "boolean"
    },
    "power_status": {
      "type": "string"
    },
    "under_voltage": {
      "type": "boolean"
    }
  }
}
```


Example Response


```
{
  "input_power": 0.0,
  "input_voltage": 0.0,
  "output_current": -25.70631970260223,
  "over_current": false,
  "pmbus_raw": "0,0,0,0",
  "power_ok": true,
  "power_status": "on",
  "under_voltage": false
}
```


Errors

- If a power action fails, or an invalid board/device combination are specified, an error (500) is returned.

1.4.8 Test

Description

- The test command may be used to verify that the OpenDCRE endpoint is up and running, but without attempting to address the device bus. The command takes no arguments, and if successful, returns a simple status message of "ok".

Request Format

```
http://<ipaddress>:<port>/opendcre/<version>/test
```

Response Schema

```
{
  "$schema": "http://schemas.vapor.io/opendcre/v1.1/opendcre-1.1-test-status",
  "title": "OpenDCRE Test Status",
  "type": "object",
  "properties": {
    "status": {
      "type": "string"
    }
  }
}
```

Example Response

```
{
  "status": "ok"
}
```

Errors

- If the endpoint is not running no response will be returned, as the command will always return the response above while the endpoint is functional.

1.5 OpenDCRE / OpenMistOS Release Notes

1.5.1 OpenDCRE v1.1.0 Release Notes

November 16, 2015

Included in this release:

- OpenDCRE power, sensor, version, scan commands via PLC.
- Simplified device_id (2-byte) and board_id (4-byte) scheme.
- IPMI 1.5 Bridge and power control support.
- Additional testing, documentation.
- OpenDCRE UI demo, for evaluation and development purposes.

- OpendDCRE on GitHub.