ROCm™ Data Center Tool™ User Guide

Revision:	**0802**
Issue Date:	**August 2021**

Table of Contents

Chapter 1 Overview

1.1 ROCm™ Data Center Tool

The ROCm™ Data Center Tool™ simplifies the administration and addresses key infrastructure challenges in AMD GPUs in cluster and datacenter environments. The main features are:

- GPU telemetry
- GPU statistics for jobs
- Integration with third-party tools
- Open source

The tool can be used in stand-alone mode if all components are installed. However, the existing management tools can use the same set of features available in a library format.

Refer *Starting RDC* for details on different modes of operation.

1.1.1 Objective

This user guide is intended to:

- Provide an overview of the ROCm Data Center Tool features
- Describe how system administrators and Data Center (or HPC) users can administer and configure AMD GPUs
- Describe the components
- Provide an overview of the open source developer handbook

1.1.2 Terminology

Term	Description
RDC	ROCm^{TM} Data Center Tool
Compute node (CN)	One of many nodes containing one or more GPUs in the Data Center on which compute jobs are run
Management node (MN) or Main console	A machine running system administration applications to administer and manage the Data Center
GPU Groups	Logical grouping of one or more GPUs in a compute node
Fields	A metric that can be monitored by the RDC, such as GPU temperature, memory usage, and power usage
Field Groups	Logical grouping of multiple fields
Job	A workload that is submitted to one or more compute nodes

1.1.3 Target Audience

The audience for the AMD ROCm Data Center™ tool consists of:

- **Administrators:** The tool will provide cluster administrator with the capability of monitoring, validating, and configuring policies.
- **HPC Users:** Provides GPU centric feedback for their workload submissions
- **OEM**: Add GPU information to their existing cluster management software
- **Open Source Contributors**: RDC is open source and will accept contributions from the community

For more information about the API, see the *AMD ROCm Data Center API Guide at*

https://github.com/RadeonOpenCompute/ROCm/blob/master/AMD_RDC_API_Guide_v4.3.pdf

Chapter 2 Installation and Integration

2.1 Supported platforms

The ROCm Data Center Tool™ (RDC) is part of the AMD ROCm™ software and is available on the distributions supported by AMD ROCm.

- Ubuntu 18.04.4 (Kernel 5.3)
  Note: In the AMD ROCm v3.10 release, pre-built packages are only available for Ubuntu.

- CentOS v7.7 (Using devtoolset-7 runtime support)
- RHEL v7.7 (Using devtoolset-7 runtime support)
- SLES 15 SP1
- CentOS and RHEL 8.1(Kernel 4.18.0-147)

2.2 Prerequisites

For RDC installation from prebuilt packages, follow the instructions in this section.

2.2.1 Install gRPC

The following components are required as RDC relies on them for communication and authentication:

1.
    - gRPC along with protoc
    - protoc plugins

For more information on gRPC, see the gRPC GitHub locations at:

- *https://github.com/grpc/grpc/blob/v1.25.0/src/cpp/README.md*
- *https://github.com/grpc/grpc/blob/master/BUILDING.md*

Note: CMake 3.15 or greater is required to build gRPC

```
$ sudo apt-get install -y automake make g++ unzip
$ sudo apt-get install -y build-essential autoconf libtool pkg-config
$ sudo apt-get install -y libgflags-dev libgtest-dev
$ sudo apt-get install -y clang-5.0 libc++-dev curl
$ git clone -b v1.28.1 https://github.com/grpc/grpc
$ cd grpc
$ git submodule update --init
$ mkdir -p cmake/build
$ cd cmake/build
# By default (without using the CMAKE_INSTALL_PREFIX option), the following
will install

# to /usr/local lib, include and bin directories

$ cmake -DgRPC_INSTALL=ON \
 -DBUILD_SHARED_LIBS=ON \
 <-DCMAKE_INSTALL_PREFIX=<install dir>> ../..


$ make
$ sudo make install
$ sudo ldconfig
```

2.2.1.1 Authentication keys

The ROCm Data Center (RDC)™ tool can be used with or without authentication. If authentication is required, then proper authentication keys need to be configured.

On how to configure SSL keys, refer to section 5.2 *Authentication* in this document.

2.2.2 Pre-built Packages

The RDC tool is packaged as part of the ROCm software repository. You must install the AMD ROCm software before installing RDC. For details on how to install ROCm, see the AMD ROCm Installation Guide.

Follow the instructions below to install RDC after installing the ROCm package:

2.2.2.1 Ubuntu

```
$ sudo apt-get install rdc
# to install a specific version
$ sudo apt-get install rdc<x.y.z>
```

2.2.2.2 SLES 15 Service Pack I

In the AMD ROCm v3.10 release, the pre-built package is not available for SLES 15 Service Pack (SP) 1. You must build and install the ROCm Data Center (RDC) tool from the source.

NOTE: By default (without using the CMAKE_INSTALL_PREFIX option), the following installations will occur in the */usr/local lib, include, and bin* directories.

To build the RDC tool from source:

1. Build and install gRPC.

```
$  git clone -b v1.28.1 https://github.com/grpc/grpc && cd grpc

$ git submodule update --init

$ mkdir -p cmake/build && cd cmake/build/

$ cmake -DgRPC_INSTALL=ON \
  -DBUILD_SHARED_LIBS=ON \
  <-DCMAKE_INSTALL_PREFIX=<gRPC install dir>> ../..

$ make

$ sudo make install

$ echo "<gRPC install dir>/lib" | sudo tee /etc/ld.so.conf.d/grpc.conf
```

2. Build and install the ROCm Data Center Tool (RDC).

   $ git clone https://github.com/RadeonOpenCompute/rdc && cd rdc

   $ mkdir -p build && cd build

   *$ cmake -DROCM_DIR=/opt/rocm \\
     -DGRPC_ROOT=<gRPC install dir> \\
     <-DCMAKE_INSTALL_PREFIX=< RDC install dir>> ../..*

   $ make

   $ sudo make install

3. Update the system library path.

   NOTE: The following commands must be executed as root (sudo). It is recommended to insert the commands into a script and run the script as root.

   $ RDC_LIB_DIR=<RDC install dir>/lib

   $ GRPC_LIB_DIR=<gRPC install dir>/lib

   $ echo "$GRPC_LIB_DIR" > /etc/ld.so.conf.d/x86_64-librdc_client.conf

   $ echo "$GRPC_LIB_DIR"64 >> /etc/ld.so.conf.d/x86_64-librdc_client.conf

   $ echo "$RDC_LIB_DIR" >> /etc/ld.so.conf.d/x86_64-librdc_client.conf

   $ echo "$RDC_LIB_DIR"64 >> /etc/ld.so.conf.d/x86_64-librdc_client.conf

   $ ldconfig

2.3 Components

The ROCm Data Center Tool™ components are as follows:

RDC (API) Library
This library is the central piece, by interacting with different modules, that provides all the features described. This shared library provides C API and Python bindings so that third-party tools should be able to use it directly if required.

RDC Daemon (rdcd)
The daemon will record telemetry information from GPUs. It will also provide an interface to RDC command-line tool (rdci) running locally or remotely. It depends on the above RDC Library for all the core features.

RDC Command Line Tool (rdci)
A command-line tool to invoke all the features of the RDC tool. This CLI can be run locally or remotely.

ROCm-SMI library
A stateless system management library provides low-level interfaces to access GPU information.

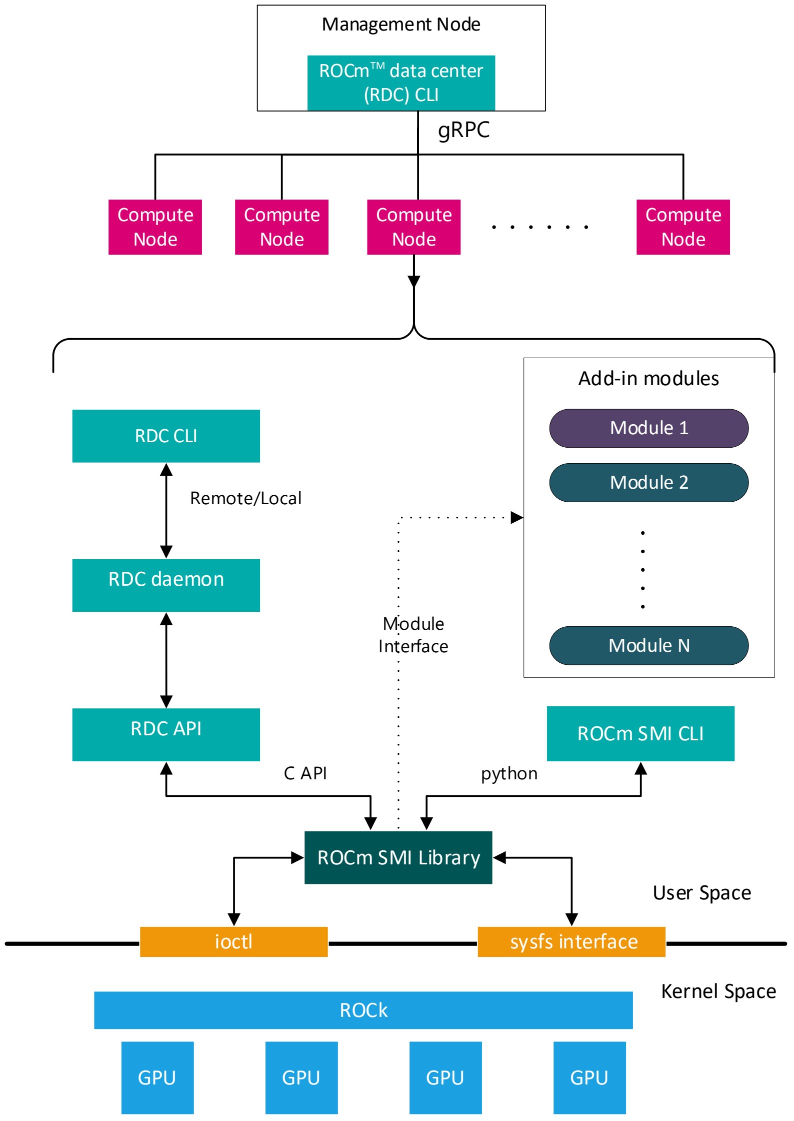


Figure 1 High-level diagram of RDC components

2.4 Starting ROCm Data Center Tool

The ROCm Data Center (RDC) Tool™ can be run in the following two modes. Note, the feature set is similar in both cases. Users have the flexibility to choose the right option that best fits their environment.

- Standalone
- Embedded mode

The capability in each mode depends on the privileges the *user* has for starting RDC. A normal *user* will have access only to monitor (access to GPU telemetry) capabilities. A *privileged user* can run the tool with full capability. In the full capability mode, GPU configuration features can be invoked. This may or may not affect all the users and processes sharing the GPU.

2.4.1 Standalone mode

This is the preferred mode of operation as it does not have any external dependencies. To start RDC in stand-alone mode, RDC Server Daemon (rdcd) must run on each compute node. You can start RDC daemon (rdcd) as a systemd service or directly from the command-line.

2.4.1.1 Start RDC using systemd

The capability of RDC can be configured by modifying the rdc.service system configuration file. Use the *systemctl* command to start *rdcd.*

```
$ systemctl start rdc
```

By default, *rdcd* starts with full capability. To change to monitor only comment out the following two lines.

```
$ sudo vi /lib/systemd/system/rdc.service

# CapabilityBoundingSet=CAP_DAC_OVERRIDE
# AmbientCapabilities=CAP_DAC_OVERRIDE
```

NOTE: *rdcd* can be started by using the *systemctl* command

```
$ systemctl start rdc
```

If the GPU reset fails, restart the server. Note, restarting the server also initiates *rdcd*. Users may, then, encounter the following two scenarios:

- *rdcd* returns the correct GPU information to *rdci*
- *rdcd* returns the "No GPUs found on the system" error to *rdci.* To resolve this error, restart *rdcd* with the following instruction:

```
         sudo systemctl restart rdcd
```

2.4.1.2 Start ROCm Data Center Tool™ from command-line

While *systemctl* is the preferred way to start *rdcd*, you can also start directly from the command-line. The installation scripts will create a default user - *"rdc"*. Users have the option to edit the profile file (rdc.service installed at /lib/systemd/system) and change these lines accordingly:

```
[Service]
User=rdc
Group=rdc
```

```
#Start as user rdc
$ sudo -u rdc rdcd

# Start as root
$ sudo rdcd
```

From the command-line, start *rdcd* as a *user* (for example, *rdc*) or *root*.

Note, in this use case, the *rdc.service* file mentioned in the previous section is not involved. Here, the capability of RDC is determined by the privilege of the user starting *rdcd*. If *rdcd* is running under a normal user account, then it has the Monitor-only capability. If *rdcd* is running as root, then *rdcd* has Full capability.

NOTE: If a user other than rdc or root starts the rdcd daemon, the file ownership of the SSL keys mentioned in the *Authentication*Authentication section must be modified to allow read and write access.

2.4.1.3 Troubleshooting rdcd

When *rdcd* is started using systemctl, the logs can be viewed using the following command.

```
$ journalctl -u rdc
```

These messages provide useful status and debugging information. The logs can also help debug problems like rdcd failing to start, communication issues with a client, and others.

2.4.2 Embedded mode

The embedded mode is useful if the end user has a monitoring agent running on the compute node. The monitoring agent can directly use the RDC library and will have a finer grain control on how and when RDC features are invoked. For example, if the monitoring agent has a facility to synchronize across multiple nodes, then it can synchronize GPU telemetry across these nodes.

The RDC daemon rdcd can be used as a reference code for this purpose. The dependency on gRPC is also eliminated if the RDC library is directly used.

CAUTION: RDC command-line rdci will not function in this mode. Third-party monitoring software is responsible for providing the user interface and remote access/monitoring. Refer to *ROCm™ Data Center API Guide (Alpha Release)* for API details and pseudocode.

Chapter 3 Feature Overview

Note, ROCm™ Data Center Tool™ is in active development. This section highlights the current feature set.

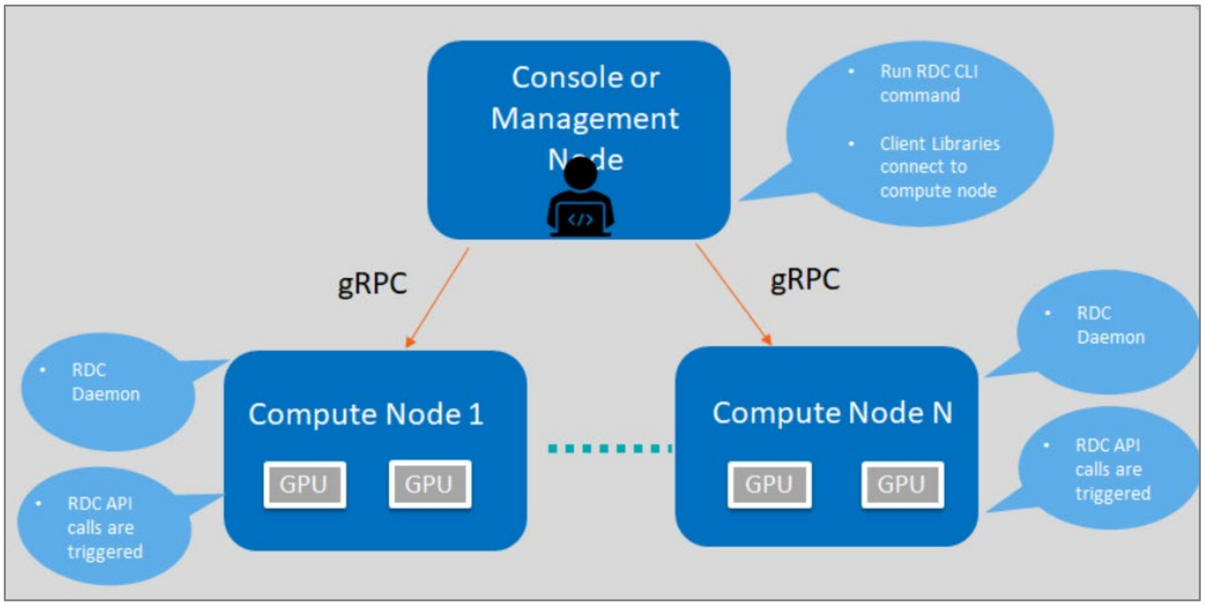

Console or Management Node
Run RDC CLI command
Client Libraries connect to compute node
gRPC
gRPC
RDC Daemon
Compute Node 1
GPU
GPU
Compute Node N
GPU
GPU
RDC Daemon
RDC API calls are triggered
RDC API calls are triggered

Figure 2 Shows RDC components and framework for describing features

3.1 Discovery

The Discovery feature enables you to locate and display information of GPUs present in the compute node. For example,

```
$ rdci discovery <host_name> -l

 2 GPUs found

 GPU Index        Device Information

 0                Name: AMD Radeon Instinct™ MI50 Accelerator
 1                Name: AMD Radeon Instinct™ MI50 Accelerator

$ rdci -l : list available GPUs
$ rdci -u: No SSL authentication
```

3.2 Groups

3.2.1 GPU Groups

With the GPU groups feature, you can create, delete, and list logical groups of GPU.

```
$ rdci group -c GPU_GROUP
Successfully created a group with a group ID 1

$ rdci group -g 1 -a 0,1
Successfully added the GPU 0,1 to group 1

$ rdci group -l

1 group found

┌──────────────┬─────────────────────────────┬─────────────────────────────┐
│ Group ID     │ Group Name                  │ GPU Index                   │
├──────────────┼─────────────────────────────┼─────────────────────────────┤
│ 1            │ GPU_GROUP                   │  0,1                        │
└──────────────┴─────────────────────────────┴─────────────────────────────┘

$ rdci group -d 1
Successfully removed group 1

-c create; -g group id; -a add GPU index; -l list; -d delete group
```

3.2.2 Field Groups

The Field groups feature provides you the options to create, delete, and list field groups.

```
$ rdci fieldgroup -c <fgroup> -f 150,155
Successfully created a field group with a group ID 1

$ rdci fieldgroup -l

1 group found

┌──────────────┬─────────────────────────────┬─────────────────────────────┐
│ Group ID     │ Group Name                  │ Field Ids                   │
├──────────────┼─────────────────────────────┼─────────────────────────────┤
│ 1            │ fgroup                      │ 150,155                     │
└──────────────┴─────────────────────────────┴─────────────────────────────┘

$ rdci fieldgroup -d 1
Successfully removed field group 1

rdci dmon -l
Supported fields Ids:
100 RDC_FI_GPU_CLOCK:        Current GPU clock freq.
150 RDC_FI_GPU_TEMP:         GPU temp. in milli Celsius.
155 RDC_FI_POWER_USAGE:      Power usage in microwatts.
203 RDC_FI_GPU_UTIL:         GPU busy percentage.
525 RDC_FI_GPU_MEMORY_USAGE: VRAM Memory usage in bytes

-c create; -g group id; -a add GPU index; -l list; -d delete group
```

```

```

3.2.2.1 Monitoring Errors

You can define the *RDC_FI_ECC_CORRECT_TOTAL* or *RDC_FI_ECC_UNCORRECT_TOTAL* field to get the RAS Error-Correcting Code (ECC) counter:

- 312 RDC_FI_ECC_CORRECT_TOTAL: Accumulated correctable ECC errors
- 313 RDC_FI_ECC_UNCORRECT_TOTAL: Accumulated uncorrectable ECC errors

3.2.3 Device Monitoring

The ROCm Data Center tool™ enables you to monitor GPU fields.

```
$ rdci dmon -f <field_group> -g <gpu_group> -c 5 -d 1000


1 group found
┌───────────────┬──────────────────────────────┬──────────────────────────────┐
│ GPU Index     │ TEMP (m°C)                   │ POWER (µW)                   │
│               │                              │                              │
├───────────────┼──────────────────────────────┼──────────────────────────────┤
│ 0             │ 25000                        │ 520500                       │
├───────────────┼──────────────────────────────┼──────────────────────────────┤
│ 0             │ 25000                        │ 520500                       │
├───────────────┼──────────────────────────────┼──────────────────────────────┤
│ 0             │ 25000                        │ 520500                       │
├───────────────┼──────────────────────────────┼──────────────────────────────┤
│ 0             │ 25000                        │ 520500                       │
└───────────────┴──────────────────────────────┴──────────────────────────────┘

rdci dmon -l
Supported fields Ids:
100 RDC_FI_GPU_CLOCK:       Current GPU clock freq.
150 RDC_FI_GPU_TEMP:        GPU temp. in milli Celsius.
155 RDC_FI_POWER_USAGE:     Power usage in microwatts.
203 RDC_FI_GPU_UTIL:        GPU busy percentage.
525 RDC_FI_GPU_MEMORY_USAGE: VRAM Memory usage in bytes

-e field ids; -i GPU index; -c count; -d delay; -l list; -f fieldgroup id
```

3.2.4 Job Stats

You can display GPU statistics for any given workload.

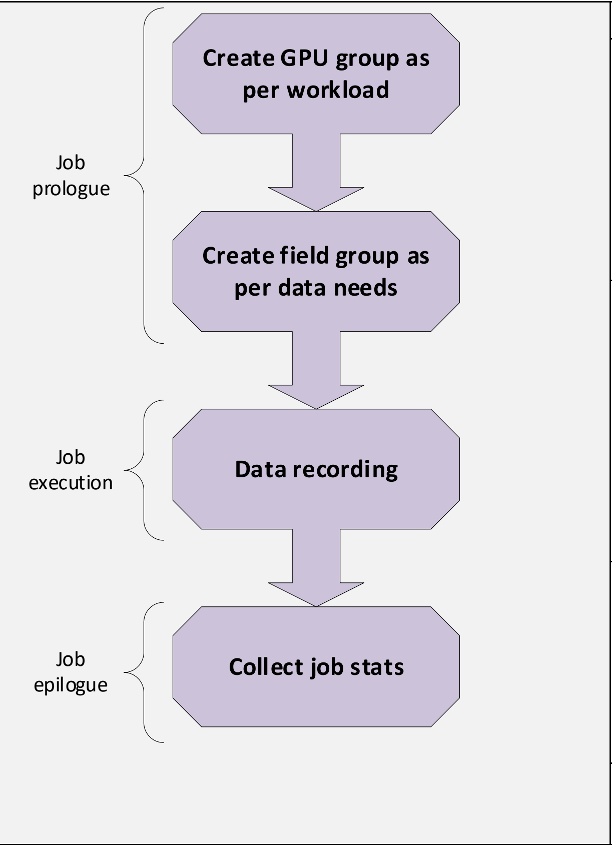
```
$ rdci stats -s 2 -g 1
Successfully started recording job 2 with a group ID 1

$ rdci stats -j 2

┌───────────────────────────────┬────────────────────────────────────────┐
│ Summary                       │                                        │
│ Executive Status              │                                        │
├───────────────────────────────┼────────────────────────────────────────┤
│ Start time                    │ 1586795401                             │
│ End time                      │ 1586795445                             │
│ Total execution time          │ 44                                     │
└───────────────────────────────┴────────────────────────────────────────┘
```

```
 ----------------------------------------------------------------------------
| Energy Consumed (Joules)       | 21682                                      |
| Power Usage (Watts)            | Max: 49 Min: 13 Avg: 34                    |
| GPU Clock (MHz)                | Max: 1000 Min: 300 Avg: 903                |
| GPU Utilization (%)            | Max: 69 Min: 0 Avg: 2                      |
| Max GPU Memory Used (bytes)    | 524320768                                  |
| Memory Utilization (%)         | Max: 12 Min: 11 Avg: 12                    |
 ----------------------------------------------------------------------------


$ rdci stats -x 2
Successfully stopped recording job 2

-s start recording on job id; -g group id; -j display job stats; -x stop
recording
```

3.2.4.1 Job stats use case

A common use case is to record GPU statistics associated with any job or workload. The following example shows how all these features can be put together for this use case.

	rdci commands

 Create GPU group as per workload
Create field group as per data needs
Data recording
Collect job stats
Job prologue
Job execution
Job epilogue
	`$ rdci group -c group1`
	`Successfully created a group with a group ID 1`
	`$ rdci group -g 1 -a 0,1`
	`GPU 0,1 is added to group 1 successfully`

	`rdci stats -s 123 -g 1`
	`job 123 recorded successfully with the group ID`

	`rdci stats -x 123`
	`Job 123 stops recording successfully`

	`rdci stats -j 123`
	`job stats printed`

Table 1 An example showing how job statistics can be recorded

3.2.4.2 Error-Correcting Code Output

In the job stats output, this feature prints out the Error-Correcting Code (ECC) errors while running the job.

Chapter 4 Third-Party Integration

4.1 Python[1] Bindings

The ROCm™ Data Center Tool™ (RDC) provides a generic python class *RdcReader* to simplify telemetry gathering. *RdcReader* simplifies usage by providing the following functionalities.

- The user only needs to specify telemetry fields. *RdcReader* will create the necessary groups and fieldgroups, watch the fields, and fetch the fields.
- The *RdcReader* can support embedded and standalone mode. The standalone can be with or without authentication.
- In standalone mode, the *RdcReader* can automatically reconnect to *rdcd* if the connection is lost.
- When *rdcd* is restarted, the previously created group and fieldgroup may be lost. The *RdcReader* can re-create them and watch the fields after reconnecting.
- If the client is restarted, *RdcReader* can detect the groups and fieldgroups created before and avoid re-creating them.
- A custom unit converter can be passed to *RdcReader* to override the default RDC unit.

See below for a sample program to monitor the power and GPU utilization using the *RdcReader*.

```
from RdcReader import RdcReader
from RdcUtil import RdcUtil
from rdc_bootstrap import *

default_field_ids = [
        rdc_field_t.RDC_FI_POWER_USAGE,
        rdc_field_t.RDC_FI_GPU_UTIL
]

class SimpleRdcReader(RdcReader):
    def __init__(self):
        RdcReader.__init__(self,ip_port=None, field_ids = default_field_ids, 
update_freq=1000000)
    def handle_field(self, gpu_index, value):
        field_name = self.rdc_util.field_id_string(value.field_id).lower()
        print("%d %d:%s %d" % (value.ts, gpu_index, field_name, value.value.l
_int))

if __name__ == '__main__':
    reader = SimpleRdcReader()
```

[1] "Python" is a registered trademark of Python Software Foundation

```
    while True:
        time.sleep(1)
        reader.process()
```

2.
In the sample program,

- class *SimpleRdcReader* is derived from the *RdcReader*.
- The field "ip_port=None" in *RdcReader* dictates that the ROCm Data Center tool runs in the embedded mode.
- *SimpleRdcReader::process(),* then, fetches fields specified in *default_field_ids*. *RdcReader.py* can be found in the *python_binding* folder located at RDC install path.

To run the example, use

```
# Ensure that RDC shared libraries are in the library path and
# RdcReader.py is in PYTHONPATH

$ python SimpleReader.py
```

4.2 Prometheus Plugin

4.2.1 Prometheus Plugin Installation

The ROCm™ Data Center Tool™ (RDC) Prometheus plugin *rdc_prometheus.py* can be found in the *python_binding* folder.

NOTE: Ensure the *Prometheus client* is installed before the Prometheus plugin installation process.

```
$ pip install prometheus_client
```

The options the plugin provides can be viewed with *–help*.

```
% python rdc_prometheus.py --help
usage: rdc_prometheus.py [-h] [--listen_port LISTEN_PORT] [--rdc_embedded]
                         [--rdc_ip_port RDC_IP_PORT] [--rdc_unauth]
                         [--rdc_update_freq RDC_UPDATE_FREQ]
                         [--rdc_max_keep_age RDC_MAX_KEEP_AGE]
                         [--rdc_max_keep_samples RDC_MAX_KEEP_SAMPLES]
                         [--rdc_fields RDC_FIELDS [RDC_FIELDS ...]]
                         [--rdc_fields_file RDC_FIELDS_FILE]
                         [--rdc_gpu_indexes RDC_GPU_INDEXES [RDC_GPU_INDEXES ...]]
                         [--enable_plugin_monitoring]

RDC Prometheus plugin.

optional arguments:
  -h, --help            show this help message and exit
  --listen_port LISTEN_PORT
                        The listen port of the plugin (default: 5000)
  --rdc_embedded        Run RDC in embedded mode (default: standalone mode)
  --rdc_ip_port RDC_IP_PORT
                        The rdcd IP and port in standalone mode (default:
                        localhost:50051)
  --rdc_unauth          Set this option if the rdcd is running with unauth in
                        standalone mode (default: false)
  --rdc_update_freq RDC_UPDATE_FREQ
                        The fields update frequency in seconds (default: 10))
  --rdc_max_keep_age RDC_MAX_KEEP_AGE
                        The max keep age of the fields in seconds (default:
                        3600)
  --rdc_max_keep_samples RDC_MAX_KEEP_SAMPLES
                        The max samples to keep for each field in the cache
                        (default: 1000)
  --rdc_fields RDC_FIELDS [RDC_FIELDS ...]
                        The list of fields name needs to be watched, for
                        example, " --rdc_fields RDC_FI_GPU_TEMP
                        RDC_FI_POWER_USAGE " (default: fields in the
                        plugin)
  --rdc_fields_file RDC_FIELDS_FILE
                        The list of fields name can also be read from a file
                        with each field name in a separated line (default:
                        None)
  --rdc_gpu_indexes RDC_GPU_INDEXES [RDC_GPU_INDEXES ...]
                        The list of GPUs to be watched (default: All GPUs)
  --enable_plugin_monitoring
                        Set this option to collect process metrics of
                        the plugin itself (default: false)
```

By default, the plugin runs in the standalone mode and will connect to *rdcd* at localhost:50051 to fetch fields. The plugin should use the same authentication mode as *rdcd* i.e. if *rdcd* is running with *-u/--unauth* flag, then --rdc_unauth flag should be used by the plugin. The plugin can be used in the embedded mode without *rdcd,* by setting --rdc_embedded flag.

To override the default fields that are monitored, the *--rdc_fields* option can be used to specify the list of fields. If the fields list is long, then the *--rdc_fields_file* option provides a convenient way to fetch fields list from a file. The *max_keep_age* and *max_keep_samples* can be used to control how the fields are cached.

The plugin can provide the metrics of the plugin itself, including the plugin process CPU, memory, file descriptor usage, and native threads count, including the process start and up times. This can be enabled using *--enable_plugin_monitoring*.

You can test the plugin with the default settings.

```
# Ensure that rdcd is running on the same machine
$ python rdc_prometheus.py


# Check the plugin using curl
$ curl localhost:5000
   # HELP gpu_util gpu_util
# TYPE gpu_util gauge
gpu_util{gpu_index="0"} 0.0
# HELP gpu_clock gpu_clock
# TYPE gpu_clock gauge
gpu_clock{gpu_index="0"} 300.0
# HELP gpu_memory_total gpu_memory_total
# TYPE gpu_memory_total gauge
gpu_memory_total{gpu_index="0"} 4294.0
# HELP gpu_temp gpu_temp
# TYPE gpu_temp gauge
# HELP power_usage power_usage
# TYPE power_usage gauge
power_usage{gpu_index="0"} 9.0
# HELP gpu_memory_usage gpu_memory_usage
# TYPE gpu_memory_usage gauge
gpu_memory_usage{gpu_index="0"} 134.0
```

4.2.2 Prometheus Integration

1. Download and install Prometheus in the management machine. You can access it at *https://github.com/prometheus/prometheus*.

2. Use the example configuration file *rdc_prometheus_example.yml* in the *python_binding* folder. This file can be used in its original state, however, note that this file refers to *prometheus_targets.json*. Ensure this is modified to point to the correct compute nodes.

```
// Sample file: prometheus_targets.json

// Replace rdc_test*.amd.com to point the correct compute nodes

// Add as many compute nodes as necessary

[

  {

   "targets": [

     "rdc_test1.amd.com:5000",

     "rdc_test2.amd.com:5000"

   ]

  }

]
```

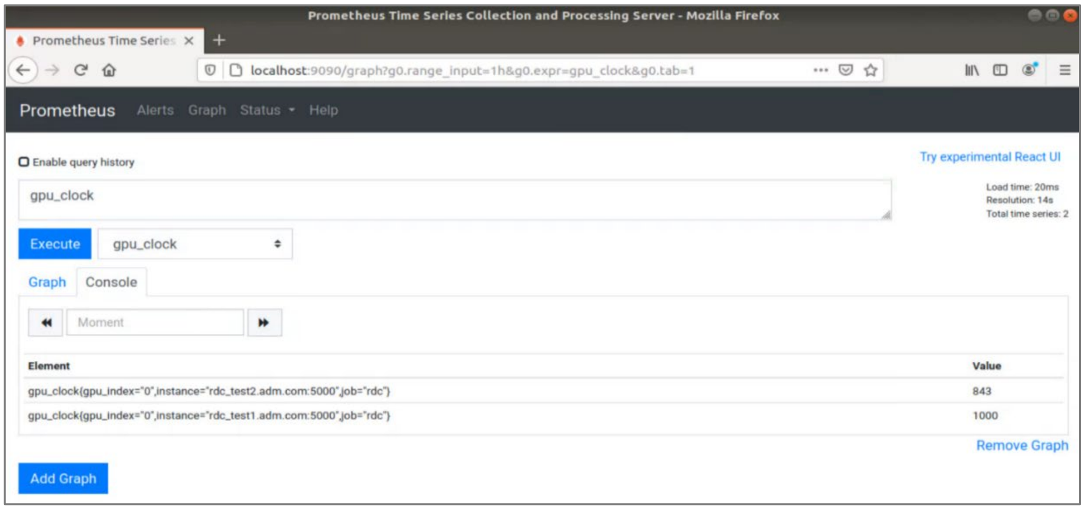
NOTE: In the above example, there are two compute nodes *rdc_test1.adm.com* and *rdc_test2.adm.com*. Ensure the Prometheus plugin is running on those compute nodes.

3. Start the Prometheus plugin.

```
% prometheus --config.file=<full path of the rdc_prometheus_example.yml>
```

4. From the management node, using a browser, open the URL *http://localhost:9090*.

5. Select one of the available metrics. For example, gpu_clock


Prometheus Time Series Collection and Processing Server - Mozilla Firefox
Prometheus Time Series
localhost:9090/graph?g0.range_input=1h&g0.expr=gpu_clock&g0.tab=1
Prometheus
Alerts
Graph
Status
Help
Enable query history
Try experimental React UI
Execute
Graph
Console
Moment
Remove Graph
Add Graph

gpu_clock

Load time: 20ms
Resolution: 14s
Total time series: 2

gpu_clock

Element	Value
gpu_clock{gpu_index="0",instance="rdc_test2.adm.com:5000",job="rdc"}	843
gpu_clock{gpu_index="0",instance="rdc_test1.adm.com:5000",job="rdc"}	1000

The Prometheus image shows the GPU clock for both rdc_test1 and rdc_test2.

4.3 Grafana Plugin

Grafana is a common monitoring stack used for storing and visualizing time series data. Prometheus acts as the storage backend and Grafana is used as the interface for analysis and visualization. Grafana has a plethora of visualization options and can be integrated with Prometheus for the ROCm Data Center (RDC) dashboard.

4.3.1 Grafana Plugin Installation

1. Download Grafana from *https://grafana.com/grafana/download*

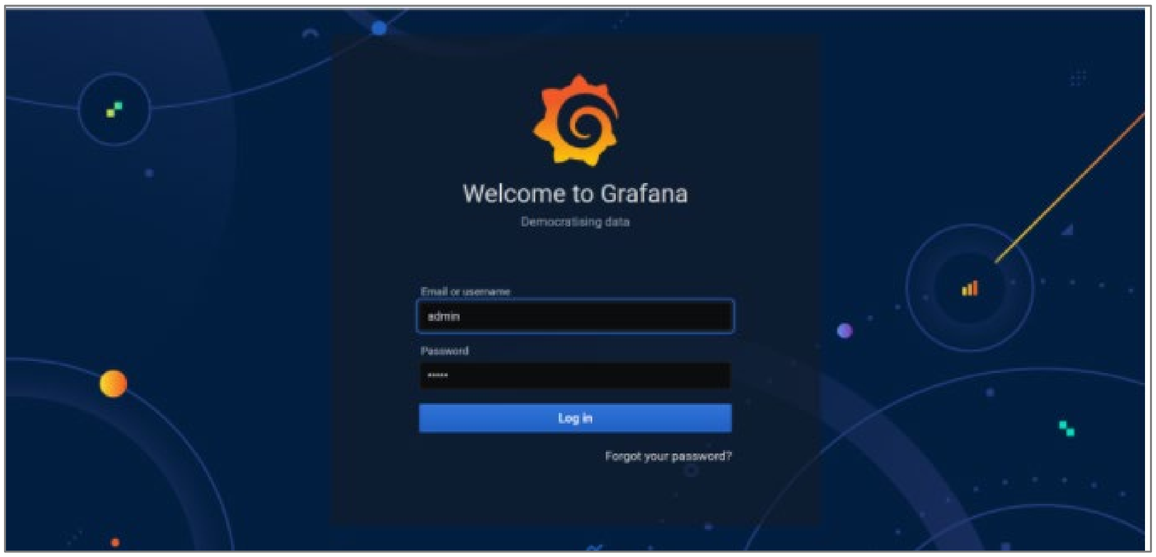
2. Install Grafana. You can access the installation instructions at *https://grafana.com/docs/grafana/latest/installation/debian/*

3. Use the following instructions to start Grafana:

```
   sudo systemctl start grafana-server
   sudo systemctl status grafana-server
```

4. Browse to *http://localhost:3000/*

5. Log in using the default user name and password (admin/admin) as shown in the image below:



4.3.2 Grafana Integration

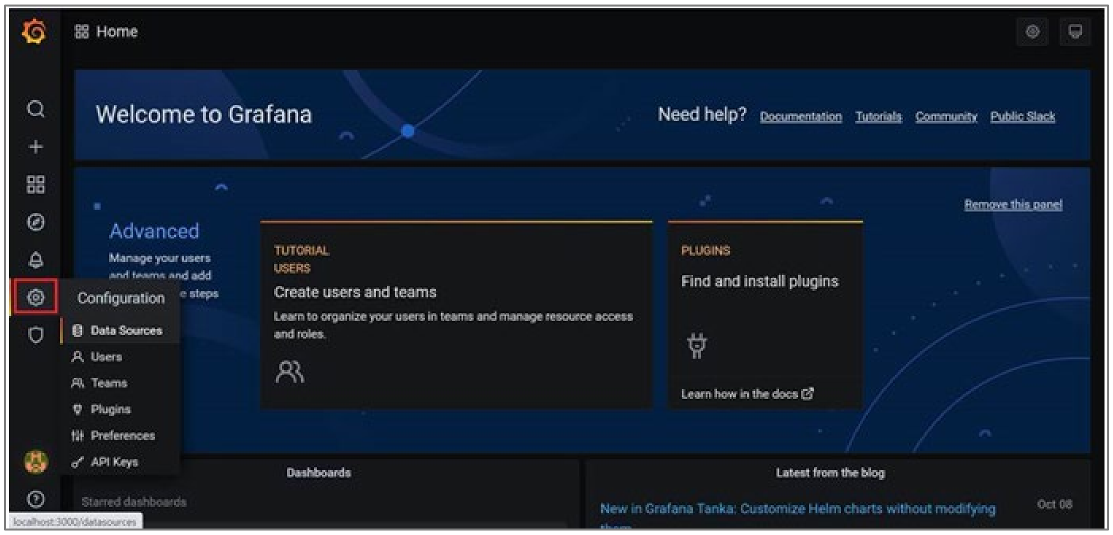
As a prerequisite, ensure:

- ROCm Data Center Prometheus plugin is running in each compute node
- Prometheus is set up to collect metrics from the plugin

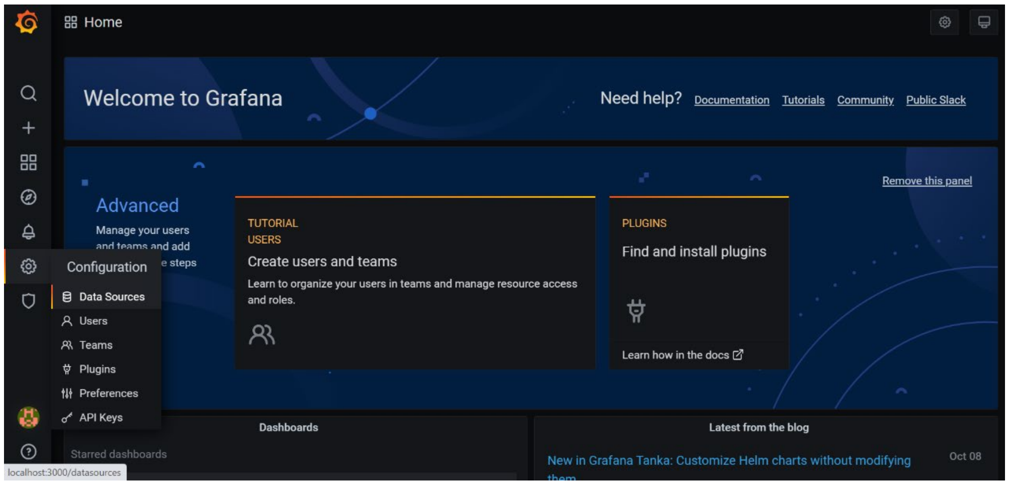
For more information about installing and configuring Prometheus, see the section on *Prometheus Plugin*.

4.3.2.1 Grafana Configuration

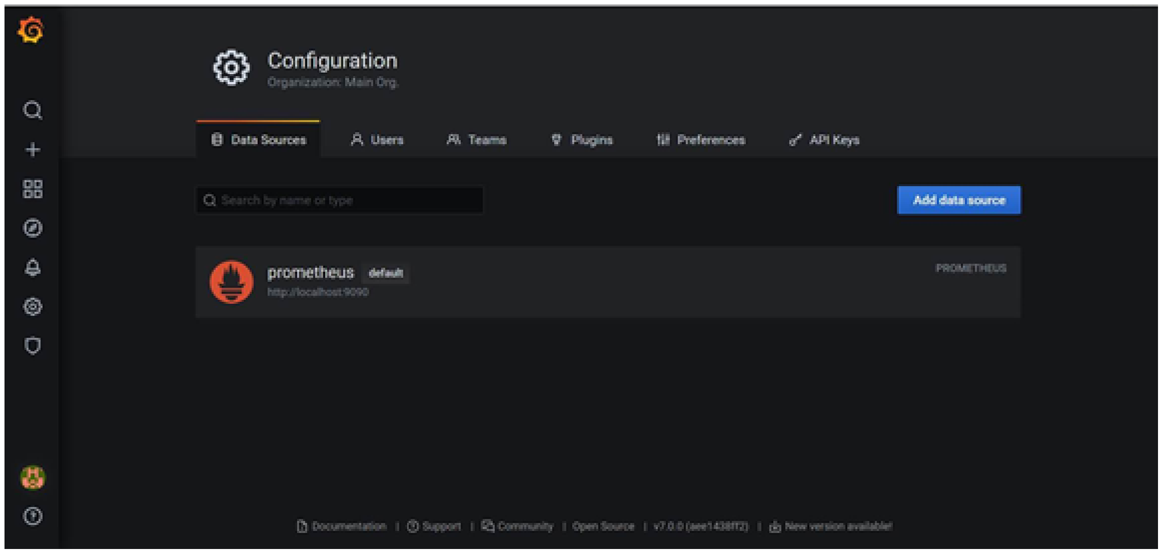
1. Click Configuration.


Home
Welcome to Grafana
Need help? Documentation Tutorials Community Public Slack
Remove this panel
Advanced
Manage your users and teams and add steps
Configuration
Data Sources
Users
Teams
Plugins
Preferences
API Keys
TUTORIAL
USERS
Create users and teams
Learn to organize your users in teams and manage resource access and roles.
PLUGINS
Find and install plugins
Learn how in the docs
Dashboards
Starred dashboards
Latest from the blog
New in Grafana Tanka: Customize Helm charts without modifying them
Oct 08
localhost:3000/datasources

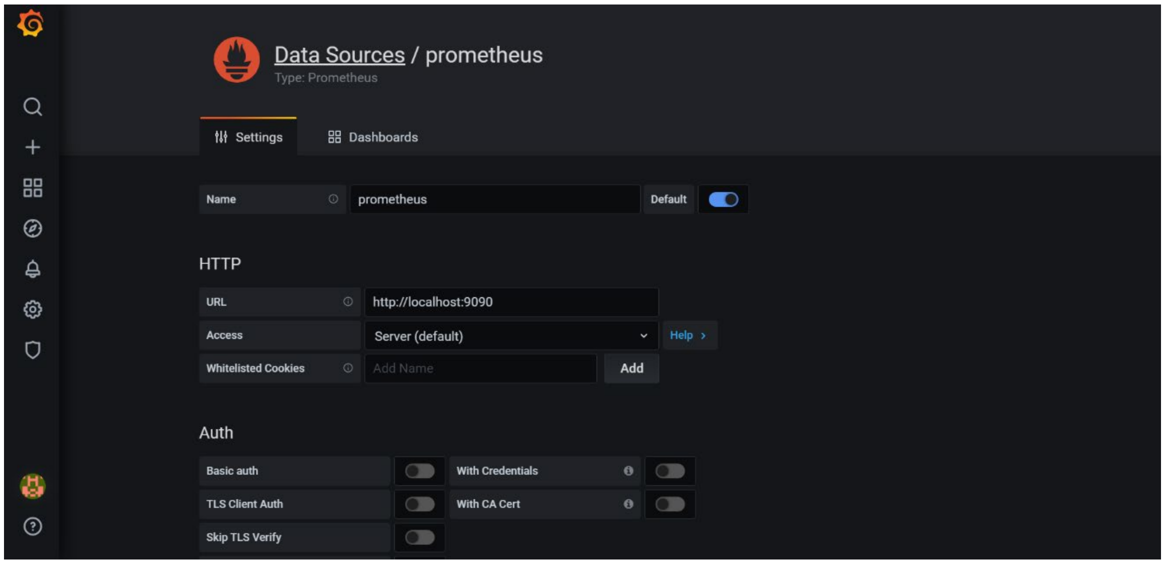
2. Select **Data Sources,** as shown in the image below:


Home
Welcome to Grafana
Need help? Documentation Tutorials Community Public Slack
Remove this panel
Advanced
Manage your users and teams and add
Configuration
Data Sources
Users
Teams
Plugins
Preferences
API Keys
TUTORIAL
USERS
Create users and teams
Learn to organize your users in teams and manage resource access and roles.
PLUGINS
Find and install plugins
Learn how in the docs
Dashboards
Starred dashboards
Latest from the blog
New in Grafana Tanka: Customize Helm charts without modifying them
Oct 08
localhost:3000/datasources

3. Click Add data source.



4. Select Prometheus.


Data Sources / prometheus
Type: Prometheus

Name: prometheus — Default: on

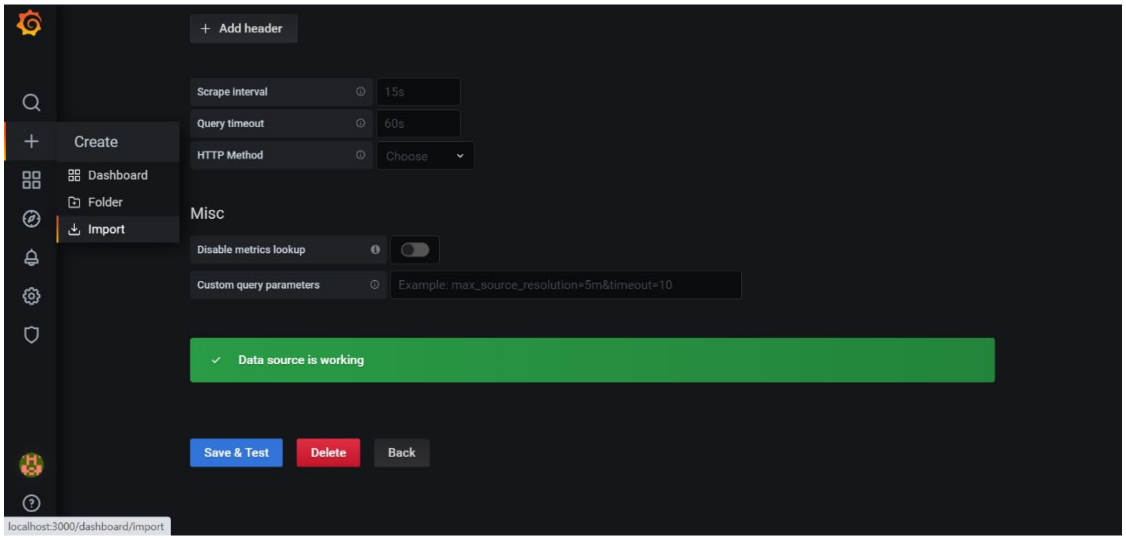
HTTP

URL: http://localhost:9090
Access: Server (default)
Whitelisted Cookies: Add Name

Auth

Basic auth: off — With Credentials: off
TLS Client Auth: off — With CA Cert: off
Skip TLS Verify: off

NOTE: Ensure the name of the data source is "prometheus". If Prometheus and Grafana are running on the same machine, use the default URL *http://localhost:9090*. Otherwise, ensure the URL matches the Prometheus URL, save, and test it.



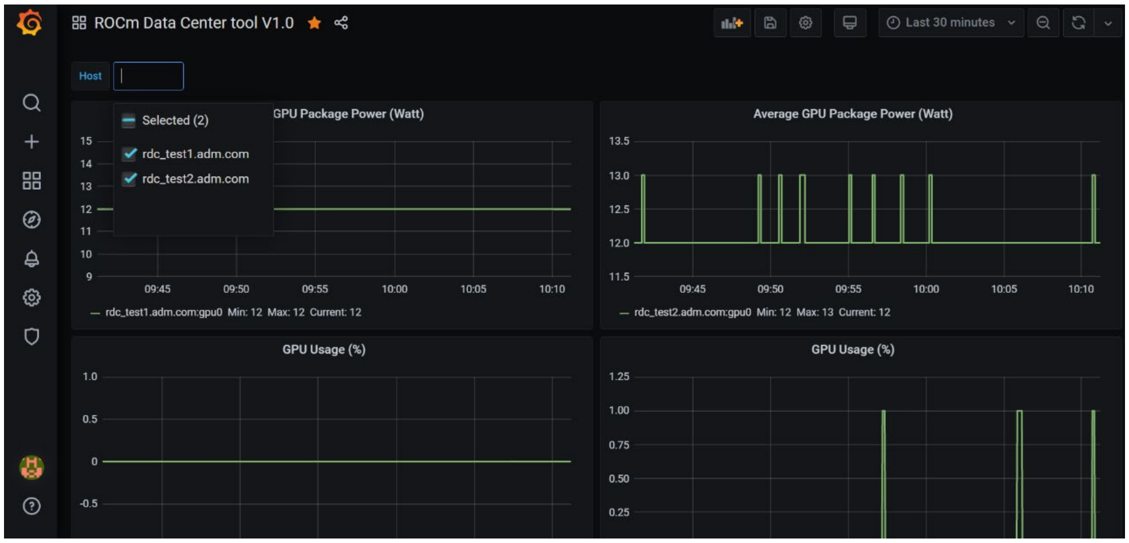
Scrape interval: 15s
Query timeout: 60s
HTTP Method: Choose

Misc

Disable metrics lookup: off
Custom query parameters: Example: max_source_resolution=5m&timeout=10

Data source is working

5. Click '+' and select **Import** to import the ROCm Data Center Tool dashboard.
6. Click the **Upload.json** file.
7. Choose rdc_grafana_dashboard_example.json, which is located in the python_binding folder.
8. Import the *rdc_grafana_dashboard_example.json* file and select the desired compute node on the dashboard, as shown in the image below:


ROCm Data Center tool V1.0
Last 30 minutes
Host
Selected (2)
rdc_test1.adm.com
rdc_test2.adm.com
GPU Package Power (Watt)
Average GPU Package Power (Watt)
rdc_test1.adm.com:gpu0 Min: 12 Max: 12 Current: 12
rdc_test2.adm.com:gpu0 Min: 12 Max: 13 Current: 12
GPU Usage (%)
GPU Usage (%)

4.3.3 Prometheus (Grafana) Integration with Automatic Node Detection

The ROCm Data Center (RDC) tool now enables you to use Consul to discover the *rdc_prometheus* service automatically. Consul is "a service mesh solution providing a full featured control plane with service discovery, configuration, and segmentation functionality". For more information, refer to their website at *https://www.consul.io/docs/intro*.

The ROCm Data Center Tool uses Consul for health checks of RDC's integration with the Prometheus plug-in (*rdc_prometheus*), and these checks provide information on its efficiency.

Previously, when a new compute node was added, users had to manually change *prometheus_targets.json* to use Consul. Now, with the Consul agent integration, a new compute node can be discovered automatically.

4.3.3.1 Installing the Consul Agent for Compute and Management Nodes

Follow the instructions below to install the latest Consul agent for compute and management nodes.

1. Set up the *apt* repository to download and install the Consul agent.

```
   curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
   sudo apt-add-repository "deb [arch=amd64]
https://apt.releases.hashicorp.com $(lsb_release -cs) main"
   sudo apt-get update && sudo apt-get install consul
```

2. Generate a key to encrypt the communication between consul agents. Note, you can generate the key once, and both the compute and management nodes use the same key for communication.

```
   $ consul keygen
```

For the purpose of this feature documentation, the following key is used in the configuration file.

```
   $ consul keygen
   4lgGQXr3/R2QeTi5vEp7q5Xs1KoYBhCsk9+VgJZZHAo=
```

4.3.3.2 Setting up the Consul Server in Management Nodes

While Consul can function with one server, it is recommended to use 3 to 5 servers to avoid failure scenarios, which often leads to data loss.

Note: For example purposes, the configuration settings documented below are for a single server.

1. Create a configuration file */etc/consul.d/server.hcl*:

```
server = true
encrypt = "<CONSUL_ENCRYPTION_KEY>"
bootstrap_expect = 1
ui = true
client_addr = "0.0.0.0"
bind_addr = "<The IP address can be reached by client>"
```

2. Run the agent in server mode and set the encrypt to the key generated in the first step. The *bootstrap_expect* variable indicates the number of servers required to form the first Consul cluster.

3. Set the number of servers to 1 to allow a cluster with a single server.

   - The User Interface (UI) variable is used to enable the Consul Web UI.
   - The client_addr variable is used to connect the API and UI.
   - The bind_addr variable is used to connect the client to the server. If you have multiple private IP addresses, use the address that can connect to a client.

4. Start the agent using the following instruction.

```
sudo consul agent -config-dir=/etc/consul.d/
```

5. Browse to http://localhost:8500/ on the management node. You will see a single instance running.

4.3.3.3 Setting up the Consul Client in Compute Nodes

1. Create a configuration file */etc/consul.d/client.hcl*

```
server = false
encrypt = "<CONSUL_ENCRYPTION_KEY>"
retry_join = ["<The consul server address>"]
client_addr = "0.0.0.0"
bind_addr = "<The IP address can reach server>"
```

Note: Use the same CONSUL_ENCRYPTION_KEY as the servers. In the retry_join, use the IP address of the management nodes.

2. Start the Consul agent.

```
sudo consul agent -config-dir=/etc/consul.d/
```

The client has now joined the Consul.

```
   $ consul members
   Node              Address            Status  Type    Build  Protocol  DC
Segment
   management-node   10.4.22.70:8301    alive   server  1.9.3  2     dc1  <all>
   compute-node      10.4.22.112:8301   alive   client  1.9.3  2     dc1
<default>
```

3. Setup the Consul client to monitor the health of the RDC Prometheus plug-in.

4. Start the RDC Prometheus plugin.

```
python rdc_prometheus.py --rdc_embedded
```

5. Add the configuration file */etc/consul.d/rdc_prometheus.hcl:*

```
{
  "service": {
    "name": "rdc_prometheus",
    "tags": [
      "rdc_prometheus",
      "rdc"
    ],
    "port": 5000,
    "check": {
      "id": "rdc_plugin",
      "name": "RDC Prometheus plugin on port 5000",
      "http": "http://localhost:5000",
      "method": "GET",
      "interval": "15s",
      "timeout": "1s"
    }
  }
}
```

Note: By default, the Prometheus plug-in uses port 5000. If you do not use the default setting, ensure you change the configuration file accordingly.

After the configuration file is changed, restart the Consul client agent.

```
   sudo consul agent -config-dir=/etc/consul.d/
```

6. Enable the Prometheus integration in the Management node. For more information, refer to the Prometheus Integration chapter in this guide.

7. In the Management node, inspect the service:

```
   $ consul catalog nodes -service=rdc_prometheus

   Node                  ID          Address        DC
   compute-node          76694ab1    10.4.22.112    dc1
```

8. Create a new Prometheus configuration *rdc_prometheus_consul.yml* file for the Consul integration:

```
   global:
     scrape_interval:     15s # Set the scrape interval to every 15 seconds.
Default is every 1 minute.
     evaluation_interval: 15s # Evaluate rules every 15 seconds. The default
is every 1 minute.
   scrape_configs:
     - job_name: 'consul'
       consul_sd_configs:
         - server: 'localhost:8500'
       relabel_configs:
         - source_labels: [__meta_consul_tags]
           regex: .*,rdc,.*
           action: keep
         - source_labels: [__meta_consul_service]
           target_label: job
```

Note: If you are not running the consul server and Prometheus in the same machine, change the server under consul_sd_configs to your consul server address.

9. Start Prometheus.

```
   $ ./prometheus --config.file="rdc_prometheus_consul.yml"
```

10. Browse to the Prometheus UI at http://localhost:9090 on the Management node and query RDC Prometheus metrics. Ensure the plug-in started before running the query.

4.4 Reliability, Availability, and Serviceability Plugin

4.4.1 RAS Plugin Installation

In this release, the ROCm Data Center Tool (RDC) extends support to the Reliability, Availability, and Serviceability (RAS) integration. When the RAS feature is enabled in the graphic card, users can use RDC to monitor RAS errors.

4.4.1.1 Prerequisite

You must ensure the graphic card supports RAS.

NOTE: The RAS library is installed as part of the RDC installation and no additional configuration is required for RDC.

The ROCm Data Center Tool installation dynamically loads the RAS library *librdc_ras.so*. The configuration files required by the RAS library are installed in the *sp3* and *config* folders.

```
% ls /opt/rocm-4.2.0/rdc/lib
... librdc_ras.so ...
... sp3 ... config ...
```

4.4.2 RAS Integration

RAS exposes a list of Error-Correcting Code (ECC) correctable and uncorrectable errors for different IP blocks and enables users to successfully troubleshoot issues.

For example, the *dmon* command passes the *ECC_CORRECT* and *ECC_UNCORRECT* counters field id to the command:

```
rdci dmon -i 0 -e 600,601
```

The *dmon* command monitors GPU index 0, field 600, and 601, where 600 is for the *ECC_CORRECT* counter and 601 is for the *ECC_UNCORRECT* counter.

```
% rdci dmon -l
... ...
600 RDC_FI_ECC_CORRECT_TOTAL : Accumulated Single Error Correction.
601 RDC_FI_ECC_UNCORRECT_TOTAL : Accumulated Double Error Detection.
602 RDC_FI_ECC_SDMA_SEC : SDMA Single Error Correction.
603 RDC_FI_ECC_SDMA_DED : SDMA Double Error Detection.
604 RDC_FI_ECC_GFX_SEC : GFX Single Error Correction.
605 RDC_FI_ECC_GFX_DED : GFX Double Error Detection.
606 RDC_FI_ECC_MMHUB_SEC : MMHUB Single Error Correction.
607 RDC_FI_ECC_MMHUB_DED : MMHUB Double Error Detection.
608 RDC_FI_ECC_ATHUB_SEC : ATHUB Single Error Correction.
609 RDC_FI_ECC_ATHUB_DED : ATHUB Double Error Detection.
610 RDC_FI_ECC_BIF_SEC : BIF Single Error Correction.
611 RDC_FI_ECC_BIF_DED : BIF Double Error Detection.
612 RDC_FI_ECC_HDP_SEC : HDP Single Error Correction.
613 RDC_FI_ECC_HDP_DED : HDP Double Error Detection.
614 RDC_FI_ECC_XGMI_WAFL_SEC : XGMI WAFL Single Error Correction.
615 RDC_FI_ECC_XGMI_WAFL_DED : XGMI WAFL Double Error Detection.
616 RDC_FI_ECC_DF_SEC : DF Single Error Correction.
617 RDC_FI_ECC_DF_DED : DF Double Error Detection.
618 RDC_FI_ECC_SMN_SEC : SMN Single Error Correction.
619 RDC_FI_ECC_SMN_DED : SMN Double Error Detection.
620 RDC_FI_ECC_SEM_SEC : SEM Single Error Correction.
621 RDC_FI_ECC_SEM_DED : SEM Double Error Detection.
622 RDC_FI_ECC_MP0_SEC : MP0 Single Error Correction.
623 RDC_FI_ECC_MP0_DED : MP0 Double Error Detection.
624 RDC_FI_ECC_MP1_SEC : MP1 Single Error Correction.

625 RDC_FI_ECC_MP1_DED : MP1 Double Error Detection.
626 RDC_FI_ECC_FUSE_SEC : FUSE Single Error Correction.
627 RDC_FI_ECC_FUSE_DED : FUSE Double Error Detection.
628 RDC_FI_ECC_UMC_SEC : UMC Single Error Correction.
629 RDC_FI_ECC_UMC_DED : UMC Double Error Detection.
... ...
```

Use the following command to access the ECC correctable and uncorrectable error counters.

```
% rdci dmon -i 0 -e 600,601
GPU     ECC_CORRECT         ECC_UNCORRECT
0       0                   0
0       0                   0
0       0                   0
```

Chapter 5 Developer Handbook

The ROCm™ Data Center Tool™ (RDC) is open source and available under the MIT License. This section is targeted at open source developers. Third-party integrators may also use the section.

5.1.1 Prerequisites for building RDC

NOTE: The ROCm Data Center Tool is tested on the following software versions. Earlier versions may not work.

- CMake 3.15
- g++ (5.4.0)
- AMD ROCm which includes AMD ROCm SMI Library
- gRPC and protoc (Refer 2.2.1)

The following components are required to build the latest documentation:

- Doxygen (1.8.11)
- Latex (pdfTeX 3.14159265-2.6-1.40.16)

```
$ sudo apt install libcap-dev
$ sudo apt install -y doxygen
```

5.1.2 Building and Installing RDC

Clone the RDC source code from GitHub and use CMake to build and install.

```
$ git clone <GitHub for RDC>
$ cd rdc
$ mkdir -p build; cd build
$ cmake -DROCM_DIR=/opt/rocm[2] -DGRPC_ROOT="$GRPC_PROTOC_ROOT"..
$ make
#Install library file and header and the default location is /opt/rocm
$ make install
```

5.1.3 Building documentation

You can generate PDF documentation after a successful build. The reference manual, `refman.pdf`, appears in the `latex` directory.

[2] location of ROCm install root. By default, it is /opt/rocm

```
$ make doc
$ cd latex
$ make
```

5.1.4 Build unit tests for ROCm Data Center Tool™

```
$ cd rdc/tests/rdc_tests
$ mkdir -p build; cd build
$ cmake -DROCM_DIR=/opt/rocm -DGRPC_ROOT="$GRPC_PROTOC_ROOT"..
$ make

# To run the tests

$ cd build/rdctst_tests
$ ./rdctst
```

5.1.5 Test

```
# Run rdcd daemon
$ LD_LIBRARY_PATH=$PWD/rdc_libs/  ./server/rdcd -u

# In another console run the RDC command-line
$ LD_LIBRARY_PATH=$PWD/rdc_libs/  ./rdci/rdci discovery -l -u
```

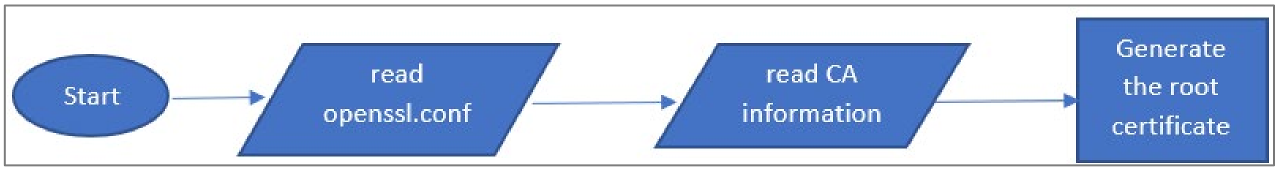
5.2 Authentication

5.2.1 Generating Files for Authentication

The ROCm Data Center Tool™ supports encrypted communications between clients and servers. The communication can be configured to be authenticated or not authenticated. By default, authentication is enabled.
To disable authentication, when starting the server, use the "--unauth_comm" flag (or "-u" for short). You must also use *"-u"* in *rdci* to access *unauth rdcd*. The */lib/systemd/system/rdc.service* file can be edited to pass arguments to rdcd on starting. On the client side, when calling rdc_channel_create(), the "secure" argument must be set to False.

5.2.1.1 Scripts

RDC users manage their own keys and certificates. However, some scripts generate self-signed certificates in the RDC source tree in the authentication directory for test purposes. The following flowchart depicts how to generate the root certificates using the *openssl* command in 01gen_root_cert.sh:



The section where the default responses to *openssl* questions can be specified is included in *openssl.conf*. To locate the section, look for the following comment line:

```
# < ** REPLACE VALUES IN THIS SECTION WITH APPROPRIATE VALUES FOR YOUR ORG.
**>
```

It is helpful to modify this section with values appropriate for your organization if you expect to call this script many times. Additionally, you must replace the dummy values and update the alt_names section for your environment.

To generate the keys and certificates using these scripts, make the following calls:

```
$ 01gen_root_cert.sh
# provide answers to posed questions
$ 02gen_ssl_artifacts.sh
# provide answers to posed questions
```

At this point, the keys and certificates are in the newly-created "CA/artifacts" directory. This directory must be deleted if you need to rerun the scripts.

To install the keys and certificates, access the artifacts directory, and run the install.sh script as root, specifying the install location. By default, RDC expects this to be in /etc/rdc:

```
$ cd CA/artifacts
$ sudo install_<client|server>.sh /etc/rdc
```

These files must be copied to and installed on all client and server machines that are expected to communicate with one another.

5.2.1.2 Known Limitation

The ROCm Data Center Tool™ has the following authentication limitation.

The client and server are hardcoded to look for the *openssl* certificate and key files in /etc/rdc. There is no workaround available currently.

5.2.2 Verifying Files for Authentication

Several SSL keys and certificates must be generated and installed on clients and servers for authentication to work properly. By default, the RDC server will look in the */etc/rdc* folder for the following keys and certificates:

5.2.2.1 Client

```
  $ sudo tree /etc/rdc

        /etc/rdc

        |-- client

           |-- certs

           |    |-- rdc_cacert.pem

           |    |-- rdc_client_cert.pem

           |-- private

|-- rdc_client_cert.key
```

NOTE: Machines that are clients and servers will consist of both directory structures.

5.2.2.2 Server

```
$ sudo tree /etc/rdc

     /etc/rdc

     |-- server

        |-- certs

        |   |-- rdc_cacert.pem

        |   |-- rdc_server_cert.pem

        |-- private

      |-- rdc_server_cert.key
```

Chapter 6 ROCm™ Data Center API

Disclaimer: This is the alpha version of ROCm Data Center (RDC) API™ and is subject to change without notice. The primary purpose of this API is to solicit feedback. AMD accepts no responsibility for any software breakage caused by API changes.

6.1 RDC API

The ROCm™ Data Center Tool™ API is the core library that provides all the RDC features. This section focuses on how RDC API can be used by third-party software.

The RDC includes the following libraries:

- librdc_bootstrap.so: Loads during runtime one of the two libraries by detecting the mode
- librdc_client.so: Exposes RDC functionality using gRPC client
- librdc.so: RDC API. This depends on librocm_smi.so
- librocm_smi.so: Stateless low overhead access to GPU data

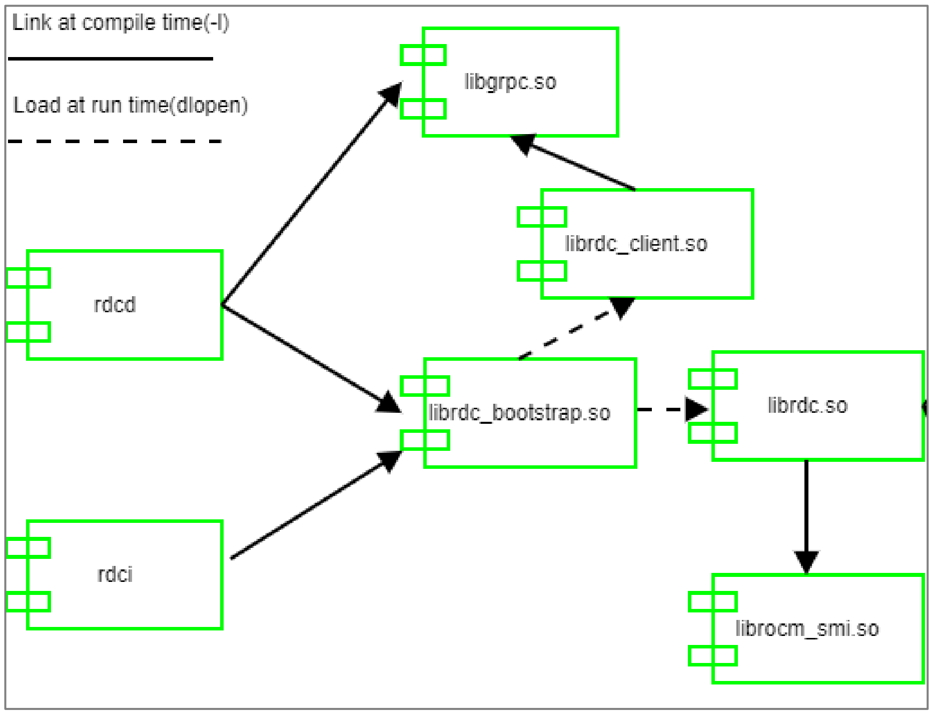


Figure 3 Different libraries and how they are linked

Note that librdc_bootstrap.so loads different libraries based on the modes. For example,

- rdci: librdc_bootstrap.so loads librdc_client.so

- rdcd: librdc_bootstrap.so loads librdc.so

For more information, see the *ROCm™ Data Center Tool API Guide* at,

https://github.com/RadeonOpenCompute/ROCm/blob/master/AMD_RDC_API_Guide_v4.3.pdf

6.2 Job stats use case

The following pseudocode shows how ROCm Data Center (RDC) API™ can be directly used to record GPU statistics associated with any job or workload. Refer to the example code provided with RDC on how to build it.

For more information, see *Job Stats*

```
//Initialize the RDC
rdc_handle_t rdc_handle;
rdc_status_t result=rdc_init(0);

//Dynamically choose to run in standalone or embedded mode
bool standalone = false;
std::cin>> standalone;
if (standalone)
    result = rdc_connect("127.0.0.1:50051", &rdc_handle, nullptr, nullptr,
nullptr); //It will connect to the daemon
else
    result = rdc_start_embedded(RDC_OPERATION_MODE_MANUAL, &rdc_handle);
//call library directly, here we run embedded in manual mode

//Now we can use the same API for both standalone and embedded
//(1) create group
rdc_gpu_group_t groupId;
result = rdc_group_gpu_create(rdc_handle, RDC_GROUP_EMPTY, "MyGroup1",
&groupId);

//(2) Add the GPUs to the group
result = rdc_group_gpu_add(rdc_handle, groupId, 0); //Add GPU 0
result = rdc_group_gpu_add(rdc_handle, groupId, 1); //Add GPU 1

//(3) start the recording the Slurm job 123. Set the sample frequency to once
per second
result = rdc_job_start_stats(rdc_handle, group_id,
        "123", 1000000);

//For standalone mode, the daemon will update and cache the samples
//In manual mode, we must call rdc_field_update_all periodically to take
samples
if (!standalone) { //embedded manual mode
    for (int i=5; i>0; i--) { //As an example, we will take 5 samples
        result = rdc_field_update_all(rdc_handle, 0);
        usleep(1000000);
    }
} else { //standalone mode, do nothing
    usleep(5000000); //sleep 5 seconds before fetch the stats
}

//(4) stop the Slurm job 123, which will stop the watch
// Note: we do not have to stop the job to get stats. The rdc_job_get_stats
can be called at any time before stop
result = rdc_job_stop_stats(rdc_handle, "123");

//(5) Get the stats
rdc_job_info_t job_info;
result = rdc_job_get_stats(rdc_handle, "123", &job_info);
std::cout<<"Average Memory Utilization: "
<<job_info.summary.memoryUtilization.average <<std::endl;

//The cleanup and shutdown ....
```