Collectd how to implement a simple plugin

Clone the collectd repo: https://github.com/collectd/collectd.git

An Overview of the architecture of a collectd plugin exists here: https://collectd.org/wiki/index.php/Plugin_architecture

To start off creating a plugin, create a c file under the src directory in collectd, the c file should have:

1. A config function
2. An init function
3. A read function
4. A shutdown function
5. A function to register callbacks for all the functions above.

```
#include "collectd.h"
#include "common.h"

#define PLUGIN_NAME "my_plugin"

static _Bool g_enable_option1;
static char g_option2 [DATA_MAX_NAME_LEN];

static int my_config_function(oconfig_item_t *ci) {
  int ret = 0;

  INFO (PLUGIN_NAME ": %s:%d", __FUNCTION__, __LINE__);

  for (int i = 0; i < ci->children_num; i++) {
    oconfig_item_t *child = ci->children + i;

    if (strcasecmp("Option1", child->key) == 0) {
      ret = cf_util_get_boolean(child, & g_enable_option1);
    } else if (strcasecmp("Option2", child->key) == 0) {
      ret = cf_util_get_string_buffer(child, g_option2,
                                sizeof(g_option2));

    }else {
      ERROR(PLUGIN_NAME ": Unknown configuration parameter \"%s\".", child->key);
      ret = (-1);
    }

    if (ret != 0) {
      INFO (PLUGIN_NAME ": %s:%d ret=%d", __FUNCTION__, __LINE__, ret);
      return ret;
    }
  }

  return (0);
}


static int my_read_function(__attribute__((unused)) user_data_t *ud) {

  INFO (PLUGIN_NAME ": %s:%d", __FUNCTION__, __LINE__);
  /*  In order to dispatch values to the collectd daemon you need to fill out a value list and use
plugin_dispatch_values() An example is provided below
   *
   *  value_list_t vl = VALUE_LIST_INIT;
   *
   *  vl.values = &(value_t){.derive = value}; //NOTE please change value to the actual value and use an
appropriate type: checkout https://wiki.opnfv.org/display/fastpath/Collectd+101 for more info
   *
   *  vl.values_len = 1;
   *  sstrncpy(vl.host, hostname_g, sizeof(vl.host));
   *  sstrncpy(vl.plugin, PLUGIN_NAME, sizeof(vl.plugin));
   *  sstrncpy(vl.plugin_instance, "my_plugin_instance", sizeof(vl.plugin_instance)); // replace
```

```
my_plugin_instance with an appropriate string/instance... see the end of this wiki
   *  sstrncpy(vl.type, "my_plugin_type", sizeof(vl.type));// replace my_plugin_type with an appropriate
string... see the end of this wiki
   *
   *  plugin_dispatch_values(&vl);
   *
   *
   * To dispatch a notification you need to fill out a notification structure and use
plugin_dispatch_notification()
   *  notification_t n = {
   *      .severity = severity, // where severity is one of NOTIF_OKAY, NOTIF_WARNING, NOTIF_FAILURE
   *      .time = cdtime(),
   *      .message = "Some message",
   *      .plugin = PLUGIN_NAME};
   *
   *  sstrncpy(n.host, hostname_g, sizeof(n.host));
   *  sstrncpy(n.plugin_instance, "my_plugin_instance", sizeof(n.plugin_instance)); // replace
my_plugin_instance with an appropriate string... see the end of this wiki
   *
   *  plugin_dispatch_notification(&n);
   */


  return (0);
}


static int my_shutdown_function (void) {

  INFO (PLUGIN_NAME ": %s:%d", __FUNCTION__, __LINE__);

  return (0);
}

static int my_init_function(void) {
  INFO(PLUGIN_NAME ": %s:%d", __FUNCTION__, __LINE__);

  return (0);
}


void module_register(void) {
 plugin_register_init("my_plugin", my_init_function);
  plugin_register_complex_config("my_plugin", my_config_function);
  plugin_register_complex_read(NULL, "my_plugin", my_read_function, 0, NULL);
  plugin_register_shutdown("my_plugin", my_shutdown_function);
}
```

Under src/ Modify the types.db file to include a new type

```
my_plugin_type           value:DERIVE:0:U
```

Under the top level collectd directory Modify the Makefile.am file to build your plugin

```
if BUILD_PLUGIN_MY_PLUGIN
pkglib_LTLIBRARIES += my_plugin.la
my_plugin_la_SOURCES = src/my_plugin.c
my_plugin_la_CFLAGS = $(AM_CFLAGS)
my_plugin_la_LDFLAGS = $(PLUGIN_LDFLAGS)
endif
```

Under the top level collectd directory Modify the configure.ac file to include your plugin (search for plugin_ascent="no" and add it in the correct place alphpbetically), and set it to disabled (so it doesn't build on all platforms):

```
plugin_my_plugin="no"
```

configure (in configure.ac) your plugin so that it's enabled on linux under:

```
# Linux
if test "x$ac_system" = "xLinux"; then

plugin_my_plugin="yes"
```

Configure (in configure.ac) your plugin to report if it's enabled or not:

```
AC_PLUGIN([my_plugin],           [$plugin_my_plugin],     [plugin desc])
```

In configure.ac under: "AC_MSG_RESULT([Modules:])", enable reporting on the plugin status from the configuration step:

```
AC_MSG_RESULT([    my_plugin. . . . . $enable_my_plugin])
```

Under src/ Finally it's time to configure the configuration stanzas for your new plugin. By modifygin collectd.conf.in to include:

```
#@BUILD_PLUGIN_MY_PLUGIN_TRUE@LoadPlugin my_plugin


#<Plugin my_plugin>
#    Option1 true
#    Option2 "example"
#</Plugin>
```

Under src/ Update the plugin configuration description in collectd.conf.pod

```
=head2 Plugin C<my_plugin>


The I< my_plugin > plugin collects TBD.


B<Synopsis:>


  <Plugin my_plugin>
    Option1 true
    Option2 “example”
  </Plugin>


B<Options:>


=over 2


=item B<Option1> B<false>|B<true>


TBD.


=item B<Option2> I<some_string>


TBD.


=back
```

you should now be able to build your simple collectd plugin, in the top level directory:

```
$ ./build.sh
$ ./configure --prefix=$HOME/clct
$ make
$ make -j install
```

This will install collectd to $HOME/clct.

To run your new plugin:

```
$ cd $HOME/clct
```

Edit etc/collectd.conf to include:

```
LoadPlugin my_plugin
LoadPlugin logfile

<Plugin logfile>
        LogLevel debug
        File STDOUT
        Timestamp true
        PrintSeverity false
</Plugin>

<Plugin my_plugin>
    Option1 true
    Option2 “example”
</Plugin>
```

Run collectd:

```
sbin/collectd -f -C etc/collectd.conf
```

Extra Challenge:

1. Try to add a double plugin configuration option and read it in with: cf_util_get_double
2. Write the configuration parameters to the collectd logfile

Statistics in collectd

Statistics in collectd consist of a value list. A value list includes:

Value list		**Example**	**comment**
Values		11111 2112	
Value length	the number of values in the data set.		
Time	timestamp at which the value was collected.	14758 37857	epoch
Interval	interval at which to expect a new value.	10	interval
Host	used to identify the host.	localh ost	can be uuid for vm or host… or can give host a name
Plugin	used to identify the plugin.	netwo rk	
Plugin instance (optional)	used to group a set of values together. For e.g. values belonging to a DPDK interface.	eth0	
Type	**unit** used to measure a value. In other words used to refer to a data set.	packe ts	
Type instance (optional)	**used to distinguish between values that have an identical type.**	rx or tx	
meta data	an opaque data structure that enables the passing of additional information about a value list. “Meta data in the global cache can be used to store arbitrary information about an identifier”		

Notifications in collectd

Notifications in collectd are generic messages containing:

An associated severity, which can be one of OKAY, WARNING, and FAILURE.
A time.
A Message
A host.
A plugin.
A plugin instance (optional).
A type.
A types instance (optional).
Meta-data.