UCS User Guide

Table of Contents

Build and Install Guide

Environment

The following information is based on OpenSUSE-leap 15 (The **server** version was select not the **not the desktop** version for testing). Then all system updates were applied.

Build Dependencies

UCS Build Dependencies

- Java 8+ JDK
- RPM Build Tools
- python 3.4+ (In distro as ‘python3’)

VoltDB Build Dependencies

- Java 8+ JDK
- RPM Build Tools
- ant
- gcc/g++
- libpcre2
- cmake
- valgrind
- python 2.6+ (In distro and Python 2.x is the default for the ‘python’)

Install All Build Dependencies

The following should install everything for build:

- `sudo zypper install ant gcc gcc-c++ pcre2-devel cmake valgrind java-1_8_0-openjdk java-1_8_0-openjdk-devel rpm-devel git python-setuptools`

Build UCS

These instructions will only work if the build system used has Internet access.

Build VoltDB at the Expected Location (/opt/voltdb)

- *Note:* Do not use the paid version of VoltDB with this code drop!
- Download version 6.1 as root in /opt:
  - `sudo -i`
  - `cd /opt`
  - `git clone -b voltdb-6.1 https://github.com/VoltDB/voltdb.git`
- `Build voltDB:`
  - `cd voltdb`
  - `ant -Djmemcheck=NO_MEMCHECK`
  - ***If this fails for Warnings as Errors:*** *Edit build.py line 43 removing compiler option -Werror from the line. Then re-run the ant step above. This happens with newer versions of gcc. The version used on OpenSUSE Leap 15 was 7.3.1.*

- Add the following lines to your .bashrc files or system level environmental files as appropriate:
  - `export PATH="$PATH:/opt/voltdb/bin"`
  - `export CLASSPATH="$CLASSPATH:/opt/voltdb/voltdb/*"`
- VoltDB server requires specific kernel memory values be set ideally this will be set in the kernel on boot however if not set and before you can start VoltDB be sure to set after each reboot:
  - `sudo echo never >/sys/kernel/mm/transparent_hugepage/enabled`
  - `sudo echo never >/sys/kernel/mm/transparent_hugepage/defrag`
- Make a tarball of the built bits in /opt/voltdb for installing on the management system that will run the VoltDB Server (also this could alternately be on a shared network folder).
  - `cd /`
  - `sudo tar -czf voltdb-6.1.tar.gz /opt/voltdb`
- To install the voltdb tarball on a management system do as root:
  - `cd /`
  - `sudo tar -xzf <tarball_location>/voltdb-6.1.tar.gz`

Build UCS RPMs

The UCS build process uses a technology called gradle to build the Java and Python code/packages. You **do not** need to install gradle to build UCS. This is why Internet access is required to build UCS.

- If you have access to the Internet ONLY through a proxy then you must set up gradle to use this proxy. The following template (properly filled in) must be located in a **~/.gradle** folder with a filename of **gradle.properties**. Create the folder and file if needed.
  - `cd ~`
  - `mkdir -p .gradle`
  - ```
    cat >.gradle/gradle.properties <<EOF
    systemProp.http.proxyHost=<your_proxy_url>
    systemProp.http.proxyPort=
    systemProp.https.proxyHost=<your_proxy_url>
    systemProp.https.proxyPort=<your_proxy_port>
    EOF
    ```
- Extract the delivered ucs-<**version**>-<**release**>-**source.tar.gz tarball**. This will create a **ucs** folder.
  - `cd ~`
  - `tar -xvf ucs-<version_and_release>-source.tar.gz`
  - `cd ucs`
  - `./gradlew build`
- The RPMs for UCS install will be located in **usc/build/distributions/**. They will install to **/opt/ucs** on management systems.

UCS Runtime Dependencies

- Java 8+ JRE
- VoltDB 6.1 OSS (Installed at /opt/voltdb)
- RabbitMQ Server
- Optional: logstash
    ```

UCS Installation

Install the Platform Supported Java 8+ JRE for All Management Systems

- `sudo zypper install java-1_8_0-openjdk`

Install the RabbitMQ Server Where it Will Run

- `sudo zypper install rabbitmq-server`
- Now edit the **rabbitmq.conf** file at **/etc/rabbitmq/** uncommenting the configuration line:

```
%% Uncomment the following line if you want to allow access to the
%% guest user from anywhere on the network.
{loopback_users, []}
```

- Be sure to remove the comma at the end or the rabbitmq server will fail to restart!
- Then run the following lines:
  - `sudo systemctl enable rabbitmq-server`
  - `sudo systemctl restart rabbitmq-server`
- Confirm that rabbitmq-server started with:
  - `systemctl status rabbitmq-server`

Install Logstash Where the logstash Will be Run

If using files as a data source to publish to simulated Cray messsage bus, install **logstash** on the source system hosting the simulated message bus. See the Appendix for an example of turning a cray-like file into a rabbitmq published message format. Alternately, use **eventsim.sh** to feed data to a rabbitmq subjects for telemetry and events.

- Configure the logstash repository as directed here:
  - https://www.elastic.co/guide/en/logstash/current/installing-logstash.html
- Then install logstash with:
  - `sudo zypper install logstash`
- Configure logstash to publish the expected JSON from the cray-like data files.

Install UCS

Copy the 2 RPMs from the build system's **ucs/build/distributions** folder onto each management system and install the RPMs. The installation includes a volt (**dai-volt.service**) and adapter (**adapter.service**) service files to make running the UCS simple.

See the User's Manual for configuring and using UCS.

Configuring UCS for a specific System

UCS requires three types of configuration in order to understand the type of system it is running on:

1. System Manifest
    1. Contained in the file /opt/ucs/etc/SystemManifest.json
    2. Describes the hierarchy of the system build, with racks, down to nodes.
2. Machine Configuration
    1. Contained in the file /opt/ucs/etc/MachineConfig.json
    2. Describes the parameters of the various nodes installed in the system, like MAC addresses, static IP addresses, etc.
3. Adapter Launcher Configuration
    1. Contained in the file /opt/ucs/etc/AdapterLauncher2.json
    2. Describes the various types of providers that have to be launched against specific adapters of UCS.

Configuring HSS Providers

The UCS software package contains the following sub-components, referred to as ‘providers’

1. Monitoring
    1. HSSEventMonitorProvider
    2. HSSEnvironmentalMonitorProvider
2. Provisioning
    1. HSSProvisionerProvider
3. Inventory Tracking
    1. CrayInventoryProvider

Monitoring

The two providers under monitoring are responsible for parsing and translating the RAS & Environmental events originating from the Cray’s HSS stack. They both share the same configuration file located at - /opt/ucs/etc/AdapterMonitor.json.

This is a sample configuration file:

```
{
  "logger": "log4j2",
  "db_servers": "127.0.0.1",
  "cray_bus": {
    "uri": "amqp://127.0.0.1",
    "exchange_name": "cray_sim"
  },
  "cray_environmental_telemetry": {
    "queue_name": "telemetry_data",
    "subjects": ["env"],
    "id_name_map_resource": "/resources/CraySensorMetaData.json",
    "factors": {
      "mV": 0.001,
      "mA": 0.001,
      "centiA": 0.01
    }
  },
  "cray_event_log": {
    "queue_name": "cray_event",
    "subjects": ["evt"],
    "event_map_resource": "/resources/CrayEventMetaData.json"
  },
  "cray_layout": {
    "rows": 10,
    "columns": 20
  }
}
```

The configuration file is broadly divided in to three parts – Common configuration details for both Event & Environmental monitoring providers, and two other parts exclusively for each of Event & Environmental monitoring provider.

Configuration Key		**Description**
logger		Select the type of logger to be used for displaying debug, error, etc messages. Currently ‘log4j’ & ‘console’ are supported.
db_servers		IP Address of the server running the Nearline Tier DB.
cray_bus		Connection details for registering with the Cray HSS system to receive its data.
	uri	URI pointing to the location where the Cray HSS’ AMQP server is located.
	exchange_name	The exchange name under which Cray HSS is publishing the RAS & Environmental data.
cray_environmental_telemetry		Configuration details for the HSS Environmental Monitoring Provider.
	queue_name	Queue name of the AMQP exchange publishing Environment data.
	subjects	The list of subjects under which the environmental data is published by the Cray HSS.
	id_name_map_resource	The look up table which contains the metadata of the possible Cray Environmental Sensors.
	factors	Conversion factors for various types of measurement units.
cray_event_log		Configuration details for the HSS Event Monitoring Provider.
	queue_name	Queue name of the AMQP exchange publishing RAS event data.
	subjects	The list of subjects under which the RAS event data is published by the Cray HSS.
	event_map_resource	The look up table which contains the metadata of the possible Cray RAS events.
cray_layout		Describe system layout for effectively translating Cray location information into UCS locations.
	rows	Number of Rows.
	columns	Number of Columns.

Provisioning

Although this provider is not directly related to the provisioning of nodes, it is however responsible for updating the node state depending on the RAS events generated by the Cray HSS. Currently this provider only updates the Node state into three possible options – Active, Booting, Missing.

Here is a sample configuration:

```
{
    "logger": "log4j2",
    "db_servers": "127.0.0.1",
    "providers" : ["hss"],
    "hss" : {
        "exchangeName" : "cray_sim",
        "queueName" : "event_data",
        "uri"       : "amqp://127.0.0.1",
        "subjects" : ["evt"],
        "mapping_file_name" : "/resources/CrayEventAvailabilityMapping.json",
        "row_count" : "10",
        "column_count" : "10"
    }
}
```

Configuration Key		**Description**
logger		Select the type of logger to be used for displaying debug, error, etc messages. Currently ‘log4j’ & ‘console’ are supported.
db_servers		IP Address of the server running the Nearline Tier DB.
providers		Specifies the type of provisioning providers the app has to select. Takes in a list of provider names. Currently only ‘hss’ is supported.
“hss”		This section corresponds to the configuration details of the specific provisioning provider selected in the ‘providers’ configuration.
	exchangeName	The exchange name under which Cray HSS is publishing the RAS data.
	queueName	Queue name of the AMQP exchange publishing RAS data.
	uri	URI pointing to the location where the Cray HSS’ AMQP server is located.
	subjects	The list of subjects under which the RAS event data is published by the Cray HSS. Takes in a list of subjects.
	mapping_file_name	The look up table that contains the mapping from various Cray events to UCS states.
	row_count	The number of rows in the final system layout. Used for converting Cray location information into UCS format.
	column_count	The number of racks in one row in the final system layout. Used for converting Cray location information into UCS format.

Inventory Tracking

Features

UCS has the capability to monitor the system for inventory changes and record them in the data store to keep track of current (last known) state and location history. For this, it relies on the HSS system to provide inventory updates. The use cases the feature supports are:

- Component removal
- Component installation or replacement
  - Returning to service
  - Replacing a component with a different one
  - First-time installation

NOTE: for this initial implementation, only compute and service nodes are supported.

Component removal simply results in updating the state of the component to indicate it has been removed ("missing"). The serial number and other details need not be updated as only at a later instance it will be known if the component will simply be returned to service or replaced. So returning a component to service simply results in updating the state of the component to indicate it's now present and ready ("active").

Replacing a component with a different one does result in updating the component's information (at a minimum the serial number, plus any other details provided by the HSS system). A first-time installation simply means that there didn't used to be a component registered in the database for the provided location, so all the information is filled in at that time. Both these cases result in adding a replacement history record, which will include previous vs. new component state and serial number (for the location in question).

The inventory information maintained in the database per component will consist of a standard set of fields typically needed for normal operation of the system (e.g. state, serial number, host name, IP addresses) plus a flexible set of data for additional hardware details (e.g. sockets, cores, CPU family and model, part numbers, memory details, manufacturer, etc.). Using a flexible schema (e.g. JSON) for storing these additional details is convenient for providing support for different types of systems and hardware components. The structure and amount of details stored will depend on what the HSS system can provide.

Overview of Operation

The expectation is that the HSS system will publish inventory changes on a message bus for UCS to monitor and update its data store. This initial implementation was based on Cray XC Series ("Cascade") systems, which at the moment does not provide such feature. As a result, an inventory data collector has been prototyped that can monitor for inventory changes, gather details and publish them on the message bus.

The Cray administrative manual provides a set of procedures for performing service operations (which can alter inventory). When these operations result in inventory changes (removals or replacements), the *xtdiscover* command must be used to update the Cray system's inventory database. This command

provides a summary of changes it finds in a diff file. Further details about the hardware can be obtained using the *xthwinv* command.

So the inventory data collector basically monitors the inventory change summary file to detect when changes have occurred (as a result of normal administrative operations), then uses the *xthwinv* command to gather further details, and finally publishes the changes to the message bus.

On the other side of the message bus, UCS has a Cray provider listening and processing these messages to then forward them to the inventory component (adapter) that performs the inventory tracking and records updates in the UCS data store.

Dependencies

For normal operation of this feature and in order to capture inventory changes correctly, the following are the dependencies and requirements:

- Cray XC Series system
- Service operations must be performed as required and outlined in the Cray administrative manual in order for inventory changes to be recorded correctly
- The inventory data collector must be running on a service node with access to the *xtdiscover* output file and to execute the *xthwinv* command
- The inventory data collector must have access to a message bus to publish inventory changes
- UCS must have access to this message bus to receive these messages

Configuration

The inventory adapter requires the following parameters:

- “exchange_name”: the name of the message bus exchange to subscribe and listen for messages
- “exchange_uri”: the exchange URI

These parameters can be configured in: <UCS installation directory>/etc/AdapterInventory.json.

The inventory data collector requires the following parameters:

Configuration Key	**Description**
config_path	The directory path where the *xtdiscover* command output file will reside.
inventory_changes_file	The name of the *xtdiscover* change summary file.
inventory_file	The file name to use when generating the inventory information (using the *xthwinv* command)
exchange_name	The exchange name under which Cray HSS is publishing the RAS data.
exchange_uri	URI pointing to the location where the Cray HSS’ AMQP server is located.

These parameters can be configured in: <UCS installation directory>/etc/InventoryDataCollector.json

Testing in Simulated Environment

If a Cray XC series system is not available, the inventory tracking can be simulated simply by modifying the inventory change summary file to reflect the desired changes and supplying the hardware details in XML format. Please see the appendix for an example of these files.

Steps to test the inventory tracking feature:

- Edit <UCS installation directory>/etc/AdapterInventory.json. Specify the exchange name and URI for the message bus.
- Edit <UCS installation directory>/etc/InventoryDataCollector.json.
  - Specify the path where the inventory change summary file will be located
  - Specify the name of the inventory change summary file
  - Specify the full path for the inventory XML file
  - Specify the exchange name and URI for the message bus
- Start the inventory data collector: invdatacollector.
- Simulate inventory changes by updating the inventory files:
  - Update the inventory XML file with the hardware details
  - Update the summary file to indicate what changes happened (installations and removals)

Once changes are detected by the inventory data collector, they should be published and recorded in the data store. These changes can be viewed by querying the data store through the UCS CLI, GUI or through SQL.

Sample Inventory Changes – Summary File

By default, on a Cray system this file will reside in: /opt/cray/hss/default/etc/xtdiscover-config-changes.diff. The following example shows one removal (blade c0-0c0s4 with its respective nodes) and one installation (blade c0-0c0s5 with its respective nodes):

```
/opt/cray/hss/default/etc/xtdiscover-config-changes.diff - Fri Jun 22 17:37:54 2018


This file contains configuration differences as discovered by xtdiscover

Configuration change summary:
-----------------------------
      0 new components
      2 removed components
      0 now disabled components (formerly enabled)
      0 now disabled components (formerly empty)
      54 now empty components (formerly enabled)
      0 now enabled components (formerly empty)
      0 modified components

Summary by type:
----------------

      New Components:
      -----------------------------------------------

      Removed Components:
      -----------------------------------------------
      Deleted 2 kpdcs (0 empty or disabled)

      Now Disabled (Previously Enabled) Components:
```

```
      ------------------------------------------------

      Now Disabled (Previously Empty) Components:
      ------------------------------------------------

      Now Empty (Previously Enabled) Components:
      ------------------------------------------------
      Made empty 1 slot (formerly enabled)
      Made empty 4 nodes (formerly enabled)
      Made empty 1 aries asic (formerly enabled)
      Made empty 48 aries lcbs (formerly enabled)

      Now Enabled (Previously Empty) Components:
      ------------------------------------------------

      Modified Components:
      ------------------------------------------------


New Components (detailed):
------------------------------------------------------------


Removed Components (detailed):
------------------------------------------------------------

[0]:     c0-0c0s4l0  type=rt_kpdc  subtype=0  is_svc=0  serial=NONE  state=enable
[1]:     c0-0c0s4l1  type=rt_kpdc  subtype=0  is_svc=0  serial=NONE  state=enable

Now Disabled (Previously Enabled) Components (detailed):
------------------------------------------------------------


Now Disabled (Previously Empty) Components (detailed):
------------------------------------------------------------


Now Empty (Previously Enabled) Components (detailed):
------------------------------------------------------------

[0]:   NEW: c0-0c0s4  type=rt_l0  subtype=0  is_svc=0  serial=NONE  state=empty
[0]:   OLD: c0-0c0s4  type=rt_l0  subtype=0  is_svc=0  serial=00:40:A6:82:C4:2B
state=enable
[1]:    NEW: c0-0c0s4n0  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=empty
[1]:    OLD: c0-0c0s4n0  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=enable
[2]:    NEW: c0-0c0s4n1  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=empty
[2]:    OLD: c0-0c0s4n1  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=enable
[3]:    NEW: c0-0c0s4n2  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=empty
[3]:    OLD: c0-0c0s4n2  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=enable
[4]:    NEW: c0-0c0s4n3  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=empty
```

```
[4]:    OLD: c0-0c0s4n3  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=enable
[5]:    NEW: c0-0c0s4a0  type=rt_aries  subtype=0  is_svc=0  serial=NONE
state=empty
[5]:    OLD: c0-0c0s4a0  type=rt_aries  subtype=0  is_svc=0  serial=NONE
state=enable
[6]:     NEW: c0-0c0s4a0l00  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=empty
[6]:     OLD: c0-0c0s4a0l00  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=enable
[7]:     NEW: c0-0c0s4a0l01  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=empty
[7]:     OLD: c0-0c0s4a0l01  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=enable
[8]:     NEW: c0-0c0s4a0l02  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=empty
[8]:     OLD: c0-0c0s4a0l02  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=enable
[9]:     NEW: c0-0c0s4a0l03  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=empty
[9]:     OLD: c0-0c0s4a0l03  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=enable

Now Enabled (Previously Empty) Components (detailed):
------------------------------------------------------------

[0]:   NEW: c0-0c0s5  type=rt_l0  subtype=0  is_svc=0  serial=NONE  state=enable
[0]:   OLD: c0-0c0s5  type=rt_l0  subtype=0  is_svc=0  serial=00:40:A6:82:C4:2B
state=empty
[1]:    NEW: c0-0c0s5n0  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=enable
[1]:    OLD: c0-0c0s5n0  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=empty
[2]:    NEW: c0-0c0s5n1  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=enable
[2]:    OLD: c0-0c0s5n1  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=empty
[3]:    NEW: c0-0c0s5n2  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=enable
[3]:    OLD: c0-0c0s5n2  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=empty
[4]:    NEW: c0-0c0s5n3  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=enable
[4]:    OLD: c0-0c0s5n3  type=rt_node  subtype=0  is_svc=0  serial=NONE
state=empty
[5]:    NEW: c0-0c0s5a0  type=rt_aries  subtype=0  is_svc=0  serial=NONE
state=enable
[5]:    OLD: c0-0c0s5a0  type=rt_aries  subtype=0  is_svc=0  serial=NONE
state=empty
[6]:     NEW: c0-0c0s5a0l00  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=enable
[6]:     OLD: c0-0c0s5a0l00  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=empty
[7]:     NEW: c0-0c0s5a0l01  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=enable
[7]:     OLD: c0-0c0s5a0l01  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=empty
```

```
[8]:     NEW: c0-0c0s5a0l02  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=enable
[8]:     OLD: c0-0c0s5a0l02  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=empty
[9]:     NEW: c0-0c0s5a0l03  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=enable
[9]:     OLD: c0-0c0s5a0l03  type=rt_aries_lcb  subtype=0  is_svc=0  serial=NONE
state=empty


Modified Components (detailed):
------------------------------------------------------------


========== End of File ==========
```

Sample Inventory Data File

The following example shows the inventory data for a blade and its respective nodes:

```
<?xml version="1.0"?>
<hwinv format="combined" version="1.6">
<reported_modules count="4"/>
<reported_nodes count="12"/>
<module_list>
    <module>
        <cname>c0-0c0s0</cname>
        <board_string>MARBIO</board_string>
        <board_type>10</board_type>
        <pdc_type>1</pdc_type>
        <board_seep>
            <part-number>100490604</part-number>
            <rev>REV_G</rev>
            <serial-number>TU16130011</serial-number>
            <type>Cascade IBB</type>
            <date>2016-06-21</date>
            <field11>@</field11>
        </board_seep>
        <network_list>
            <network_chip>
                <cname>c0-0c0s0a0</cname>
                <ordinal>0</ordinal>
                <type>aries</type>
                <rev>1.1</rev>
            </network_chip>
        </network_list>
        <XPDC1_board_seep>
            <part-number>100605903</part-number>
            <rev>REV_F</rev>
            <serial-number>TL15300031</serial-number>
            <type>Cascade XPDC</type>
            <date>2015-08-20</date>
            <field11>:</field11>
        </XPDC1_board_seep>
        <XPDC2_board_seep>
            <part-number>100605903</part-number>
            <rev>REV_F</rev>
```

```
    <serial-number>TL16110112</serial-number>
    <type>Cascade XPDC</type>
    <date>2016-06-21</date>
    <field11>:</field11>
</XPDC2_board_seep>
<node_list>
    <node>
        <cname>c0-0c0s0n1</cname>
        <ordinal>1</ordinal>
        <status>enabled</status>
        <nic>1</nic>
        <nid>1</nid>
        <sockets>1</sockets>
        <dies>1</dies>
        <cores>8</cores>
        <hyper_threads>2</hyper_threads>
        <class>INTELSB8Core</class>
        <speedGHZ>2.6</speedGHZ>
        <die_list>
            <die>
                <ordinal>0</ordinal>
                <cpuid>0x000206d7</cpuid>
                <family>006h</family>
                <model>2dh</model>
                <stepping>7h</stepping>
                <step_str>C2</step_str>
            </die>
        </die_list>
        <memory>
            <sizeGB>64</sizeGB>
            <num_dimms>4</num_dimms>
            <ddr_type>3</ddr_type>
            <dimm_list>
                <dimm>
                    <ordinal>0</ordinal>
                    <mfg>Samsung</mfg>
                    <partnum>M393B2G70BH0-CK0  </partnum>
                    <speed>PC3-12800</speed>
                </dimm>
            </dimm_list>
        </memory>
    </node>
    <node>
        <cname>c0-0c0s0n2</cname>
        <ordinal>2</ordinal>
        <status>enabled</status>
        <nic>2</nic>
        <nid>2</nid>
        <sockets>1</sockets>
        <dies>1</dies>
        <cores>8</cores>
        <hyper_threads>2</hyper_threads>
        <class>INTELSB8Core</class>
        <speedGHZ>2.6</speedGHZ>
        <die_list>
            <die>
                <ordinal>0</ordinal>
```

```
                        <cpuid>0x000206d7</cpuid>
                        <family>006h</family>
                        <model>2dh</model>
                        <stepping>7h</stepping>
                        <step_str>C2</step_str>
                    </die>
                </die_list>
                <memory>
                    <sizeGB>64</sizeGB>
                    <num_dimms>4</num_dimms>
                    <ddr_type>3</ddr_type>
                    <dimm_list>
                        <dimm>
                            <ordinal>0</ordinal>
                            <mfg>Samsung</mfg>
                            <partnum>M393B2G70BH0-CK0  </partnum>
                            <speed>PC3-12800</speed>
                        </dimm>
                    </dimm_list>
                </memory>
            </node>
        </node_list>
    </module>
</module_list>
</hwinv>
```

Running HSS Providers

Once the HSS providers are configured, they can be started using the ‘systemctl’ command. The first step is to start the DAI Volt service, followed by the DAI Adapter service.

.?>systemctl start dai-volt

.?>systemctl start dai-adapter

Event Simulator

Event Simulator, EventSim, is the application we developed for emulating a Cray HSS system. This application generates a stream of RAS or Environmental events for the HSS providers to consume. Here is a sample configuration for EventSim:

```
{
   "SystemManifest":"/opt/ucs/SystemManifest50K.json",
   "SensorMetadata":"/opt/ucs/CraySensorMetaData.json",
   "RASMetadata":"/opt/ucs/CrayEventMetaData.json",
   "exchangeName"  : "cray_sim",
   "uri"           : "amqp://127.0.0.1",
   "eventCount"    : 10,
   "eventRate"     : 10000,
   "eventRatioSensorToRas"    : 1,
   "randomizerSeed"    : "236"
}
```

Configuration Key	**Description**
SystemManifest	Location of the UCS System Manifest of the system EventSim must emulate
SensorMetadata	Location of the Sensor Metadata file that contains the Environmental Sensor information of a Cray System
RASMetadata	Location of the RAS metadata file that contains the RAS information of a Cray System
exchangeName	Exchange name of the AMQP server on which the events generated by EventSim are published
uri	Location where the AMQP server is running
eventCount	Total number of events to be published when RegEx based location is not passed while invoking EventSim. When RegEx based location is passed, eventSim generates this many events for each location matching with the RegEx
eventRate	Number of events that eventSim generates per second
eventRatioSensorToRas	When no command line option is specified, eventSim generates a combination of RAS and Environmental data with this ratio of Environment to RAS events.
randomizerSeed	A synthetic randomizer seed to used to trigger a random set of simulated data, but at the same time ensuring reproducibility. This paramter will only take a string of an integer. See above example.

Executing EventSim

EventSim can be launched from the command line with the following command line arguments:

.?>eventsim [-bhrsv] [-l=<locationRegex_>]

EventSim has four modes of operation:

1. ‘-r’ → Generate RAS* events only
2. ‘-e’ → Generate Environmental* events only
3. Generate a combination of RAS* & Environmental* events
    1. This is the default mode of operation
4. ‘-b’ → Generate boot Sequence of all the nodes in the system manifest

*Note: The RAS & Environmental events can be generated against a specific Regex using the ‘-l’ option

Command Line Interface

UCS also supports a simple command line interface that can be used to query the system information and status. Currently it supports querying three types of data:

1. ‘events’ - RAS
2. ‘env’ - Environmental
3. ‘inventory’ - Inventory

The required positional arguments are:

start_time provide the start time. The preferred format for the date is "YYYY-MM-DD HH:MM:SS.[f]" to ensure higher precision

end_time provide the end time. The preferred format for the date is "YYYY-MM-DD HH:MM:SS.[f]" to ensure higher precision

It also supports advanced filtering which can be enabled by using the following optional command line arguments:

Argument	**Description**
-h, --help	show help message and exit
--limit LIMIT	Provide a limit to the number of records of data being retrieved. The default value is 100
--location LOCATION	Filter all event data for a given location.
--event_type EVENT_TYPE	Filter all event data for a given event_type. The event_type data depends on the descriptive name of the events in the RASMetadata.json. Example: RasGen or Ras
--severity SEVERITY	Filter all event data for a given severity {INFO, FATAL, ERROR, CRITICAL}. This option does not take wildcards or RegEx
--output OUTPUT	Store all the output in JSON format to the provided file path else to default file location /tmp/canned_output
--timeout TIMEOUT	Timeout value for the command execution. This is currently not implemented and always uses a default of 900s

RAS Data

The following command can be issued to query the RAS events stored in the system:
.?> ucs view events start_time end_time

Environmental Data

.?> ucs view env start_time end_time

Inventory Data

Inventory contain two sub-options:

1. ‘info’ - Gives us the inventory information for a single location.
2. ‘changes’ - Gives us all the changes that have gone into a particular location.

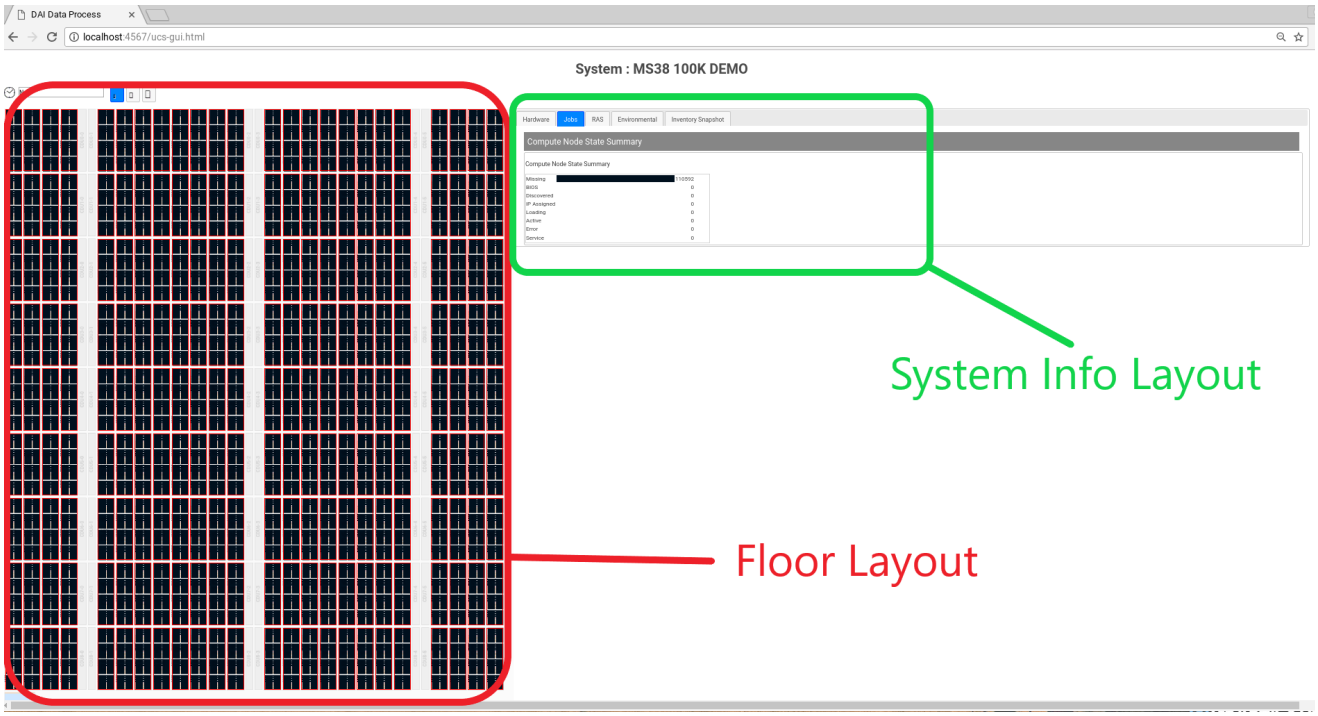
.?> ucs view inventory {info, changes} start_time end_time

Graphical User Interface

We can visualize the system using our web based GUI interface, located at “127.0.0.1:4567”

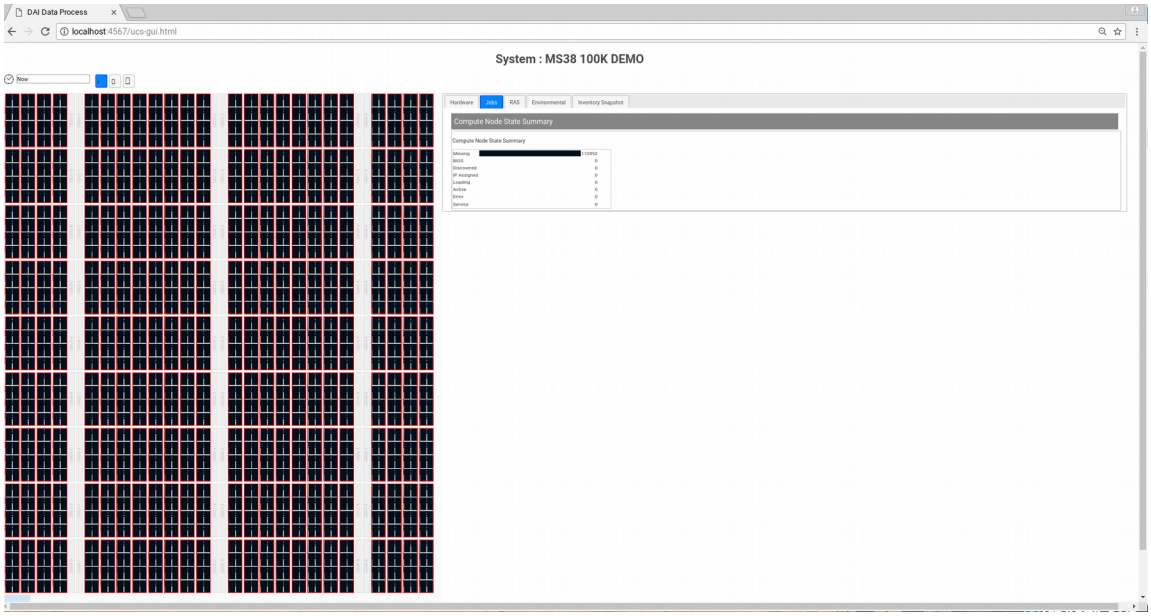
The GUI is divided into two main sections:

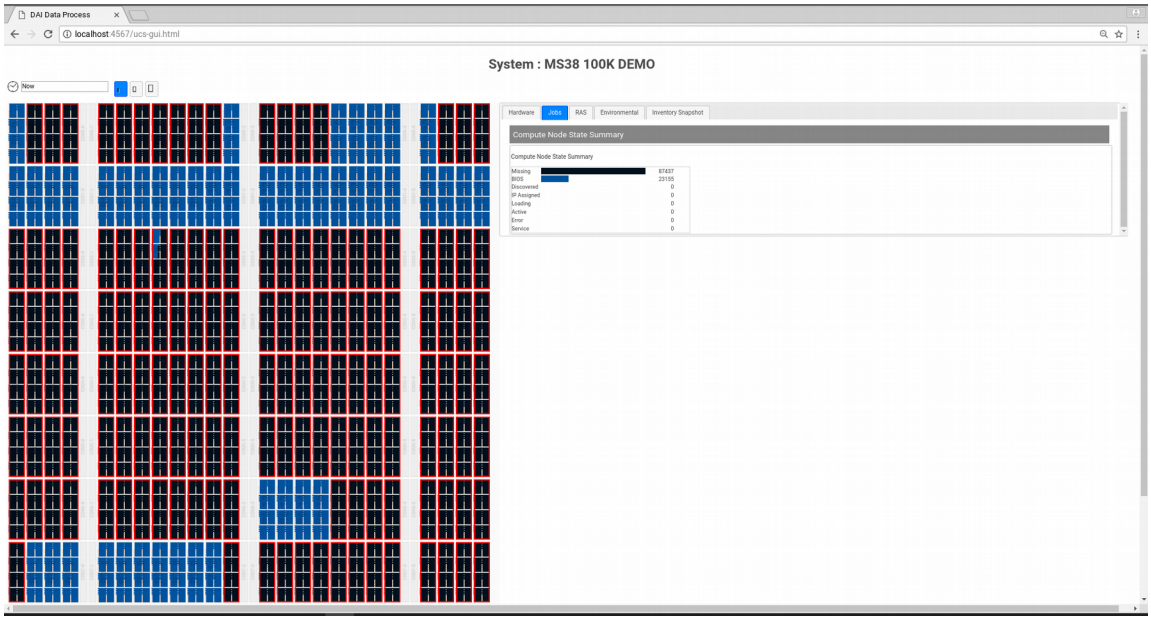
1. Floor Layout
2. System Info Layout



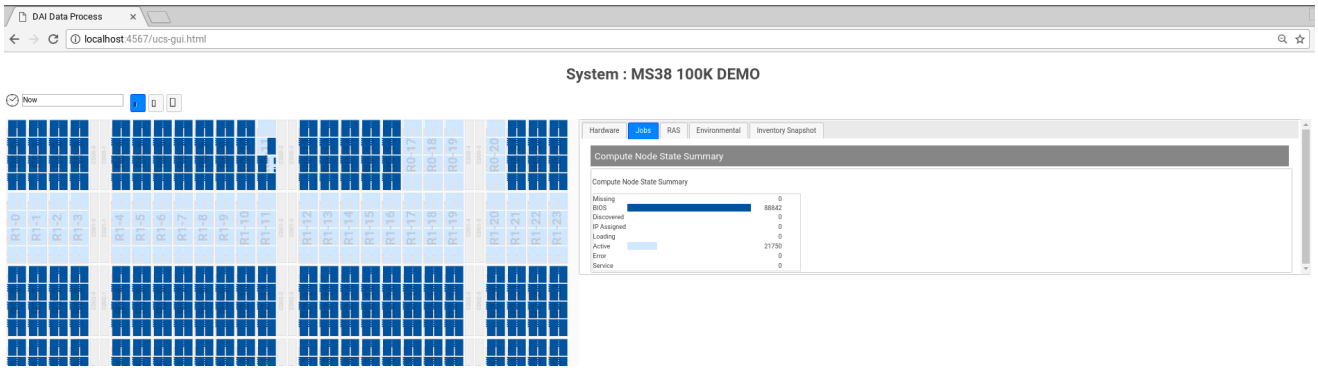
Drawing 1: UCS GUI Layout

When the system is booting, we can perceive the nodes within the floor layout transition into different states, with the change of their colors. This can be seen below in the three diagrams of the floor layout, as the nodes transition from Missing, to Booting, and finally to Available.

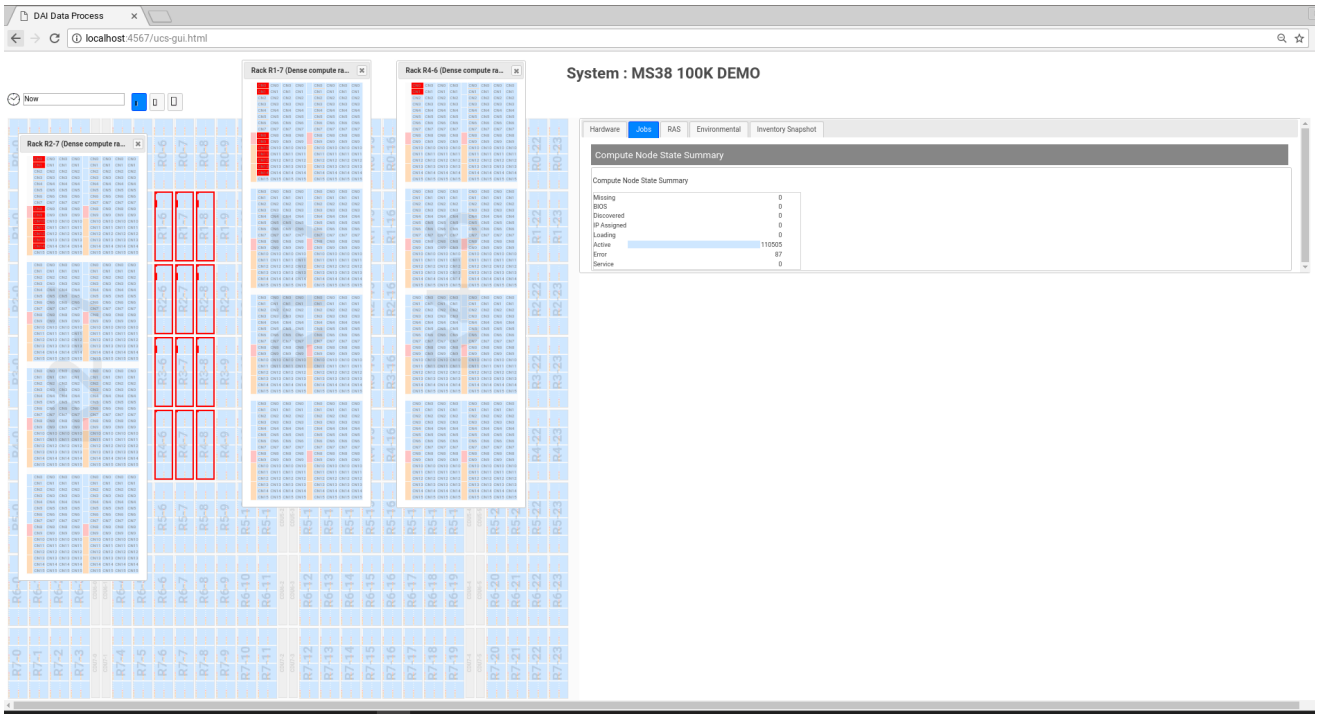




Drawing 3: Nodes transitioning to Booting state

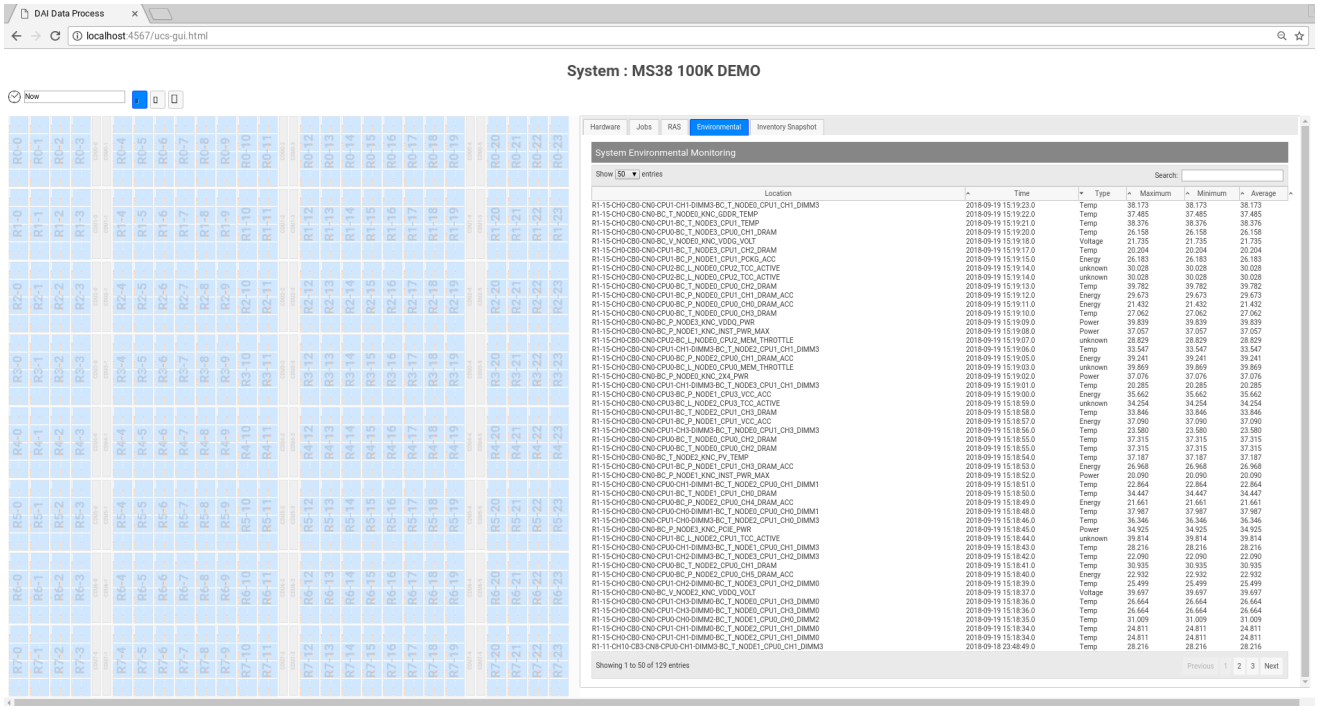


The GUI can also be used to montior the RAS events & Environmental data generated by the system. The node in error will be highlighted in red. The RAS events are located under the RAS tab as shown below:



Drawing 5: RAS events and nodes in Error state

The environmental data can also be seen within GUI under the Environmental tab of the System info layout as shown below:



Drawing 6: System Environmental data