Getting started with ivy

Allart Ian Vogelesang
ian.vogelesang@hitachivantara.com

2019-09-20

Audience – new ivy users

- Recommended to go through the “Introduction to ivy” presentation first.
  - Management view - reduced cost, improved data quality, faster time-to-market, etc.
- This presentation is intended to get new ivy users started.
  - Explain essential concepts
  - First look at ivy engine control statements
    - `[Hosts], [CreateWorkload], [CreateRollup], [EditRollup], [Go]`
  - Guided tour of some key aspects of ivyscript programs.
- After reviewing this material, the new user will be better prepared to start exploring existing ivyscript programs, and to start coding ivyscript.
- To learn more, see the comprehensive “ivyscript reference” and “programming the ivy engine” presentation materials.

How ivy was installed

- ivy family executables were put in a folder somewhere.
- This folder was put into the `PATH` for both foreground and background processes in Linux using a script in `/etc/profile.d`
  - `ivydriver` starts on each test host via `ssh` as a background process where “bash profile” type scripts don’t apply.
- Certificate-based `ssh` authentication as root was set up between the ivy central host and all test hosts.
- The `InquireAbout` executable was marked “setuid” and owned by root in order run as root to issue SCSI Inquiry commands to raw LUNs. There is a handy script to do this.
  - This lets ordinary users use `showluns.sh` independent of ivy.
- Note: You must be root to run ivy, since it opens raw LUNs to do I/O.

ivy family executables

1. `LUN_discovery` suite `showluns.sh`, `InquireAbout`, etc.
   - Open source https://github.com/Hitachi-Data-Systems/LUN_discovery
   - Decodes Hitachi storage SCSI Inquiry attributes, e.g. `port`, `LDEV`, `LDEV_type`, `Pool_ID`, etc.
2. ivy `ivy`, `ivydriver`
   - Open source https://github.com/Hitachi-Data-Systems/ivy
3. ivy command device connector `ivy_cmdev`
   - Hitachi proprietary. Not open source. Restricted to authorized internal Hitachi lab use with license key.
   - The idea when writing `ivy_cmddev` was to have the lightest possible touch on the test host with the command device, so derived data is computed in the ivy central host, which is open source.
   - Other storage vendors can see the interface in ivy's source code to `ivy_cmddev`, and are encouraged to develop a similar ivy configuration and real-time performance monitoring interface for their own products, and to contribute to the ivy project overall.

`showluns.sh`

- The `LUN_discovery` SCSI Inquiry tool suite is its own separate open source project on github, but is installed along with, and serves as a front-end to ivy.

- Try typing "`showluns.sh`".

- This produces a csv file with a header line with LUN SCSI Inquiry attribute names, and one data line for each `/dev/sdxx` LUN with the corresponding attribute values for that LUN.

```
hostname,LUN_name,LDEV,port, …
sun159,/dev/sdc,00:00,1A, …
sun159,/dev/sdd,00:01,2A, …
```

- The `showluns.sh` output csv file only shows those attribute names for which at least one LUN provided a non-empty value.
  - And thus depending on what kinds of LUNs the SCSI Inquiry tool "sees", you may get a different set of attribute names (csv columns) appearing/disappearing.

- To provide support for a different vendor's architecture and terminology in ivy, all you need is a tool that provides the equivalent csv file.

LUN attributes

- The attributes of the LUNs that were discovered using `showluns.sh` become what you select on in ivy.

- For the vast majority of attribute names, the stock ivy functionality is all you need to select test LUNs, e.g. `[select] << "port" : [ "1A", "3A" ] >>`

“raw strings” start with << and end with >>, making it easy to include quote marks within a string

- ivy does provide a couple of “specially implemented for Hitachi” attribute value matching functions to recognize shorthand for LDEV ranges and PG name ranges.

What if a Linux reboot changes `/dev` names?

- Nothing happens.
- You don’t select your test configuration by LUN name, instead, you select by LUN attribute value.
- But if you *really did* want to select on /dev name, you can do that:

```
[select] << "LUN_name" : [ "/dev/sdb", "/dev/sdc" ] >>
```

All discovered LUNs -> available test LUNs

- Each test host, when it first wakes up, runs `showluns.sh`, and sends the output to the central host. The aggregated data from all test hosts forms **all discovered LUNs**.
- **All discovered LUNs** includes information on all `/dev/sdxx` LUNs, including test host boot volumes.
- On the `[hosts]` statement, there must be a `[select]` clause that specifies at least one of `serial_number` or `vendor`. This is intended to prevent accidentally writing on test host boot volumes.
- **All discovered LUNs** is filtered and all LUNs matching the `[hosts]` statement `[select]` clause form “**available test LUNs**”.
- **All discovered LUNs** is never used again. Later when we create workloads, this selects from **available test LUNs**.

ivyscript programs start with the `[hosts]` statement

List of test host hostnames
`sun159`, `testhost1`, `testhost2`, …, `testhost8`

`serial_number` uses the default built-in attribute matcher

```
[Hosts] "sun159, testhost[1-8]"
      [Select] "serial_number : 123456, LDEV : 00:00-01:FF";
```

The `[select]` query is used to filter **all discovered LUNs** to arrive at **available test LUNs**.

`LDEV` uses a special case Hitachi specific attribute matcher that recognizes LDEV ranges.

- Select clauses accept official well-formed JSON, but ivy relaxed JSON lets you omit outer braces {}, omit quote marks around things ivy recognizes.

[CreateWorkload]

We create a "flock" of workloads all with this name on a selected group of available test LUNs.

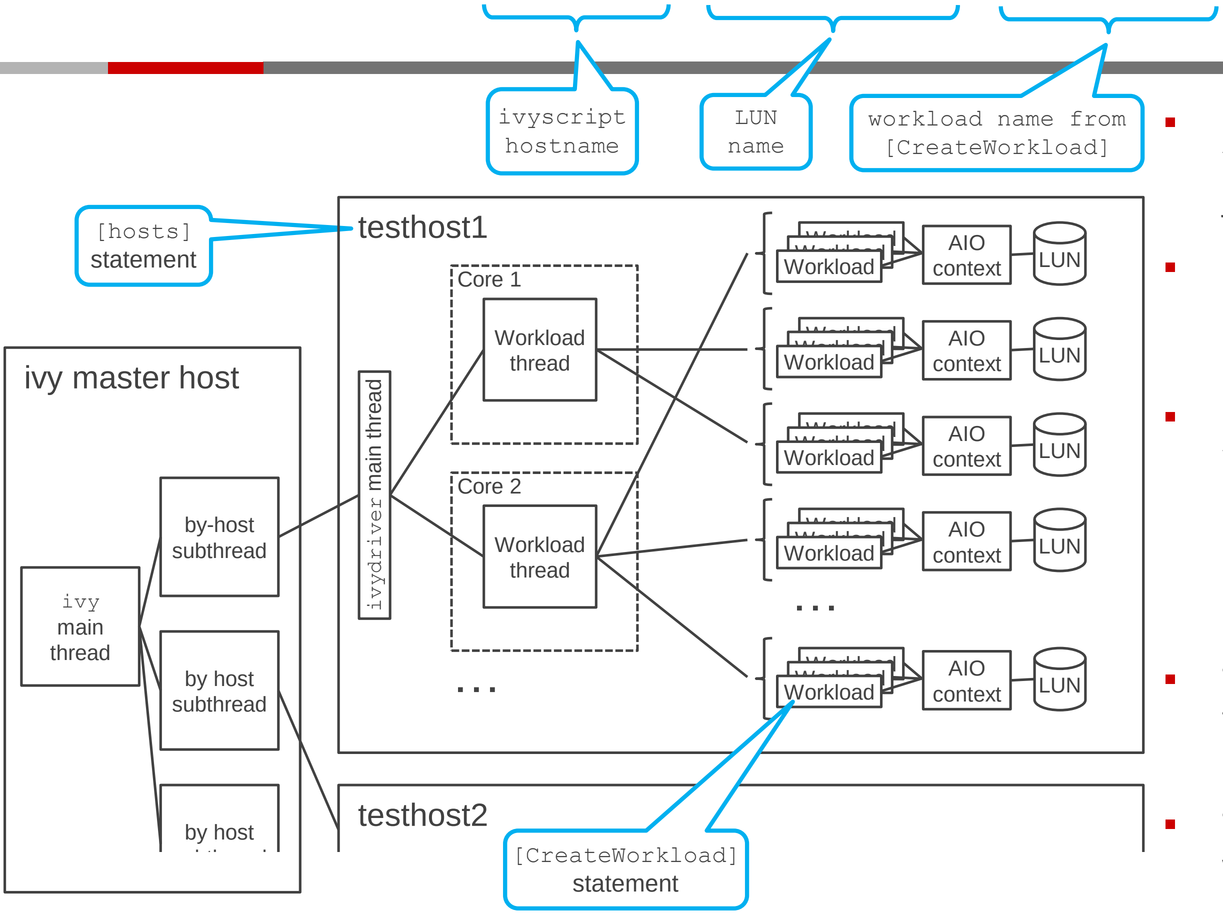
Selects from **available test LUNs**.

`random_steady`, `random_independent`, or `sequential`

```
[CreateWorkload] "fluffy"
     [select] << "port" : "1A" >>
     [iosequencer] "random_independent"
     [parameters] "IOPS=max, blocksize=8KiB, fractionRead=100%,maxTags=32";
```

Each I/O sequencer type will have its own set of parameters or settings that it uses.

WorkloadID, e.g. sun159+/dev/sdc+frantic



- Every workload is identified by its `WorkloadID`, which has three parts joined with plus signs.
- `ivyscript_hostname` is what you called it on the `[hosts]` statement, which could be an alias or IPV4 dotted quad.
- For training / ivy development purposes you can run a “fake” ivy multi-host configuration on one host using the IPV4 address or an alias as a second host. Each resulting instance of `ivydriver` is unaware of any others on the same host.
- The first two parts of a WorkloadID together form the AIO context LUN ID `ivyscript_hostname+/dev/sdxx`
- The last part of the `WorkloadID` is the workload name from “create workload”

Rollups are key to how the ivy engine works

HITACHI
Inspire the Next

- Rollups are used to group workloads, to navigate between (in both directions), say, a port name `1A` and those workloads on LUNs mapping to port `1A`.
  - By-rollup csv files show data rolled up by rollup instance from results by individual `WorkloadID`.
  - [EditRollup] uses rollups in the other direction, to send, for example, `IOPS=1000` to the workloads on port `1A`.
- When driving multiple subsystems: `[CreateRollup] "serial_number+port";`
  - `serial_number` and `port` must be valid LUN attribute names.
- In every rollup, each `WorkloadID` appears in exactly one rollup instance.
  - `"Serial_Number+Port"` Rollup type
    - `"410123+1A"` Rollup instance.
      - `"sun159+/dev/sdx+workload_name", "cb28+/dev/sdy+workload_name"`

List of WorkloadIDs comprising the rollup instance

You make rollups for four reasons

1. To get an output csv file with a csv folder by rollup type (e.g. `port`) and csv files by rollup instance (e.g. `1A`)
   - This is how you get custom "sliced & diced" data.

2. To perform IOPS dynamic feedback control (`dfc=PID`) at the granularity of the rollup instance.
   - One of the demos shows measuring IOPS at MP 50% busy at the granularity of the MPU, meaning to vary the IOPS up and down separately/independently for each MPU to achieve 50% busy MP cores in that MPU.

3. To identify a valid measurement period at the granularity of the rollup instance using `measure`.
   - For the valid period, when measuring at the granularity of the rollup instance, the data for each rollup instance individually met the +/- accuracy % criteria for a valid measurement. (For every `port`, the individual data for that port met the +/- accuracy criterion.)

4. To validate the test configuration as operating correctly
   - E.g. Validates that the number of ports reporting was what you expected
   - E.g. Validate that no one port had an IOPS too far below the highest IOPS seen on any port.

Statements – `[CreateRollup]`

- `[CreateRollup] "port" [nocsv] [quantity] 64 [MaxDroop] "20%";`
- Every workload appears in exactly one instance of every rollup.
- There is always an "`all`" rollup which only has one instance "`all`".
  - For example `[select] "all=all";`
- `[nocsv]` – Optional - suppresses creation of `port` output csv files for this rollup.
- `[quantity] 64` – Optional - marks the test result invalid if there aren’t 64 `port` instances reporting data .
- `[MaxDroop] "20%"`
  - Optional - marks the test result invalid if any one instance of the rollup has an IOPS more than 20% below that of the fastest instance.
  - Useful to catch the situation where, say, one port is running slowly compared to the others because it’s in error recovery.

[EditRollup]

- The rollup concept gives you great flexibility to send out parameter setting edits to selected workloads.
- First create a rollup, e.g. `[CreateRollup] "LDEV_type";`
- Then you can say, for example

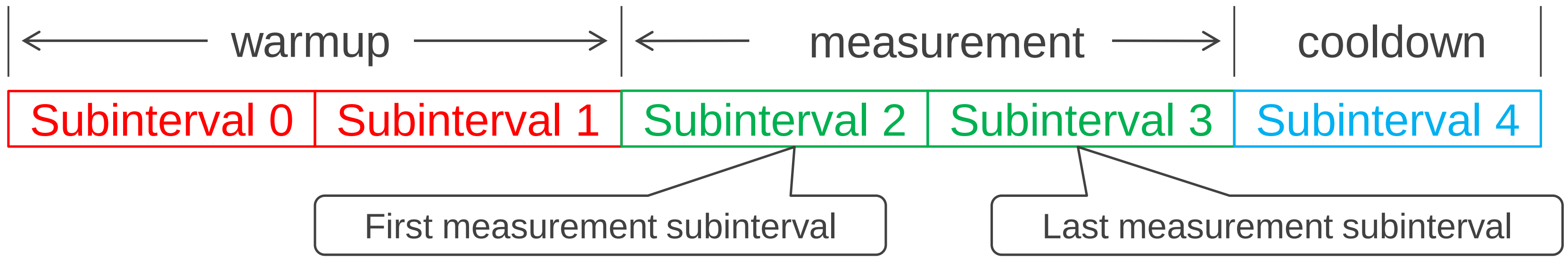
  ```
  [EditRollup] "LDEV_type=DP_vol" [parameters] "IOPS=1000";
  ```

- To send a parameter change globally, say

  ```
  [EditRollup] "all=all" [parameters] "IOPS=1000";
  ```

- The underlying ivy engine C++ API edit rollup call is what’s also used internally within the ivy engine to send out Dynamic Feedback Control IOPS edits in real time at the granularity of the rollup instance while the workloads are running and driving I/O.

An ivy test step – a series of subintervals



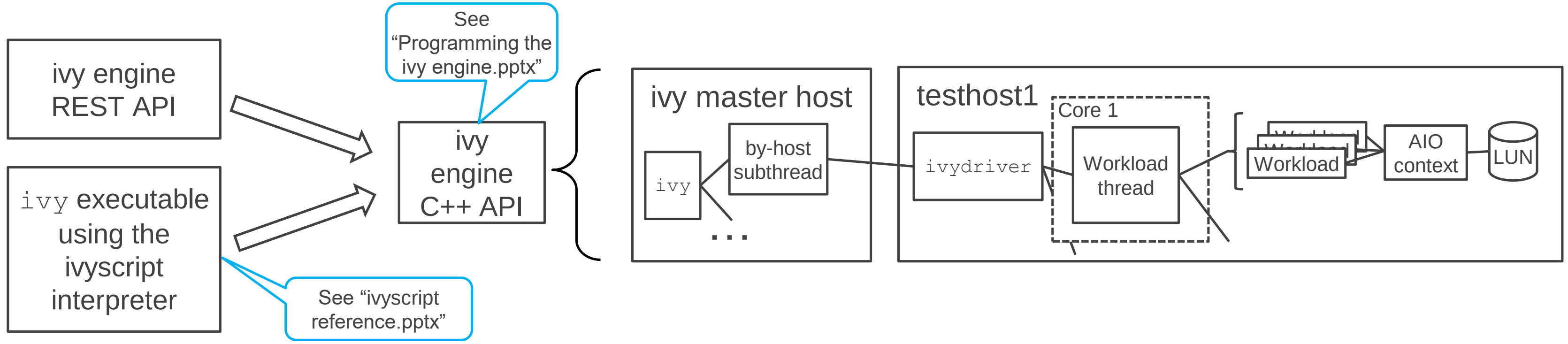
- There may be zero or more warmup subintervals (`warmup_seconds`), at least one measurement subinterval (`measure_seconds`), and zero or more cooldown subintervals.
- Without the `measure` feature, warmup and measurement run for a fixed number of subintervals.
- When using the `measure` feature, e.g. `measure = service_time_seconds` with `accuracy_plus_minus = 1%`, `warmup_seconds` and `measure_seconds` become minimums will be extended as long as necessary (up to `timeout_seconds`) to reach the +/- target accuracy.
- Either way, if a command device connector is available, cooldown may be extended at zero IOPS using `cooldown_by_WP` and `cooldown_by_MP_busy`, which both default to `on`.

The default `[Go]` statement

- `[Go];`
  - Default `warmup_seconds = 5`
  - Default `measure_seconds = 60`
  - Default `subinterval_seconds = 5`
  - Default `cooldown_by_wp = "on"`
    - Runs at least one cooldown subinterval continuing to drive I/O.
    - If you have a command device and the proprietary command device connector software, more cooldown subintervals at `IOPS=0` (zero) continue until Write Pending in all CLPRs being used is empty.
  - Useful when you are developing an ivyscript workflow and you just want to see quick sample csv files.

These parameters support notation like `1:00:00` (one hour), or `30:00` (30 minutes).

The ivy engine and ivyscript



- The ivy engine is written in C++ and can be operated directly via the ivy engine C++ API.
- The ivy engine REST API and the ivyscript interpreter both show you "create workload", "create rollup", etc.
- In ivyscript, when you see a statement starting with a word in square brackets like `[Hosts]`, `[CreateWorkload]`, etc. this reflects an access to the corresponding underlying ivy engine C++ API call that you are making to operate the ivy engine, and the remainder of the statement shows which operands are being provided, e.g. `[parameters]`.
  - `[CreateWorkload]` and `[ create workload ]` are equivalent.
- Every time you run an ivyscript program, you get a log file of all the underlying ivy engine C++ API calls that were performed in the course of running the ivyscript program.

ivyscript

- Ivyscript is a bare-bones (very basic) scripting language that looks like C/C++.
- Three types: `string`, `int`, and `double`.

  (floating point)

- The usual if statements, for loops, nested code blocks, user defined functions, etc.
- Special statements starting with square-brackets tokens like `[hosts]` are where ivyscript exposes the underlying ivy engine C++ API call.
- ivyscript and the ivy engine C++ API now have `ivy_engine_get()` and `ivy_engine_set`() functions, but many older individual ivyscript ivy engine built-in accessor functions to get things from the ivy engine still work and now map onto the appropriate ivy engine C++ API `ivy_engine_get`() call.
  - E.g. `ivy_engine_get("summary_csv")` retrieves the filename of the summary csv file.

Our first “config discovery” ivyscript program

ivyscript string expression for set of test hosts.
Here we are giving the ivy engine a string constant.

```
[hosts] "table, chair[1-6]" [select] "serial_number : 123456";
```

string expression for select clause to pick **available test LUNs**.

An ivyscript program is a series of statements, and each statement ends with a semicolon ;

- This starts `ivydriver` on all the test hosts, and selects available test LUNs, and it makes csv files of all discovered LUNs and available test LUNs.
- With a command device, you get subsystem configuration csv files, and the description of LUNs in available test LUNs will be augmented with config info from the subsystem.
- It’s an easy way to confirm your test setup.

Looping over workload parameter settings

HITACHI Inspire the Next

```
[hosts] "table, chair[1-6]" [select] "serial_number : 123456";
```

Create a flock of workloads each named “steady”

This null select creates a workload on **all available test LUNs**.

`random_steady`, `random_independent`, or `sequential`

```
[CreateWorkload] "steady"
   [select]        ""
   [iosequencer]   "random_steady"
   [parameters]    "IOPS=max, fractionread=100%, maxTags=32";
```

`100%` and `1.0` mean the same thing and are interchangeable in ivy.

```
[Go!] << blocksize = (4KiB, 8192, 16KiB, 32KiB, "64 KiB"), measure_seconds = 30 >>;
```

"raw strings" are character strings starting with << and ending with >>, which makes it easy to have embedded quotes .

If there is no space between "4" and "KiB", then putting quotes around 4KiB is optional.

Example of embedded quotes inside a raw string.

Same thing but looping in ivyscript

```
[hosts] "table, chair[1-6]" [select] "serial_number : 123456";

[CreateWorkload] "steady"
   [select]       ""
   [iosequencer] "random_steady"
   [parameters]  "IOPS=max, fractionread=100%, maxTags=32";

int blocksizeKiB;

for blocksizeKiB = { 4, 8, 16, 32, 64 }
{
   [EditRollup] "all=all" [parameters] "blocksize = \"" + string(blocksizeKiB) + "KiB\"" ;

   [Go!] "stepname=\"" + string(blocksizeKiB) + " KiB\", measure_seconds = 30";
};
```

Create a flock of workloads each named “steady”

This null select creates a workload on **all available test LUNs**.

`random_steady`, `random_independent`, or `sequential`

Traditional C-style for loops are also supported

“all=all” selects all workloads

Building a string that looks like “`blocksize = 4 KiB`”

`[EditRollup]` sends a parameter update to selected workloads.

The `stepname` shows up in the output csv files to auto-populate the legend for a data series.

Makes it easy to make Excel charts.

[CreateRollup] examples

```
[hosts] "table, chair[1-6]" [select] "serial_number : 123456";
[CreateWorkload] "steady"
   [select]       ""
   [iosequencer] "random_steady"
   [parameters]  "IOPS=100, fractionread=100%, maxTags=32";
[CreateRollup] "Port";
[CreateRollup] "Serial_Number+Port";
[CreateRollup] "MPU";  // only with a command device
[CreateRollup] "host+LUN_name+workload";  // same as workloadID
[CreateRollup] "workloadID";

print("Rollup structure:\n" + show_rollup_structure() + "\n");

[Go]  "stepname=step_eh, warmup_seconds = 5, measure_seconds = 5";
```

ivyscript built-in function accessing the ivy engine.

Retrieve result of a test step, to decide what to do next

- Assume we would like to retrieve the overall IOPS value from test step 0, in order to decide what to do in step 1.

Always works to use ivy engine accessor built-in functions to generate summary csv file name.

```
string summary_filename = ivy_engine_get("summary_csv");

double step0_IOPS = double(csv_cell_value(summary_filename,0,"Overall IOPS"));

string s = "step 0 result - overall IOPS = " + string(step0_IOPS) + "\n";

print (s); log(masterlogfile(),s);
```

ivy csv utility returning string value of what was between the commas in the requested row and column of the csv file.

- ivy csv utilities let you access a csv file like a spreadsheet.
  - ivy test steps ([Go] statements) are numbered from zero, and within a test step, subintervals are numbered from zero.
  - ivy csv utilities number the csv header line as line number -1 (minus one). This means the row number is the test step number in summary csv files, and the row number is the subinterval number for by-subinterval test step detail csv files.
  - You typically refer to columns by the text used in the header line with column title text, but you can also retrieve by column number.

End of guided tour.

- Enjoy discovering the new things you can do in ivy.

HITACHI
Inspire the Next