High Level Configuration

As it stands now

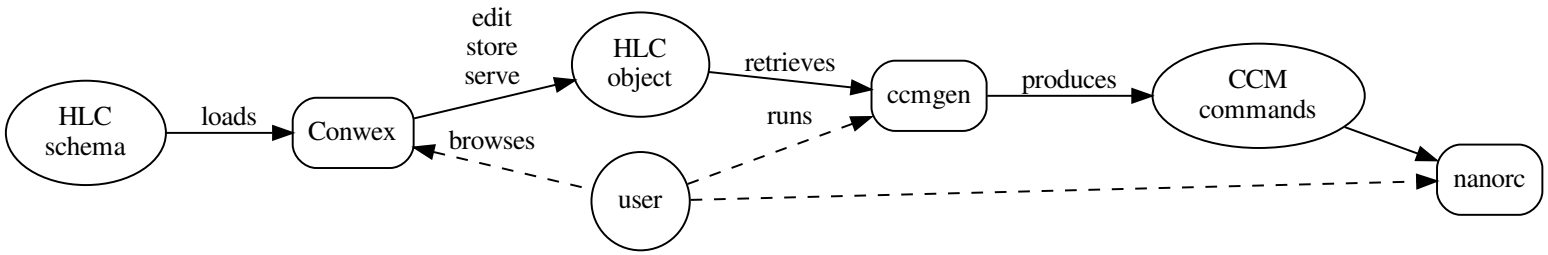
Brett Viren

June 29, 2021

outline

- desired setup
- current generator integration problems
- propose generator API

Want something like:



- High-Level Configuration (HLC) schema loaded as JSON Schema
  - Schema encodes generator input expectations.
- Conwex UI generated from schema, validates user edits.
- Single, generic `ccmgen` calls specific generator, inflates HLC object to CCM command set.
  - We focus in this point of integration today.
- `nanorc` dispatches commands as directed by user.

Current (non) integration with generators

- Generators do not define any schema for their input.
- Take input in the form of CLI options/arguments.
- No common Python function API for generator module.

To do anything with this,

- moo schema must be manually constructed by reading generator code
- some kind of "hacky" post processing of HLC object needed.
  - eg, use moo templates to **generate a shell script calling a python script** which transforms HCL object to specific CLI options.

Problems with (non) intgegration

- Must manually divine generator input schema by examining its source code. Labor-intensive, error prone, moving goal posts.
- The “hacky” post-processing needed has not yet been developed
  - In principle possible, but very awkward
- Novel generators will compound these integration problems and likely invent new ones.

We need a “generator API”.

- What might that look like?
- How to get there?

User experience

User wants simplicity, configuration must be named ∴ obvious user UI:

```
$ ccmgen <config-name>
$ nanorc <config-name>/
```

The `<config-name>` maps to:

- A Conwex URL for the HLC JSON object.
- The Python module+function providing generator.
  - And, thus also the original schema.
- Directory holding generator output / `nanorc` input.

Note this allows `ccmgen` to automatically be extensible to novel, compliant generators.

Generator API

- provide a HLC schema in moo Jsonnet
- factor generator into function following API:

```
def mygen(hlc: dict) -> dict:
    "My arbitrarily gnarly generator"
    if hlc["myparam"] == "foo":
        return cfg_with_foo(hlc)
    return cfg_with_bar(hlc)
```

- The `hlc` argument is pre-validated against scheam,
  - generator may trust it, avoid "clutter" of re-checking
- Caller responsibility to dispatch the returned CCM cmd objects
  - eg, save as JSON files in outdir for `nanorc`
- Python module+function path used to form `<config-name>`.

But, what about my old CLI?!?

```
$ myoldgen --option=42 [...20 others...]
$ myothergen --foo=bar [...20 other others...]
```

Possible options:

- Delete them, craft HLC JSON and use `ccmgen`.
- Add support to `ccmgen` CLI to set params (eg below)
- Use `jq` or others to generate "local" HLC JSON, use `ccmgen`.
- Manually maintain old CLI to reflect HLC schema.
- Develop a `moo` template to *render* schema to a CLI.

ccmgen can be for developers too

Does this cover 90% of what a developer/expert wants from a configuration generator CLI?

```
# use all local info, no Cowex
$ ccmgen --url my-local-hlc.json \
         --gen mymod.mygen \
         --name myjunk-nonstandard-ABC

# use nominal config + local some tweaks
$ ccmgen --override option=42 <config-name>
```

Proposed path to get there

- discuss, agree on strategy (all)
- develop `ccmgen` (bv)
- modify one (prob. Kurt's?) popular generator (bv)
- developer documentation (bv)
- other developers modify (all)