BSDCan 2015

UCL Working Group

allanjude@freebsd.org

Overview

The goal of this working group is to develop a template for all future configuration files that is both human readable and writable, but is also hierarchical, expressive, and programmatically editable.

Agenda

- Opening: What is UCL
- Presentation of work in progress: converting newsyslog and bhyve to UCL
- Discuss common requirements for configuration files
- Develop a common set of grammar/keys to work across all configuration files ('enabled' activates/deactivates each block, allows disabling default configuration without modifying the default files, ala pkg)

Agenda (Continued)

- Discuss layering (/etc/defaults/foo.conf -> /etc/foo.conf -> /etc/foo.conf.d/*.conf -> /usr/local/etc/foo.conf.d/*.conf)
- Discuss required features for management utilities (uclcmd)
- Identify additional targets to UCL-ify
- Develop a universal API for using libucl in various applications, simplify loading configuration into C structs (libfigpar?)

What is the Universal Configuration Language?

- Inspired by bind/nginx style configuration
- Fully compatible with JSON, but more liberal in what it accepts, so users do not have to write strict JSON
- Can Output UCL, JSON, or YAML
- Supports handy suffixes like k, mb, min, d
- Can be as simple or as complex as required
- Allows inline comments (# and /* multiline */)
- Validation and Schema support
- Supports includes, macros, and variables

Why UCL is great -- all of this is valid

```
param = value;

flag = true;
number = 10k
time = 0.2s

foo: bar

fiz = “buz”

freebsd = “best”;
freebsd = “greatest”;

array = [
     thing1,
     thing2,
]
```

```
key = “value”;

section {
    string = "something";
    subsection {
        jail {
            host = "somejail";
            port = 900;
        }
        jail {
            host = "otherjail";
            port = 901;
        }
    }
}
```

UCL Includes

The includes feature in libUCL is very rich:

- **try**: errors loading the file are non-fatal
- **glob**: treats the filename as a shell GLOB pattern and load all files that matches the specified pattern
- **url**: allow URL includes (required libfetch or libcurl)
- **sign**: UCL loads and checks the signature for a file from path named <FILEPATH>.sig. Trusted public keys should be provided for UCL API after parser is created but before any configurations are parsed.
- **priority**: specify priority for the included file
- Included keys with the same priority are merged, keys with a higher priority overwrite the older values

Work In Progress

Add UCL support to newsyslog:

- https://reviews.freebsd.org/D1548

Give bhyve a UCL config file:

- https://reviews.freebsd.org/D2448
- Example config files:

newsyslog: http://allanjude.com/bsd/newsyslog.ucl

bhyveucl: http://allanjude.com/bsd/vmconfig.ucl

bhyve.conf: http://allanjude.com/bsd/bhyve.conf

What makes a good config file?

- Easy to read
- Easy to write
- Easy to parse
- Easy to edit programmatically
- Organized (break keys into sections of related settings, rather than flat n:v pairs)
- Short(ish) but descriptive key names

Common Keys

We will discuss and develop a common set of keys to be used in all configuration files throughout the base system, and plan for the implementation of UCL support in various utilities.

- Enabled Flag - Allow default config entries to be disabled
- Versioning - config files change over time, some level of convertibility/backwards compatibility can be achieved

Namespace

pkg(8) currently uses “enabled” as a boolean to mark individual sections as on or off. We’d like to do that for every section in every utility, so what should we name it?

- enabled (possibly conflicts with utility)
- __enabled
- freebsd_enabled (utility used outside of freebsd)
- freebsd.enabled (caused issue with dot notation)
- freebsd { enabled = true }

Layering

libUCL has a rich “includes” framework.

Makes it easier to implement blah.d/ config fragments because the work is done by libucl, rather than having to teach each utility how to include files and handle things.

libUCL also has the ‘priorities’ system, which allows included files to optionally override settings from previous config files

/etc/defaults/foo.conf -> /etc/foo.conf -> /etc/foo.conf.d/*.conf -> /usr/local/etc/foo.conf.d/*.conf

Why Layering is Useful

Currently, FreeBSD ships with a default /etc/newsyslog.conf that rotates the default logs.

If you want to change the default number of copies of /var/log/messages that are retained, you have to modify this file.

When you next upgrade FreeBSD, maybe there is a new log file to rotate, now your file needs to be 3-way merged.

Includes have solved the problem of packages needing to add their log files to the config file.

More Layering

So, what we need to do is move the defaults to /etc/defaults/ and provide a way for users to override or disable the default settings.

Management Tools

I have written uclcmd, a utility that allows you to get, set, merge, and remove keys from UCL files. It can output via libucl in UCL, JSON, or YAML, or using its own code, output as shell assignment statements, optionally with additional variables to assist with recursion.

It currently does not operate on files in place, but outputs to stdout.

What other features does it need?

uclcmd: shellvar output example

# uclcmd get -f ../bhyveucl/bhyve.conf -ekl \|recurse

```
__keys="name cpus memory uuid console loader
loader_args loader_input device acpi
vmexit_on_hlt vmexit_on_pause disk network"
name="vm0001"
cpus=4
memory=4294967296
console="/dev/nmdm0001A"
loader="bhyveload"
loader_args=""
loader_input=""
acpi=true
vmexit_on_hlt=true
vmexit_on_pause=true
```

```
disk=[array]
disk__length=2
disk_0="/vm0001/disk0001"
disk_1={object}
disk_1__keys="type path
readonly"
disk_1_type="virtio-blk"
disk_1_path="
/vm0001/img0002"
disk_1_readonly=true
```

Targets to UCL-ify

- bhyve
- crontab
- iscsi / ctld
- autofs
- config(8)
- portsnap
- launchd
- devfs
- kernel conf
- dconf
- bsnmpd
- zfs properties

jail.conf

- need support for self-referencing variables like ${host.hostname} in path
- Has keys with dots in them (UCL lookup_path problem)

wpa_supplicant

- syntax is almost the same as UCL

devd.conf

- syntax doesn’t match well, will need work

login.conf (cap_mkdb)
pw.conf (pwd_mkdb)
mac.conf

Files not to ucl

rc.conf (maybe)
loader.conf
ssh / sshd
hosts
services
nsswitch.conf
pf.conf

autofs (both)
openpam

Universal API

The goal is to develop a universal API for loading UCL configurations into the existing C structs used internally by the target application, to simplify implementing UCL in more places.

Devin Teske suggested his libfigpar may be a good fit.

Comparing Methods

newsyslog:

- Reuse existing struct
- Not perfect, but usable
- Callback for each struct member, does validation
- Uses a table to map key names to callback functions
- More plumbing, create callback for each key

bhyve:

- bhyve originally used mix of global and local variables to store config
- Created new struct to be used everywhere
- Some validation is done during parsing, some at the end of parsing, and some when the value is first used
- ugly strcmp if ladder

bhyve config infrastructure

- bhyve loads /etc/defaults/bhyve.conf
- which loads /etc/bhyve.conf and /etc/bhyve.conf.d/*.conf
- Content in the two top level bhyve.conf populate the root namespace, which bhyve interprets are the “defaults”
- Content from bhyve.conf.d is automatically nested as nameofconf { }, as it is designed to be per-vm settings
- This is a break from the traditional .d model

Future Ideas

- "show running config" - A single compiled config for the entire system, each service in its own subkey
  - Possibly implement parsers for some config files that are not converted to ucl (sshd_config) so they can be included in the running config
- libuclnv - ucl -> nvlist, and nvlist -> ucl, by borrowing the libucl parser and emitter, but using nvlists for the internal representation
- Kyua regression tests for libUCL
- More simple tests for uclcmd