Consfigurator: Lisp declarative configuration management

Sean Whitton

DebConf22

Introduction

- Propellor, configuration management system using Haskell, written by Joey Hess, presented at DebConf14 and DebConf17.

Introduction

- Propellor, configuration management system using Haskell, written by Joey Hess, presented at DebConf14 and DebConf17.
- Key ideas:
  - using Haskell's type system to find configuration problems before any hosts are touched
  - user's configuration of their hosts is expressed in the same language as the tool, not YAML or something.

Introduction

- However
  - expressing configuration within the constraints of the type system often awkward and time consuming as compared with the number of bugs actually caught

Introduction

- However
  - expressing configuration within the constraints of the type system often awkward and time consuming as compared with the number of bugs actually caught
  - only one method to apply properties to hosts: connect, compile Propellor on remote side, apply all its properties

Introduction

- However
  - expressing configuration within the constraints of the type system often awkward and time consuming as compared with the number of bugs actually caught
  - only one method to apply properties to hosts: connect, compile Propellor on remote side, apply all its properties
  - what about hosts which can't compile Propellor, or compile it very slowly?

Project status

- Stable core API, deployments unlikely to break

Project status

- Stable core API, deployments unlikely to break
- Runs well on SBCL on Debian
- Should be portable to other Unix and Common Lisp implementations

Project status

- Stable core API, deployments unlikely to break
- Runs well on SBCL on Debian
- Should be portable to other Unix and Common Lisp implementations
  - Mostly a matter of ifdef'ing out Linux- and SBCL-specific features.

Basic architecture

property	some configuration a host can have or lack
host	list of host attributes and list of properties
connection	means of applying properties to host, e.g. `:ssh`
deployment	host + connections
prerequisite data	secrets, and other files that properties need

Basic architecture

property	some configuration a host can have or lack
host	list of host attributes and list of properties
connection	means of applying properties to host, e.g. `:ssh`
deployment	host + connections
prerequisite data	secrets, and other files that properties need

- Consfigurator is just a Lisp library, no executables

Basic architecture

property	some configuration a host can have or lack
host	list of host attributes and list of properties
connection	means of applying properties to host, e.g. `:ssh`
deployment	host + connections
prerequisite data	secrets, and other files that properties need

- Consfigurator is just a Lisp library, no executables
- User's configuration of their hosts is just another Lisp library, your "consfig"

Basic architecture

property	some configuration a host can have or lack
host	list of host attributes and list of properties
connection	means of applying properties to host, e.g. `:ssh`
deployment	host + connections
prerequisite data	secrets, and other files that properties need

- Consfigurator is just a Lisp library, no executables
- User's configuration of their hosts is just another Lisp library, your "consfig"
- Typical usage
  - Loading Consfigurator and consfig into root Lisp defines hosts, properties and means of obtaining prerequisite data

Basic architecture

property	some configuration a host can have or lack
host	list of host attributes and list of properties
connection	means of applying properties to host, e.g. `:ssh`
deployment	host + connections
prerequisite data	secrets, and other files that properties need

- Consfigurator is just a Lisp library, no executables
- User's configuration of their hosts is just another Lisp library, your "consfig"
- Typical usage
  - Loading Consfigurator and consfig into root Lisp defines hosts, properties and means of obtaining prerequisite data
  - User then constructs and executes deployments at the root Lisp's REPL.

Properties

`:POSIX` properties perform only serial I/O, writing files and running shell commands

`:LISP` properties execute arbitrary Lisp code on the remote side

Properties

`:POSIX properties` perform only serial I/O, writing files and running shell commands

`:LISP properties` execute arbitrary Lisp code on the remote side

- `:SSH`, `:SUDO` can apply only `:POSIX` properties
  - Use properties to configure tiny hosts, shell accounts etc.

Properties

`:POSIX` properties perform only serial I/O, writing files and running shell commands

`:LISP` properties execute arbitrary Lisp code on the remote side

- `:SSH`, `:SUDO` can apply only `:POSIX` properties
  - Use properties to configure tiny hosts, shell accounts etc.
  - Lower startup overhead, so useful for quick tests.

Properties

`:POSIX` properties perform only serial I/O, writing files and running shell commands

`:LISP` properties execute arbitrary Lisp code on the remote side

- `:SSH`, `:SUDO` can apply only `:POSIX` properties
  - Use properties to configure tiny hosts, shell accounts etc.
  - Lower startup overhead, so useful for quick tests.
- `:SBCL` can apply both `:POSIX` and `:LISP` properties

Properties

`:POSIX` properties perform only serial I/O, writing files and running shell commands

`:LISP` properties execute arbitrary Lisp code on the remote side

- `:SSH`, `:SUDO` can apply only `:POSIX` properties
  - Use properties to configure tiny hosts, shell accounts etc.
  - Lower startup overhead, so useful for quick tests.
- `:SBCL` can apply both `:POSIX` and `:LISP` properties
  - Can be significantly faster as avoids roundtrips.

Properties

`:POSIX` properties perform only serial I/O, writing files and running shell commands

`:LISP` properties execute arbitrary Lisp code on the remote side

- `:SSH`, `:SUDO` can apply only `:POSIX` properties
  - Use properties to configure tiny hosts, shell accounts etc.
  - Lower startup overhead, so useful for quick tests.
- `:SBCL` can apply both `:POSIX` and `:LISP` properties
  - Can be significantly faster as avoids roundtrips.
  - Can do things that only a remote process can do.

Properties

`:POSIX` properties perform only serial I/O, writing files and running shell commands

`:LISP` properties execute arbitrary Lisp code on the remote side

- `:SSH`, `:SUDO` can apply only `:POSIX` properties
  - Use properties to configure tiny hosts, shell accounts etc.
  - Lower startup overhead, so useful for quick tests.
- `:SBCL` can apply both `:POSIX` and `:LISP` properties
  - Can be significantly faster as avoids roundtrips.
  - Can do things that only a remote process can do.
- Basic properties written explicitly, but most properties are compositions of other existing properties

Properties

`:POSIX properties` perform only serial I/O, writing files and running shell commands

`:LISP properties` execute arbitrary Lisp code on the remote side

- `:SSH`, `:SUDO` can apply only `:POSIX` properties
  - Use properties to configure tiny hosts, shell accounts etc.
  - Lower startup overhead, so useful for quick tests.
- `:SBCL` can apply both `:POSIX` and `:LISP` properties
  - Can be significantly faster as avoids roundtrips.
  - Can do things that only a remote process can do.
- Basic properties written explicitly, but most properties are compositions of other existing properties
  - Reduces how much Lisp you need to learn to get going.

Connections

- Arbitrary chaining, up to what makes sense.
- `(:ssh :sbcl (:lxc :name "my-lxc-hostname"))`

Connections

- Arbitrary chaining, up to what makes sense.
- `(:ssh :sbcl (:lxc :name "my-lxc-hostname"))`
  - In this case we start up a remote Lisp over SSH, and that process uses setns(2) to get into the LXC.

Connections

- Arbitrary chaining, up to what makes sense.
- `(:ssh :sbcl (:lxc :name "my-lxc-hostname"))`
  - In this case we start up a remote Lisp over SSH, and that process uses setns(2) to get into the LXC.
- `(:ssh (:lxc :name "my-lxc-hostname") :sbcl)`
  - Will use ssh and nsenter(1) to get a shell in the LXC, and then start up Lisp inside it.

Connections

- Arbitrary chaining, up to what makes sense.
- `(:ssh :sbcl (:lxc :name "my-lxc-hostname"))`
  - In this case we start up a remote Lisp over SSH, and that process uses setns(2) to get into the LXC.
- `(:ssh (:lxc :name "my-lxc-hostname") :sbcl)`
  - Will use ssh and nsenter(1) to get a shell in the LXC, and then start up Lisp inside it.
  - Much slower if you have more than one LXC!

Connections

- Arbitrary chaining, up to what makes sense.
- `(:ssh :sbcl (:lxc :name "my-lxc-hostname"))`
  - In this case we start up a remote Lisp over SSH, and that process uses setns(2) to get into the LXC.
- `(:ssh (:lxc :name "my-lxc-hostname") :sbcl)`
  - Will use ssh and nsenter(1) to get a shell in the LXC, and then start up Lisp inside it.
  - Much slower if you have more than one LXC!
- `((:ssh :user "root") :sbcl (:setuid :user "ntp"))`

Connections

- Arbitrary chaining, up to what makes sense.
- `(:ssh :sbcl (:lxc :name "my-lxc-hostname"))`
  - In this case we start up a remote Lisp over SSH, and that process uses setns(2) to get into the LXC.
- `(:ssh (:lxc :name "my-lxc-hostname") :sbcl)`
  - Will use ssh and nsenter(1) to get a shell in the LXC, and then start up Lisp inside it.
  - Much slower if you have more than one LXC!
- `((:ssh :user "root") :sbcl (:setuid :user "ntp"))`
  - Useful if you need to do things as several users.

Connections

- Arbitrary chaining, up to what makes sense.
- `(:ssh :sbcl (:lxc :name "my-lxc-hostname"))`
  - In this case we start up a remote Lisp over SSH, and that process uses setns(2) to get into the LXC.
- `(:ssh (:lxc :name "my-lxc-hostname") :sbcl)`
  - Will use ssh and nsenter(1) to get a shell in the LXC, and then start up Lisp inside it.
  - Much slower if you have more than one LXC!
- `((:ssh :user "root") :sbcl (:setuid :user "ntp"))`
  - Useful if you need to do things as several users.
- Process inside LXC and process running as other user only have the secrets they're meant to have.

Taking full advantage of subdeployments

- The fact that deployments are themselves properties, and can thus be nested, is well suited to expressing many things traditionally done without the help of declarative configuration management

Taking full advantage of subdeployments

- The fact that deployments are themselves properties, and can thus be nested, is well suited to expressing many things traditionally done without the help of declarative configuration management
  - Building bootable disc images using the same host definitions

Taking full advantage of subdeployments

- The fact that deployments are themselves properties, and can thus be nested, is well suited to expressing many things traditionally done without the help of declarative configuration management
  - Building bootable disc images using the same host definitions
  - Doing OS installs when Consfigurator is running on a live system

Taking full advantage of subdeployments

- The fact that deployments are themselves properties, and can thus be nested, is well suited to expressing many things traditionally done without the help of declarative configuration management
  - Building bootable disc images using the same host definitions
  - Doing OS installs when Consfigurator is running on a live system
  - Dumping images to execute arbitrary Lisp and re-running them from cron.

Consfigurator and Debian

- Like Propellor, an attempt to incorporate contemporary ideas about programmable infrastructure and reproducible systems into Debian-style systems administration

Consfigurator and Debian

- Like Propellor, an attempt to incorporate contemporary ideas about programmable infrastructure and reproducible systems into Debian-style systems administration
- You can get a lot of benefits of NixOS/Guix without giving up on the maturity of the Debian archive, and the advantages of the way we do things

Consfigurator and Debian

- Like Propellor, an attempt to incorporate contemporary ideas about programmable infrastructure and reproducible systems into Debian-style systems administration
- You can get a lot of benefits of NixOS/Guix without giving up on the maturity of the Debian archive, and the advantages of the way we do things
- Consfigurator's flexibility in applying small numbers of properties rather than doing full builds, and switching between connection types, is a pretty nice incremental systems administration environment