Go at Canonical
Transitioning Juju to Go

Presentation by
Dave Cheney
david.cheney@canonical.com
juju.ubuntu.com
August 2012
About this talk

- About Me
- About Juju
- Why the transition to Go
- How we develop Juju in Go
- Testing
- Error handling
- Goroutine management
- Questions
About Me

• Contributor to Go since 2011
• Work with agl on the ssh package
• Work with minux.ma on all things ARM
• Run the arm5 builder
• Joined Canonical in May 2012 to work on Juju
About Juju

- Service orchestration platform
- Juju describes *services* and their *relationships*
- A framework for developers and system administrators to deploy *services* via *charms*
  - *Charms* encapsulate the logic required to build, maintain, upgrade and decommission a *service*
  - *Relationships* between *services* allow *charms* to programmatically configure themselves by invoking *hooks*
- Juju manages the provisioning of *services* on virtual (or physical) infrastructure
What are the components of Juju

- Centralised state repository
- A provisioning agent, which interfaces with the infrastructure provider to spin up new machines when required
- One machine agent per machine, which handles starting unit agents
- One unit agent per service instance, which is responsible for running the Charm hooks
- Command line tools
- Command line utilities
- Charms
Why the transition to Go

- Juju has been shipping since Ubuntu 11.10
- Juju is a supported part of Ubuntu 12.04LTS.
- Python code uses Twisted, Python generators and callbacks heavily
- Quite hard to get right, hard to reason about
- Heavily asynchronous, hard to know when an action will occur, hard to know if an action has occurred
Why the transition to Go pt. 2

- The watcher pattern is a natural fit for channels
- Synchronous coding as an alternative to callbacks
- Static typing reduces the amount of test logic required for verification
- Go binaries have a lower resource footprint
- Go is supported on ARM
- Canonical is interested in Go, Juju is a the first of many projects
How we develop Juju in Go

- Team of seven, including a manager
- Very geographically dispersed
- Use Launchpad for project management
- IRC and mailing lists for communication
- Weekly meeting held on G+ hangout
- Occasional week long sprints
How we develop Juju in Go, pt 2

- Use Rietveld for code review via lbox
- Custom bzr wrapper, cobzr, for branch management
- Additional packages written by the team, goamz, gnuflag, goyaml, gozk
- Variety of editors; Vim, acme, Sublime text all represented
- Various $GOPATH strategies
Testing

- Use gocheck heavily
- Lots of table driven tests
- Embedding allows us to compose test suites with complex setup and teardown phases
- jujutest package runs the same integration tests against all our providers
- Test in _test packages so we don't cheat with private symbols
package state_test

import (  
    "launchpad.net/gocheck"
    "launchpad.net/juju-core/state"
    "launchpad.net/juju-core/version"
)

type MachineSuite struct {  
    ConnSuite  
    machine *state.Machine
}

var _ = Suite(&MachineSuite{})

func (s *MachineSuite) SetUpTest(c *C) {  
    s.ConnSuite.SetUpTest(c)  
    var err error  
    s.machine, err = s.State.AddMachine()  
    c.Assert(err, IsNil)  
}
id, err := m.InstanceId()
c.Assert(err, IsNil)
c.Assert(id, Equals, 1)

ch, ok := <-w.Changes()
c.Assert(ok, Equals, true)
c.Assert(ch.Changed, HasLen, 0)
c.Assert(ch.Departed, HasLen, 0)

actual := make(map[string]interface{})
c.Assert(err, IsNil)
c.Assert(actual, DeepEquals, expected)
FAIL: cmd_test.go:307: cmdSuite.TestUnexposeCommandInit

cmd_test.go:310:
    c.Assert(err, ErrorMatches, "no service specified")
... error string = "no service name specified"
... regex string = "no service specified"
Error handling

• We check errors a lot as most operations can fail
• Constantly considering the error path, and how to leave the state in a manner that actions can be retried later
• Moving to MongoDB will allow us to batch our requests and consolidate failure points
Goroutine management

- Use the tomb package
- Tombs manage a goroutine's lifecycle
- Tombs let us wait for a goroutine to exit, and capture any error if this exit was unexpected
func (w *ChildrenWatcher) loop() {
    defer w.tomb.Done()
    defer close(w.changeChan)

    watch, err := w.update(zookeeper.EVENT_CHILD)
    if err != nil {
        w.tomb.Kill(err)
        return
    }

    for {
        select {
            case <-w.tomb.Dying():
                return
            case evt := <-watch:
                if !evt.Ok() {
                    w.tomb.Killf("watcher: session event: %v", evt)
                    return
                }
                watch, err = w.update(evt.Type)
                if err != nil {
                    w.tomb.Kill(err)
                    return
                }
        }
    }
}
Thank you. Questions?
http://launchpad.net/juju-core

Dave Cheney
david.cheney@canonical.com
@davecheney
One more thing