HashiCorp

Nomad

Adventures in Writing Distributed Systems in Go:

The Good, the Bad, and the Ugly Code in HashiCorp Nomad

Michael Schurter @schmichael

Nomad Team Lead at HashiCorp

What is a HashiCorp?

https://www.hashicorp.com

HashiCorp Products

Terraform

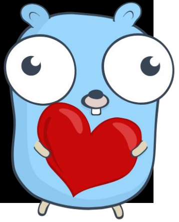
Vault

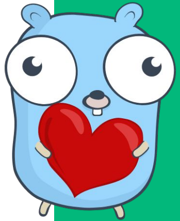
Consul

Nomad

Vagrant

Packer

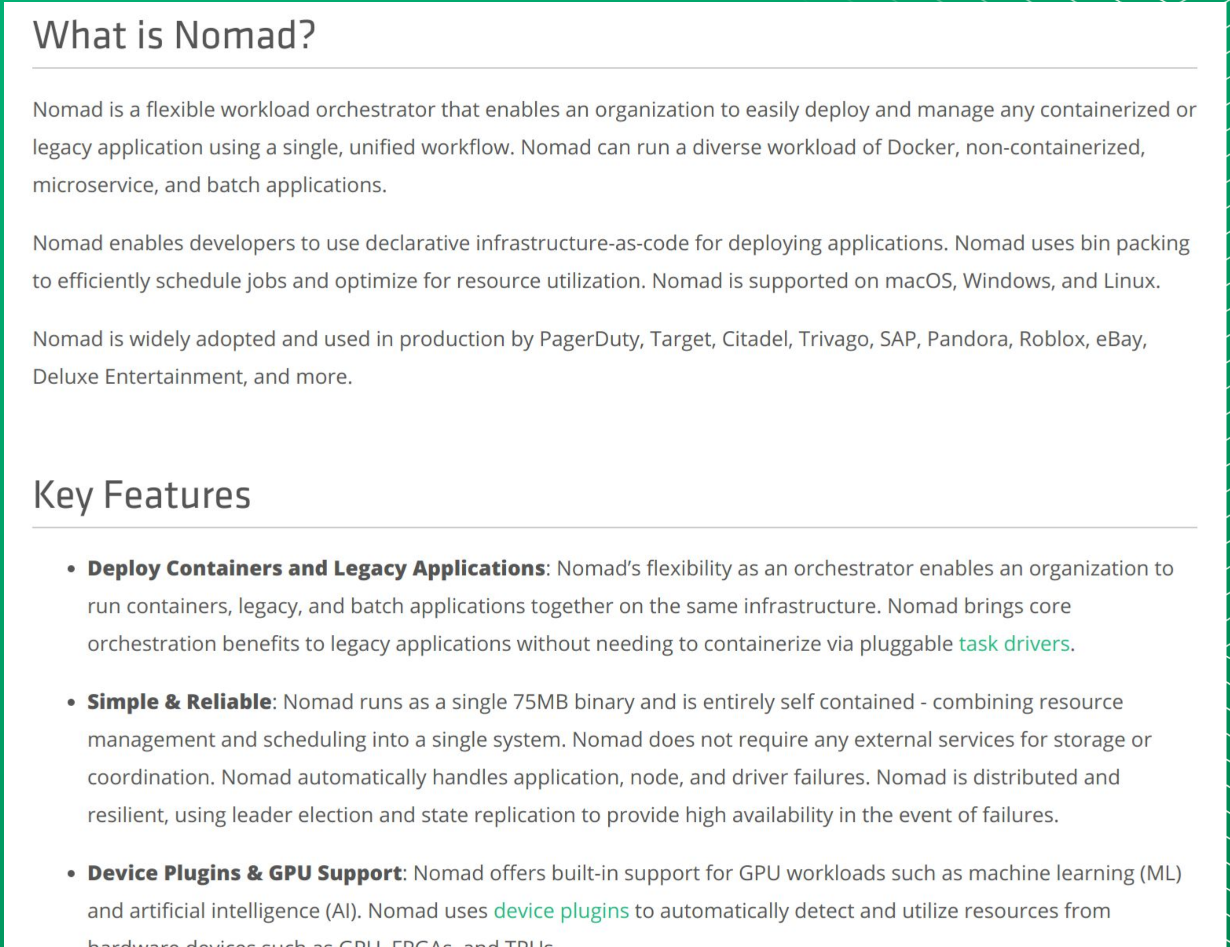




Multi-Cloud DevOps Infrastructure Stuff

Terraform	Provision
Nomad	Run
Consul	Connect
Vault	Secure

What is Nomad



What is Nomad?

Nomad is a flexible workload orchestrator that enables an organization to easily deploy and manage any containerized or legacy application using a single, unified workflow. Nomad can run a diverse workload of Docker, non-containerized, microservice, and batch applications.

Nomad enables developers to use declarative infrastructure-as-code for deploying applications. Nomad uses bin packing to efficiently schedule jobs and optimize for resource utilization. Nomad is supported on macOS, Windows, and Linux.

Nomad is widely adopted and used in production by PagerDuty, Target, Citadel, Trivago, SAP, Pandora, Roblox, eBay, Deluxe Entertainment, and more.

Key Features

- **Deploy Containers and Legacy Applications**: Nomad's flexibility as an orchestrator enables an organization to run containers, legacy, and batch applications together on the same infrastructure. Nomad brings core orchestration benefits to legacy applications without needing to containerize via pluggable task drivers.
- **Simple & Reliable**: Nomad runs as a single 75MB binary and is entirely self contained - combining resource management and scheduling into a single system. Nomad does not require any external services for storage or coordination. Nomad automatically handles application, node, and driver failures. Nomad is distributed and resilient, using leader election and state replication to provide high availability in the event of failures.
- **Device Plugins & GPU Support**: Nomad offers built-in support for GPU workloads such as machine learning (ML) and artificial intelligence (AI). Nomad uses device plugins to automatically detect and utilize resources from hardware devices such as GPU, FPGAs, and TPUs.

Nomad by the Numbers

4+ years

Started May 31, 2015

v0.1 September 28, 2015

244k+ LoC

1.1M+ counting vendoring

Go code

2.5k+ tests

Unit, integration, e2e, xplat

Excluding table/sub tests

510 deps

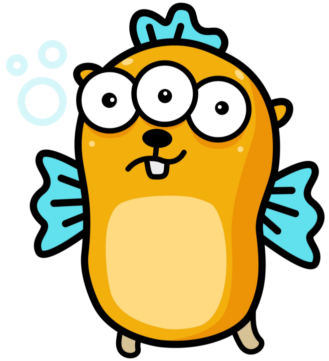
Still vendoring

Modules soon I promise.

Legend

Good Gopher

Bad Gopher


Ugly Gopher

Single Binary; Multiple Platforms

https://releases.hashicorp.com/nomad/0.9.5/

Not all platforms

Alpine
ARM
CGO

go get github.com/hashicorp/nomad

This implies go `build`, `install`, etc work as well.

go test ./...

E2E tests enabled via env var; all others run by default.

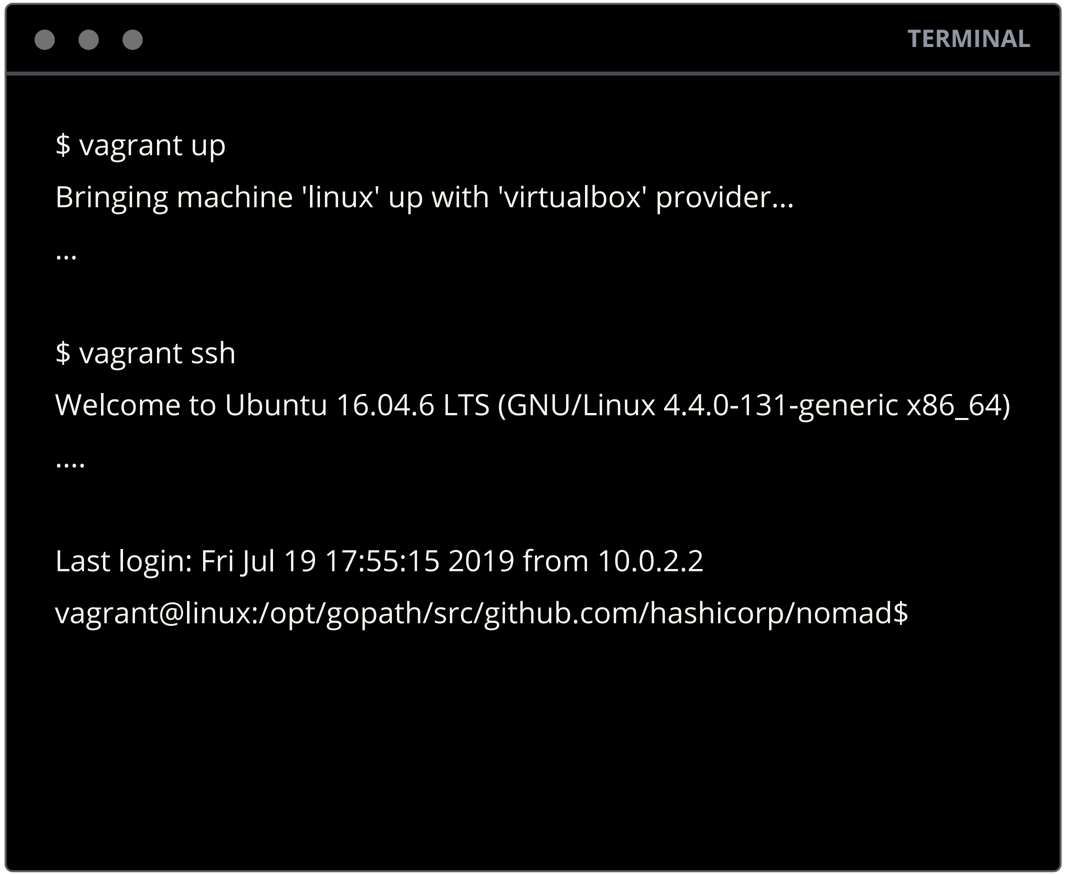
go test ./...

Slow.
Eats all your CPU.
Many tests require root which means we may destroy your computer.

Use Vagrant for Client Tests

Vagrant

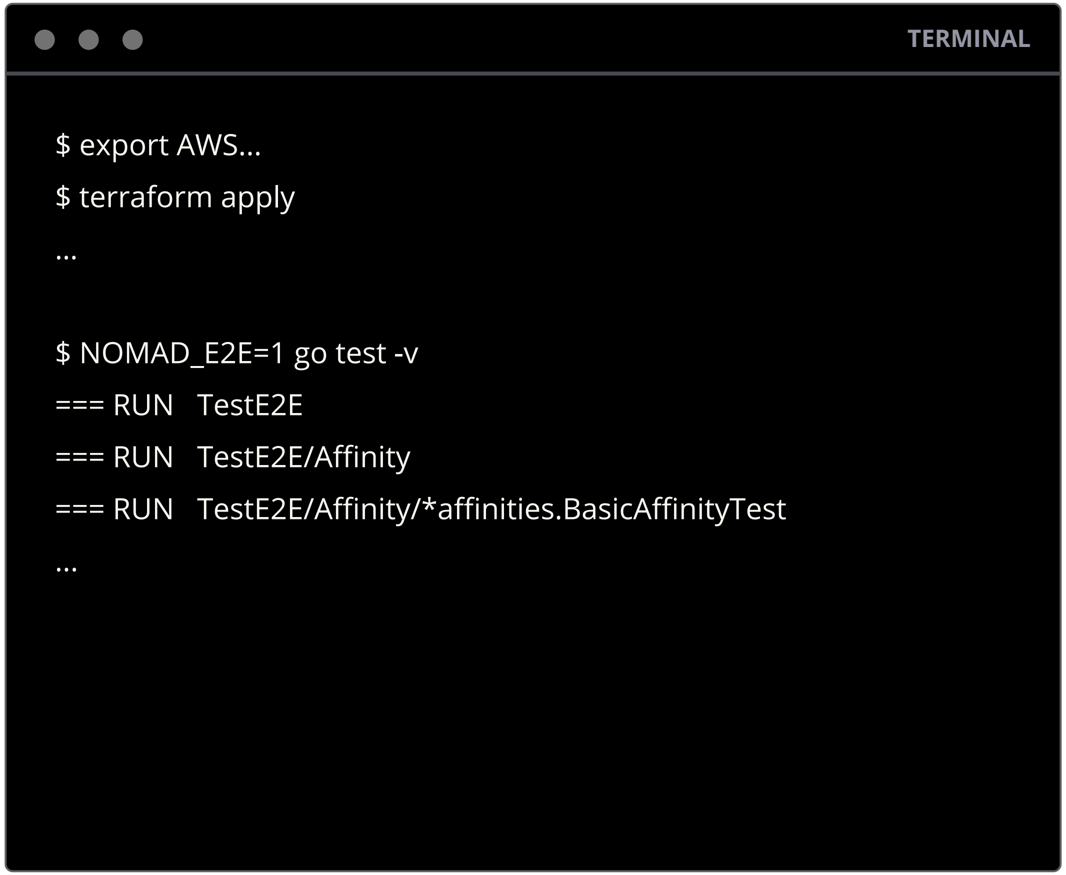
Root Vagrant box suitable for dangerous tests.


```
$ vagrant up
Bringing machine 'linux' up with 'virtualbox' provider...
...

$ vagrant ssh
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-131-generic x86_64)
....

Last login: Fri Jul 19 17:55:15 2019 from 10.0.2.2
vagrant@linux:/opt/gopath/src/github.com/hashicorp/nomad$
```

Terraform

Terraform for E2E clusters.

Many work locally too!

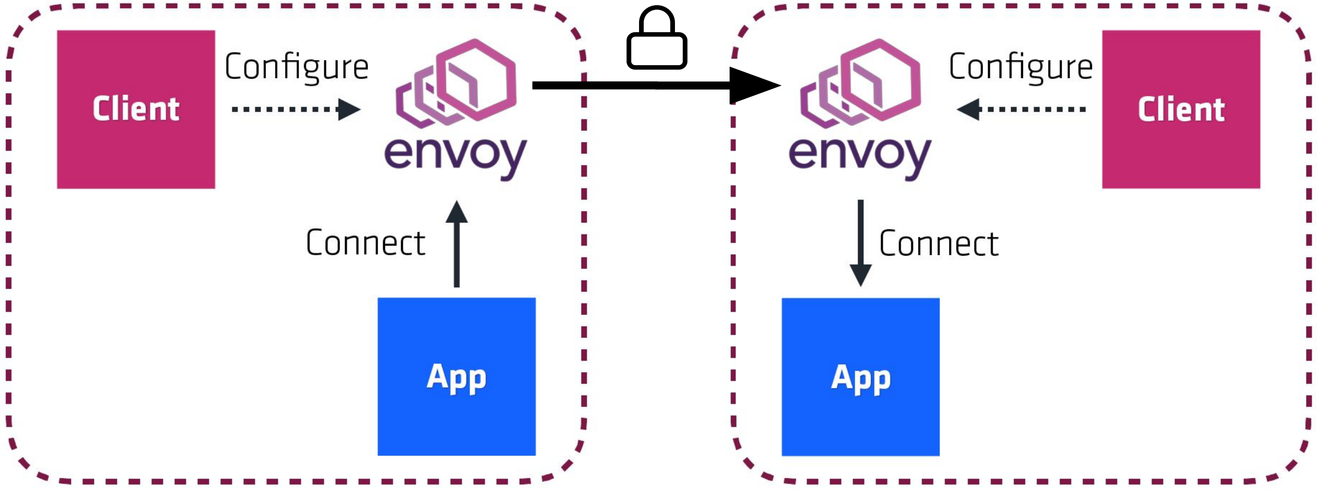


```
$ export AWS...
$ terraform apply
...

$ NOMAD_E2E=1 go test -v
=== RUN   TestE2E
=== RUN   TestE2E/Affinity
=== RUN   TestE2E/Affinity/*affinities.BasicAffinityTest
...
```

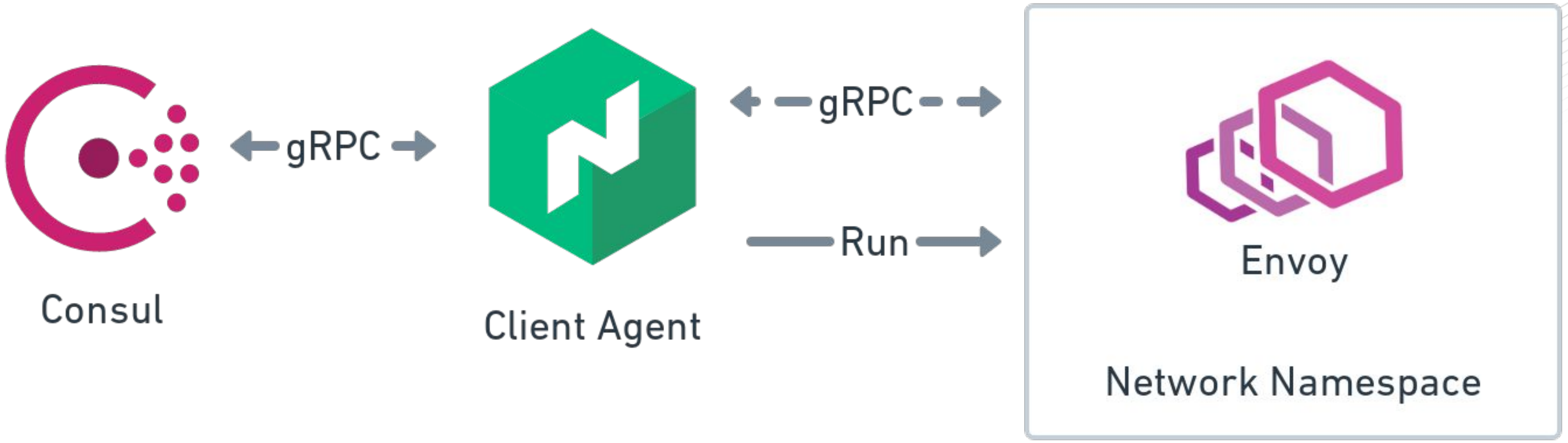
Quick and easy network proxying

Takes some bits from over here and put them over there.

Network Namespaces in 0.10



Proxy out of Network Namespace



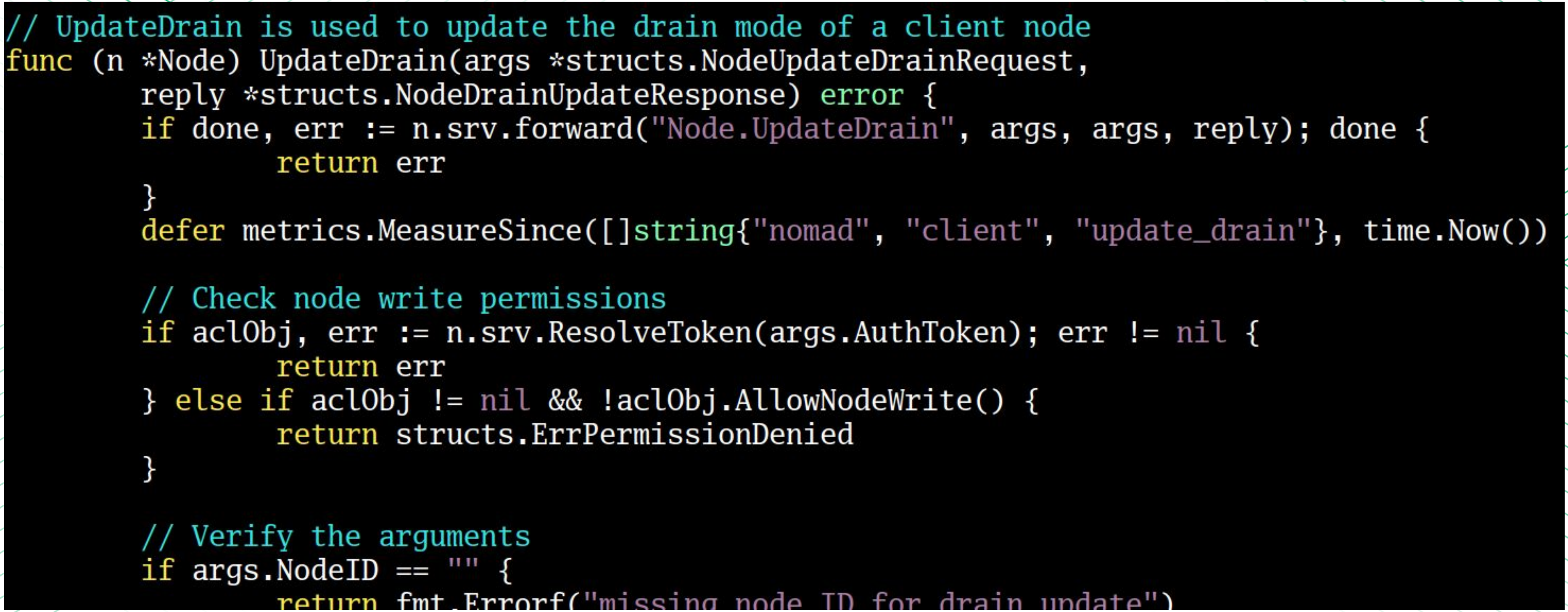
Quick and easy network proxying

~100 lines of code

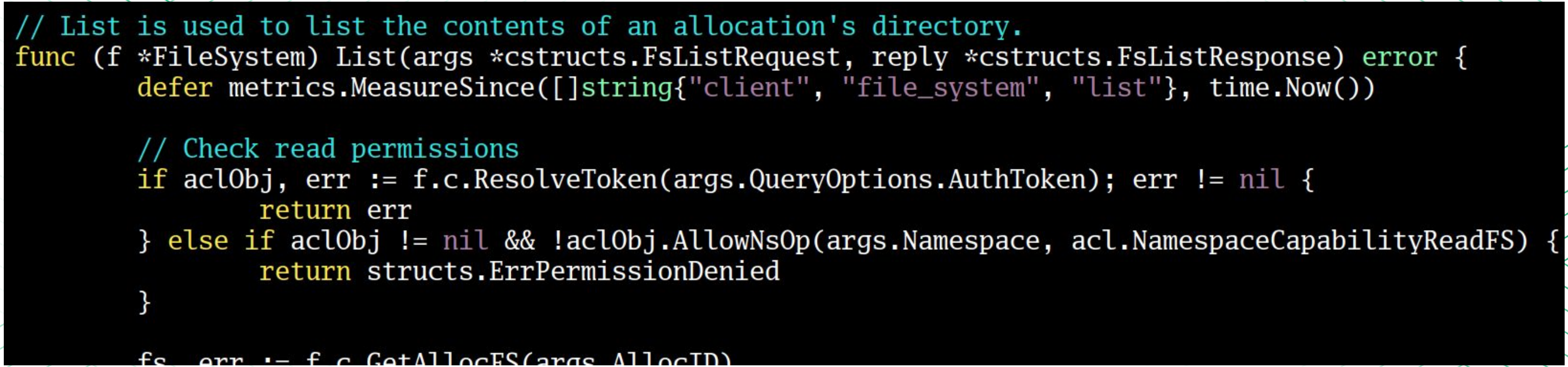
https://github.com/hashicorp/nomad/blob/v0.10.0-beta1/client/allocrunner/consulsock_hook.go#L202-L319

Copy paste copy paste copy paste

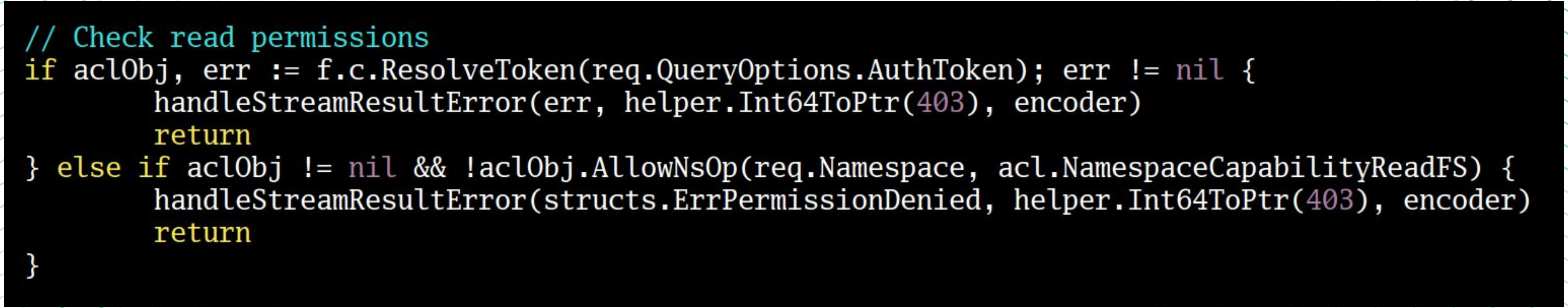
Normally I don’t miss metaprogramming but ACLs...



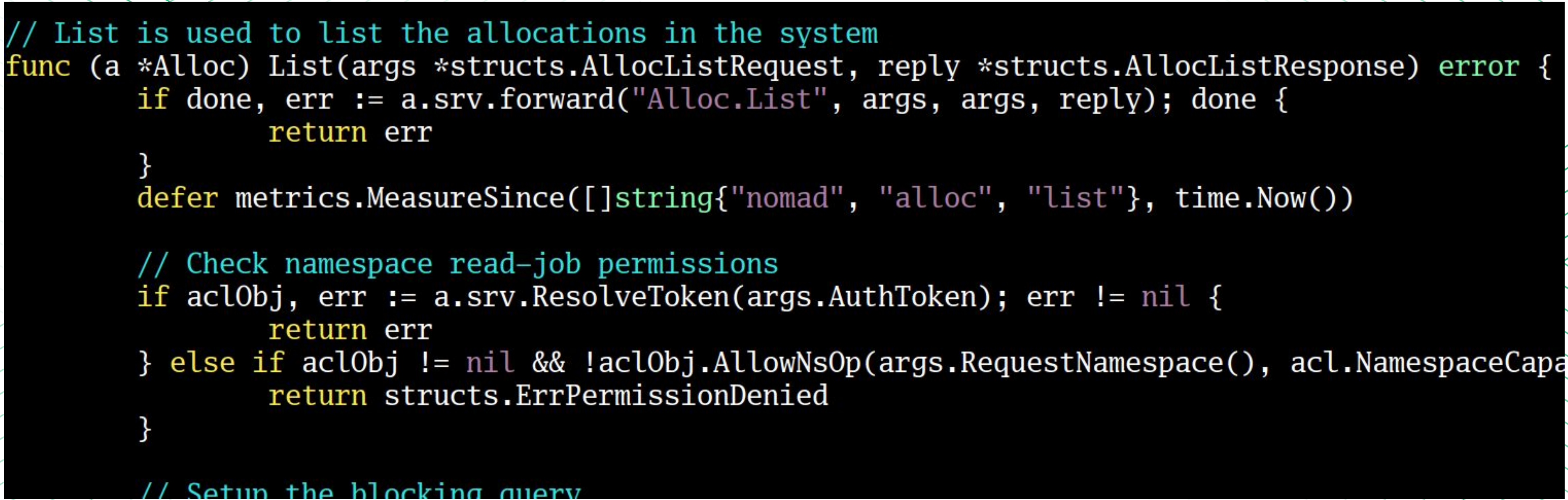
```go
// UpdateDrain is used to update the drain mode of a client node
func (n *Node) UpdateDrain(args *structs.NodeUpdateDrainRequest,
        reply *structs.NodeDrainUpdateResponse) error {
        if done, err := n.srv.forward("Node.UpdateDrain", args, args, reply); done {
                return err
        }
        defer metrics.MeasureSince([]string{"nomad", "client", "update_drain"}, time.Now())

        // Check node write permissions
        if aclObj, err := n.srv.ResolveToken(args.AuthToken); err != nil {
                return err
        } else if aclObj != nil && !aclObj.AllowNodeWrite() {
                return structs.ErrPermissionDenied
        }

        // Verify the arguments
        if args.NodeID == "" {
                return fmt.Errorf("missing node ID for drain update")
```



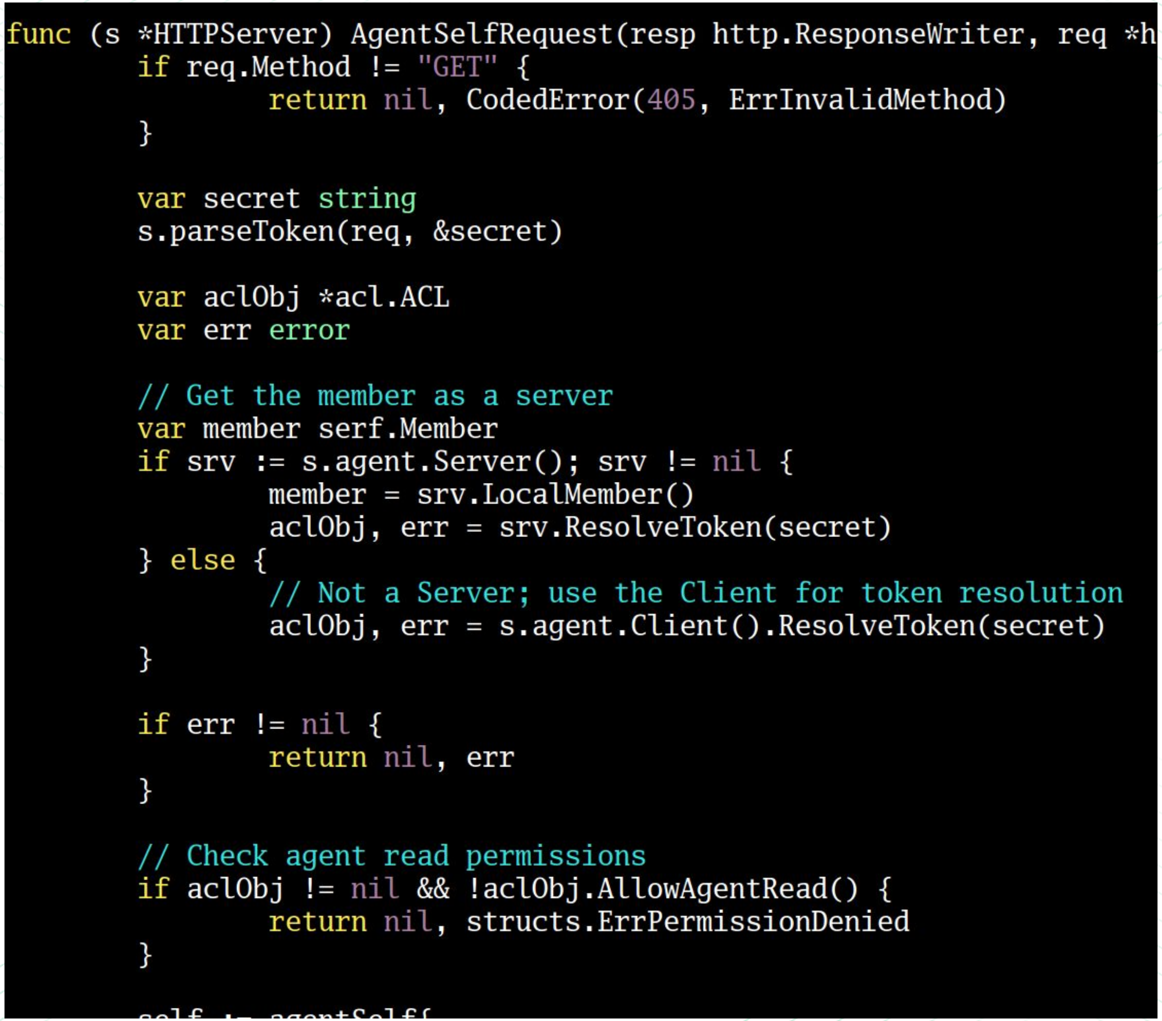
```go
// List is used to list the contents of an allocation's directory.
func (f *FileSystem) List(args *cstructs.FsListRequest, reply *cstructs.FsListResponse) error {
	defer metrics.MeasureSince([]string{"client", "file_system", "list"}, time.Now())

	// Check read permissions
	if aclObj, err := f.c.ResolveToken(args.QueryOptions.AuthToken); err != nil {
		return err
	} else if aclObj != nil && !aclObj.AllowNsOp(args.Namespace, acl.NamespaceCapabilityReadFS) {
		return structs.ErrPermissionDenied
	}

	fs, err := f.c.GetAllocFS(args.AllocID)
```



```go
// Check read permissions
if aclObj, err := f.c.ResolveToken(req.QueryOptions.AuthToken); err != nil {
	handleStreamResultError(err, helper.Int64ToPtr(403), encoder)
	return
} else if aclObj != nil && !aclObj.AllowNsOp(req.Namespace, acl.NamespaceCapabilityReadFS) {
	handleStreamResultError(structs.ErrPermissionDenied, helper.Int64ToPtr(403), encoder)
	return
}
```



```
// List is used to list the allocations in the system
func (a *Alloc) List(args *structs.AllocListRequest, reply *structs.AllocListResponse) error {
	if done, err := a.srv.forward("Alloc.List", args, args, reply); done {
		return err
	}
	defer metrics.MeasureSince([]string{"nomad", "alloc", "list"}, time.Now())

	// Check namespace read-job permissions
	if aclObj, err := a.srv.ResolveToken(args.AuthToken); err != nil {
		return err
	} else if aclObj != nil && !aclObj.AllowNsOp(args.RequestNamespace(), acl.NamespaceCapa
		return structs.ErrPermissionDenied
	}

	// Setup the blocking query
```

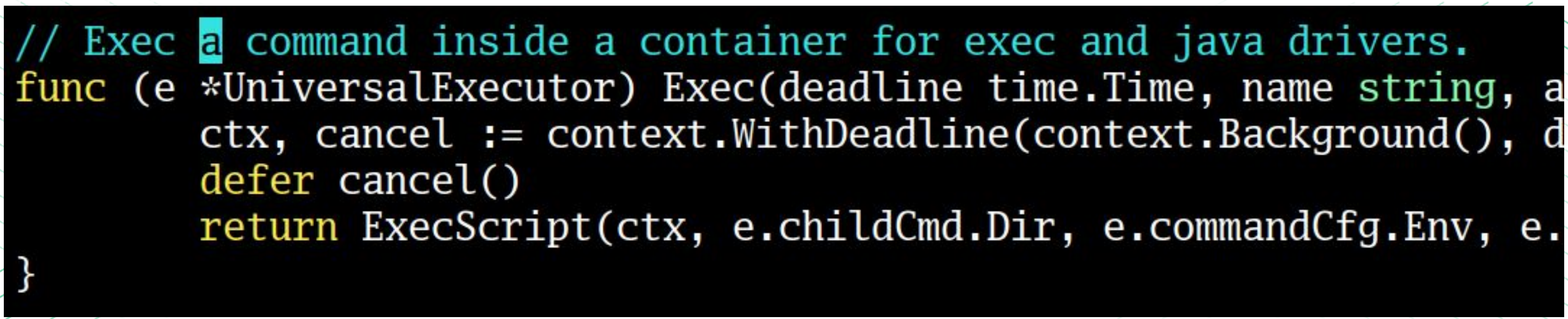


```go
func (s *HTTPServer) AgentSelfRequest(resp http.ResponseWriter, req *h
	if req.Method != "GET" {
		return nil, CodedError(405, ErrInvalidMethod)
	}

	var secret string
	s.parseToken(req, &secret)

	var aclObj *acl.ACL
	var err error

	// Get the member as a server
	var member serf.Member
	if srv := s.agent.Server(); srv != nil {
		member = srv.LocalMember()
		aclObj, err = srv.ResolveToken(secret)
	} else {
		// Not a Server; use the Client for token resolution
		aclObj, err = s.agent.Client().ResolveToken(secret)
	}

	if err != nil {
		return nil, err
	}

	// Check agent read permissions
	if aclObj != nil && !aclObj.AllowAgentRead() {
		return nil, structs.ErrPermissionDenied
	}
```

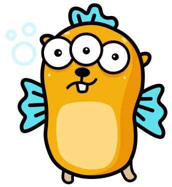
With great concurrency…
Comes great complexity.

https://github.com/hashicorp/nomad/pull/6082

Contexts are great...



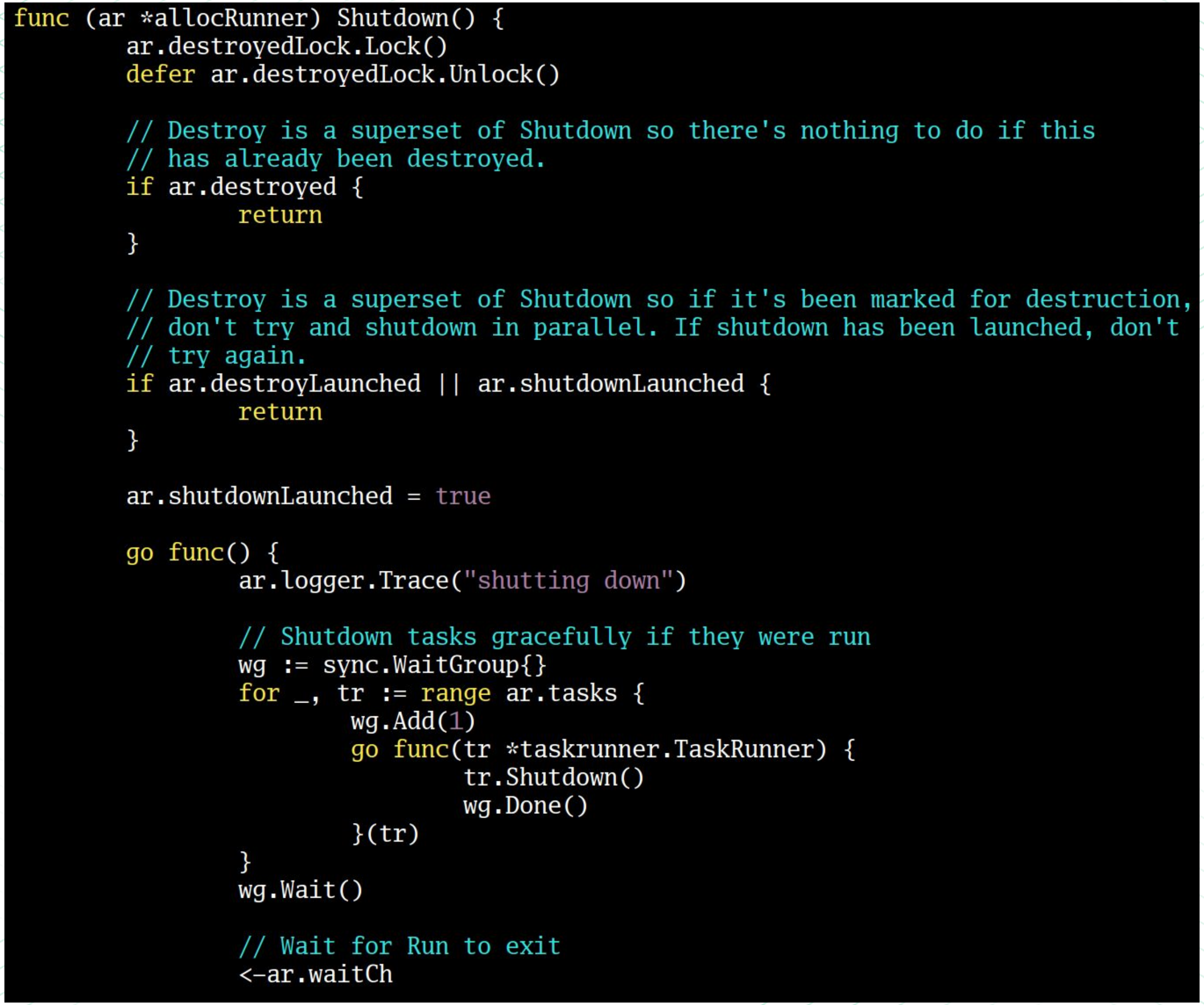
```
// Exec a command inside a container for exec and java drivers.
func (e *UniversalExecutor) Exec(deadline time.Time, name string, a
        ctx, cancel := context.WithDeadline(context.Background(), d
        defer cancel()
        return ExecScript(ctx, e.childCmd.Dir, e.commandCfg.Env, e.
}
```



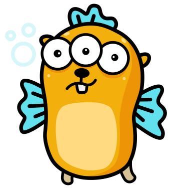
…until you need coordinated shutdowns.

- Don’t leak goroutines!
- Causes memory leaks.
- Causes race detector failures in tests:

```
panic: Log in goroutine after Test... has completed
```



```go
func (ar *allocRunner) Shutdown() {
	ar.destroyedLock.Lock()
	defer ar.destroyedLock.Unlock()

	// Destroy is a superset of Shutdown so there's nothing to do if this
	// has already been destroyed.
	if ar.destroyed {
		return
	}

	// Destroy is a superset of Shutdown so if it's been marked for destruction,
	// don't try and shutdown in parallel. If shutdown has been launched, don't
	// try again.
	if ar.destroyLaunched || ar.shutdownLaunched {
		return
	}

	ar.shutdownLaunched = true

	go func() {
		ar.logger.Trace("shutting down")

		// Shutdown tasks gracefully if they were run
		wg := sync.WaitGroup{}
		for _, tr := range ar.tasks {
			wg.Add(1)
			go func(tr *taskrunner.TaskRunner) {
				tr.Shutdown()
				wg.Done()
			}(tr)
		}
		wg.Wait()

		// Wait for Run to exit
		<-ar.waitCh
```

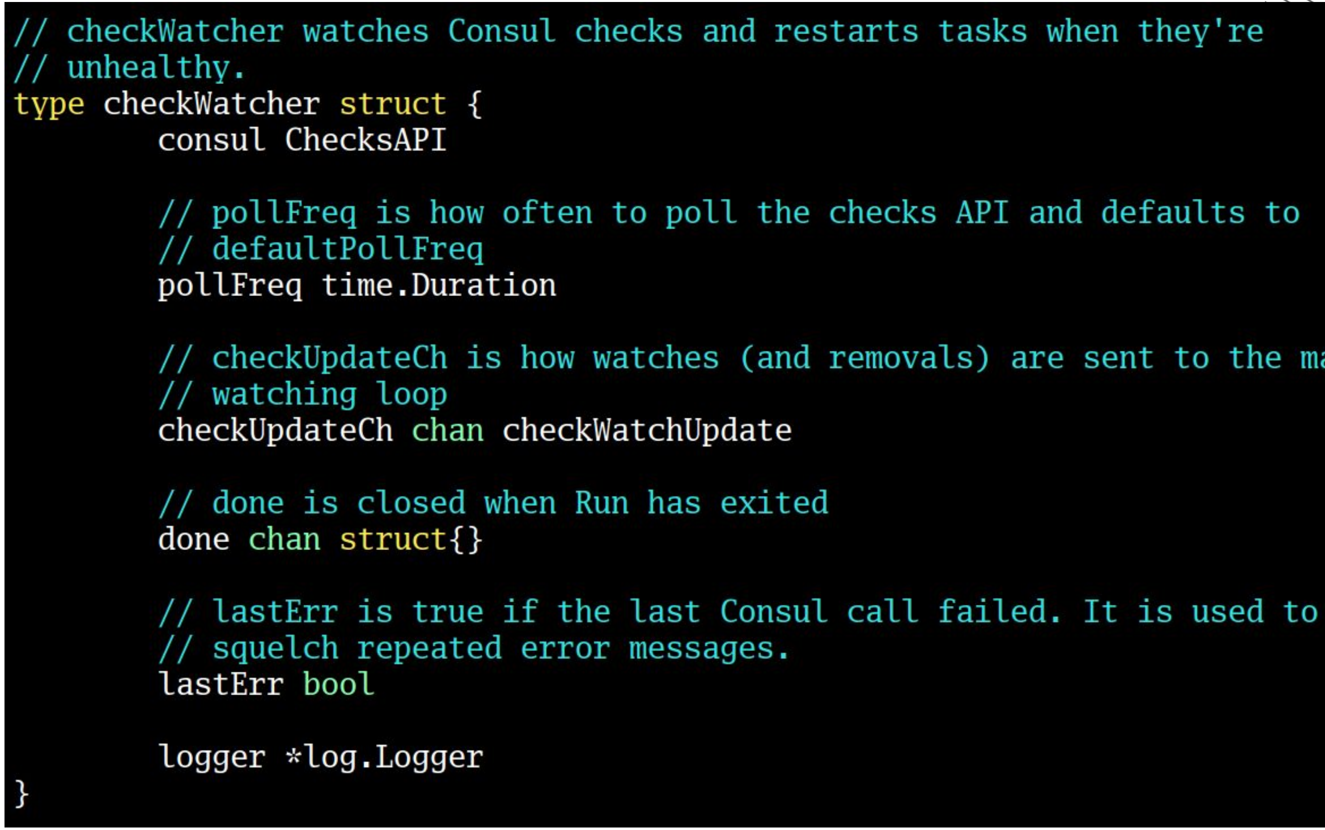


There are worse fates than locks…

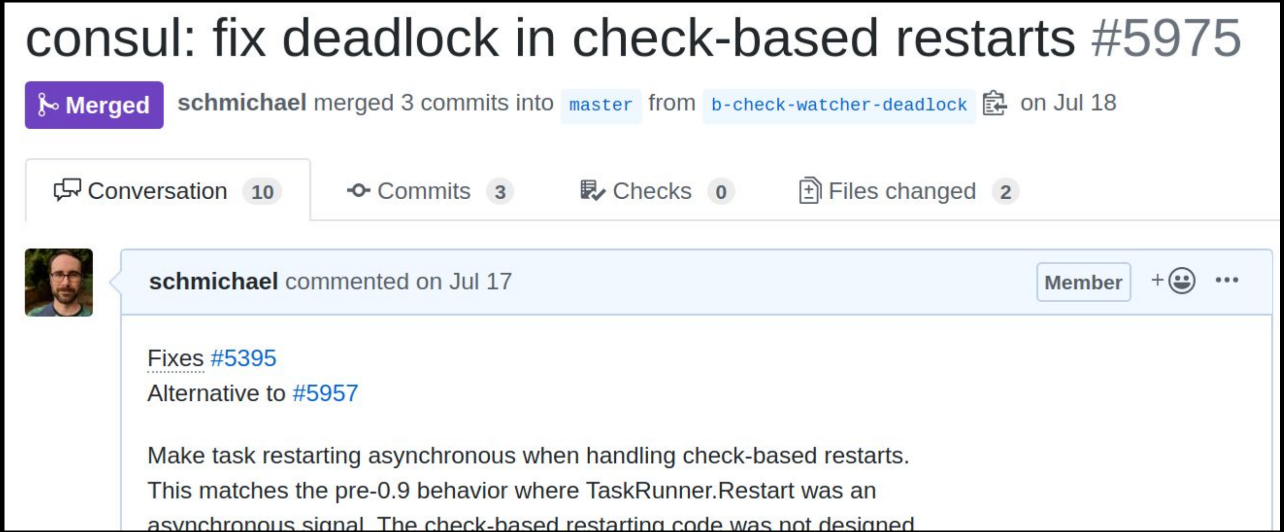
schmichael 2017

Locks are evil!

Communicate by passing messages!



```
// checkWatcher watches Consul checks and restarts tasks when they're
// unhealthy.
type checkWatcher struct {
	consul ChecksAPI

	// pollFreq is how often to poll the checks API and defaults to
	// defaultPollFreq
	pollFreq time.Duration

	// checkUpdateCh is how watches (and removals) are sent to the ma
	// watching loop
	checkUpdateCh chan checkWatchUpdate

	// done is closed when Run has exited
	done chan struct{}

	// lastErr is true if the last Consul call failed. It is used to
	// squelch repeated error messages.
	lastErr bool

	logger *log.Logger
}
```


consul: fix deadlock in check-based restarts #5975

Merged schmichael merged 3 commits into `master` from `b-check-watcher-deadlock` on Jul 18

schmichael commented on Jul 17

Fixes #5395
Alternative to #5957

Make task restarting asynchronous when handling check-based restarts.
This matches the pre-0.9 behavior where TaskRunner.Restart was an
asynchronous signal. The check-based restarting code was not designed

schmichael 2019

Thank You

hello@hashicorp.com
www.hashicorp.com