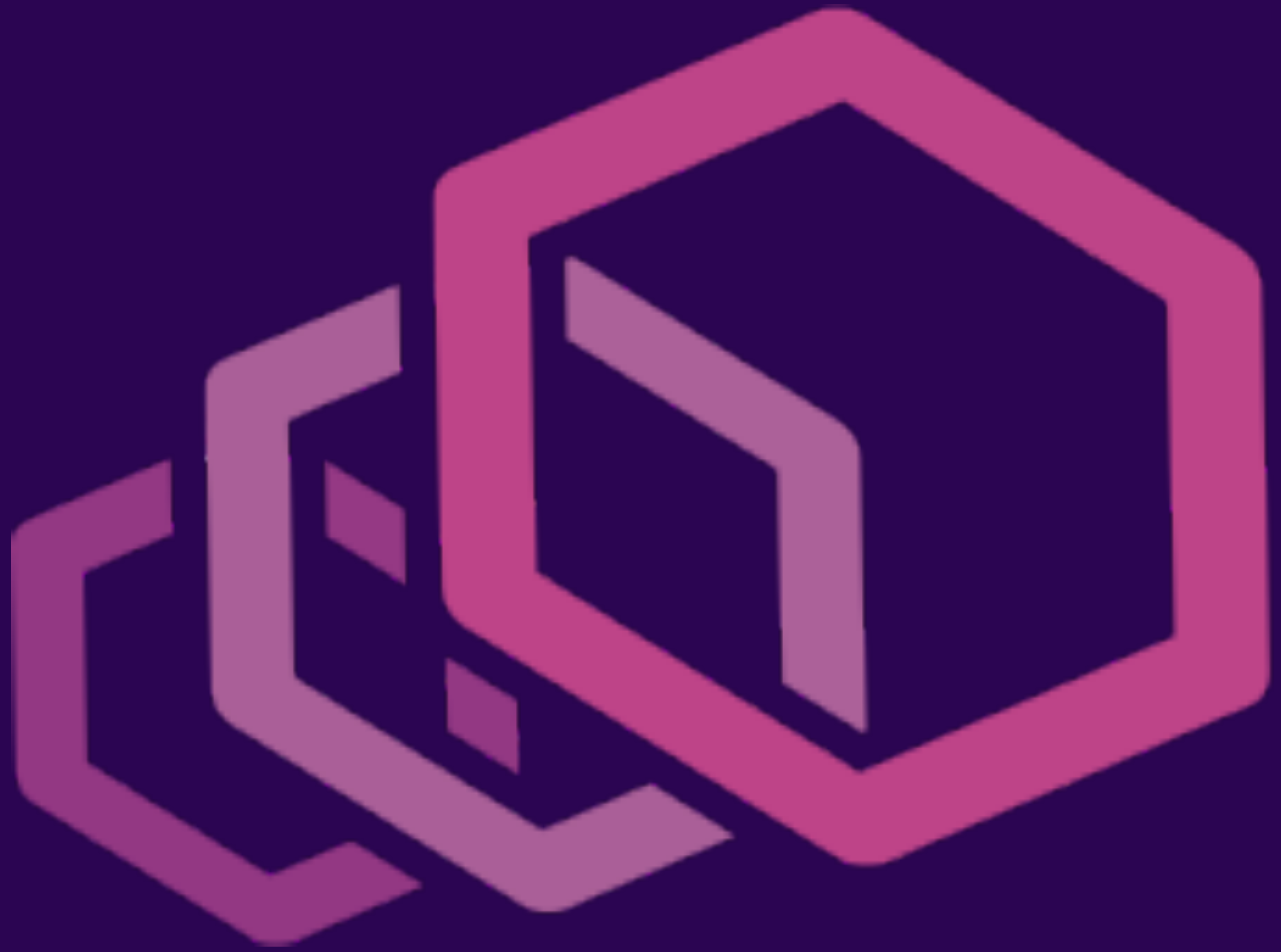
Lessons learnt building Kubernetes controllers

David Cheney - Heptio†

g’day



envoy

Contour

A Kubernetes Ingress Controller

There is
no cloud

it's just
someone else's computer

Connaissez-vous Kubernetes?

Kubernetes in one slide

- Replicated data store; etcd
- API server; auth, schema validation, CRUD operations plus watch
- Controllers and operators; watch the API server, try to make the world match the contents of the data store
- Container runtime; eg, docker, running containers on individual hosts enrolled with the API server

Ingress-*what* controller?

Ingress controllers provide load balancing and reverse proxying as a service

An ingress controller should take care of the 90% use case for deploying HTTP middleware

Traffic consolidation

heptio Contour

TLS management

TLS, SSL, Let’s Encrypt and all that good stuff

Abstract configuration

Describe your web application abstractly

Path based routing

heptio Contour

What is Contour?

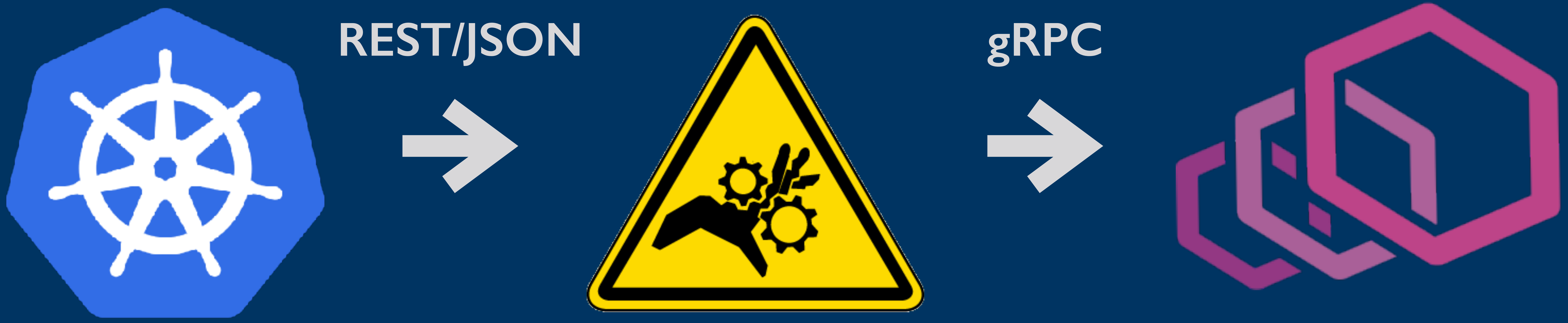
Why did Contour choose Envoy as its foundation?

heptio Contour

Envoy is a proxy *designed* for dynamic configuration

Contour is the API *server*
Envoy is the API *client*

Contour Architecture Diagram



heptio
Contour

Envoy handles configuration changes *without* reloading

Kubernetes and Envoy interoperability

	Ingress	Service	Secret	Endpoints
LDS	😀		😀	
RDS	😀			
CDS		😀		
EDS				😀

Kubernetes API objects

Envoy gRPC streams

Building software in a cloud native world

Let’s explore the developer experience building software for Kubernetes from the micro to the macro

As of the last release, Contour is around 20800 LOC

5000 source, 15800 tests

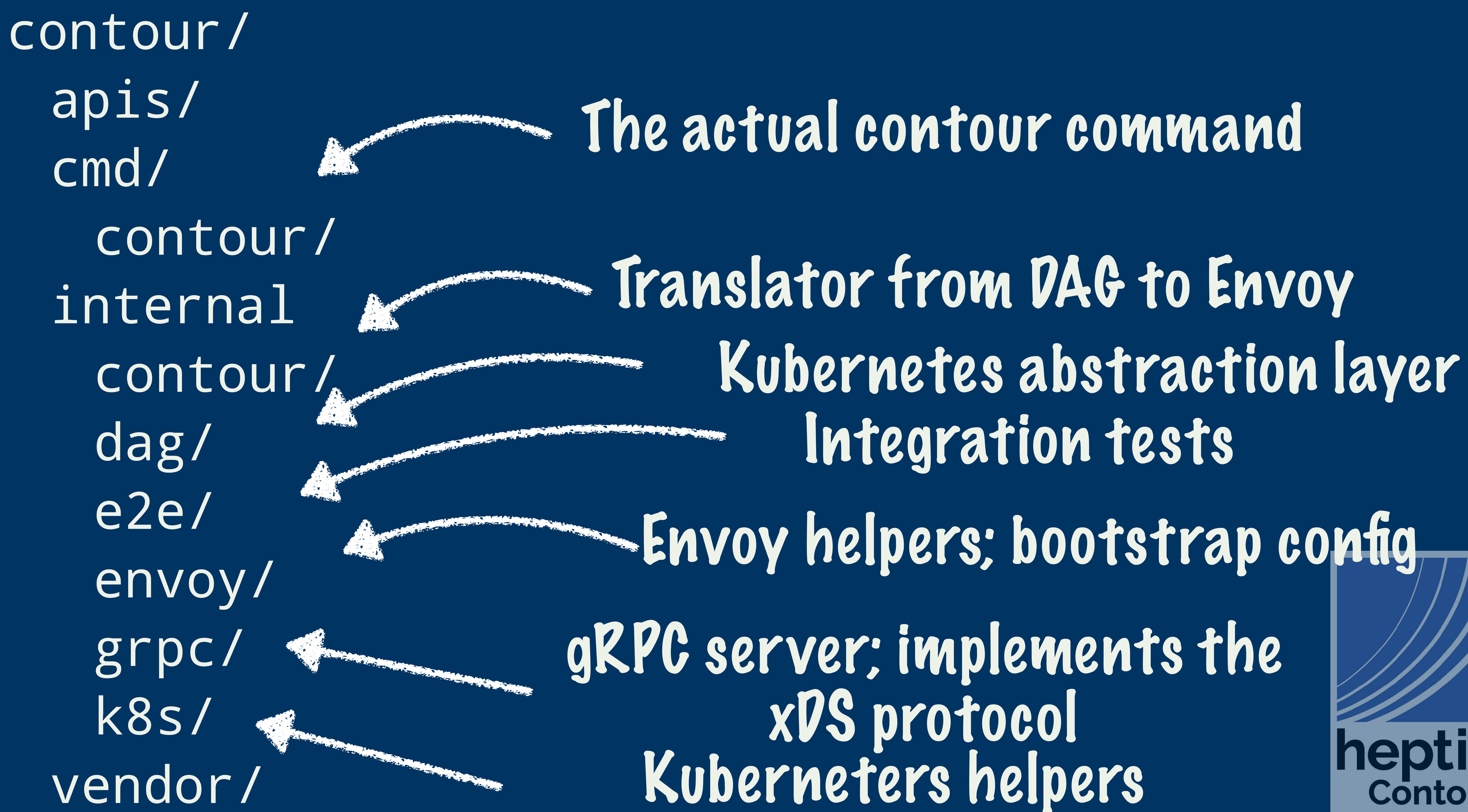
😁

Do as little as possible in `main.main`

`main.main` rule of thumb

- Parse flags
- Read configuration from disk / environment
- Set up connections; e.g. database connection, kubernetes API
- Set up loggers
- Call into your business logic and exit(3) success or fail

Ruthlessly refactor your `main` package to move as much code as possible to its own package



```
contour/
  apis/
  cmd/
    contour/
  internal
    contour/
    dag/
    e2e/
    envoy/
    grpc/
    k8s/
  vendor/
```

The actual contour command

Translator from DAG to Envoy

Kubernetes abstraction layer

Integration tests

Envoy helpers; bootstrap config

gRPC server; implements the xDS protocol

Kuberneters helpers

Name your packages for what they provide, not what they contain

heptio
Contour

Consider `internal/` for packages that you don’t want other projects to depend on

Managing concurrency

github.com/heptio/workgroup

Contour needs to watch for changes to Ingress, Services, Endpoints, and Secrets

Contour also needs to run a gRPC server for Envoy, and a HTTP server for the `/debug/pprof` endpoint

```
// A Group manages a set of goroutines with related lifetimes.
// The zero value for a Group is fully usable without initialisation.
type Group struct {
        fn []func(<-chan struct{}) error
}

// Add adds a function to the Group.
// The function will be exectuted in its own goroutine when
// Run is called. Add must be called before Run.
func (g *Group) Add(fn func(<-chan struct{}) error) {
        g.fn = append(g.fn, fn)
}

// Run executes each registered function in its own goroutine.
// Run blocks until all functions have returned.
// The first function to return will trigger the closure of the channel
// passed to each function, who should in turn, return.
// The return value from the first function to exit will be returned to
// the caller of Run.
func (g *Group) Run() error {
        // if there are no registered functions, return immediately.
```

Run each function in its own goroutine; when one exits shut down the rest

Register functions to be run as goroutines in the group

```
var g workgroup.Group

client := newClient(*kubeconfig, *incluster)

k8s.WatchServices(&g, client)
k8s.WatchEndpoints(&g, client)
k8s.WatchIngress(&g, client)
k8s.WatchSecrets(&g, client)

g.Add(debug.Start)

g.Add(func(stop <-chan struct{}) error {
        addr := net.JoinHostPort(*xdsAddr, strconv.Itoa(*xdsPort))
        l, err := net.Listen("tcp", addr)
        if err != nil {
                return err
        }
```

Make a new Group

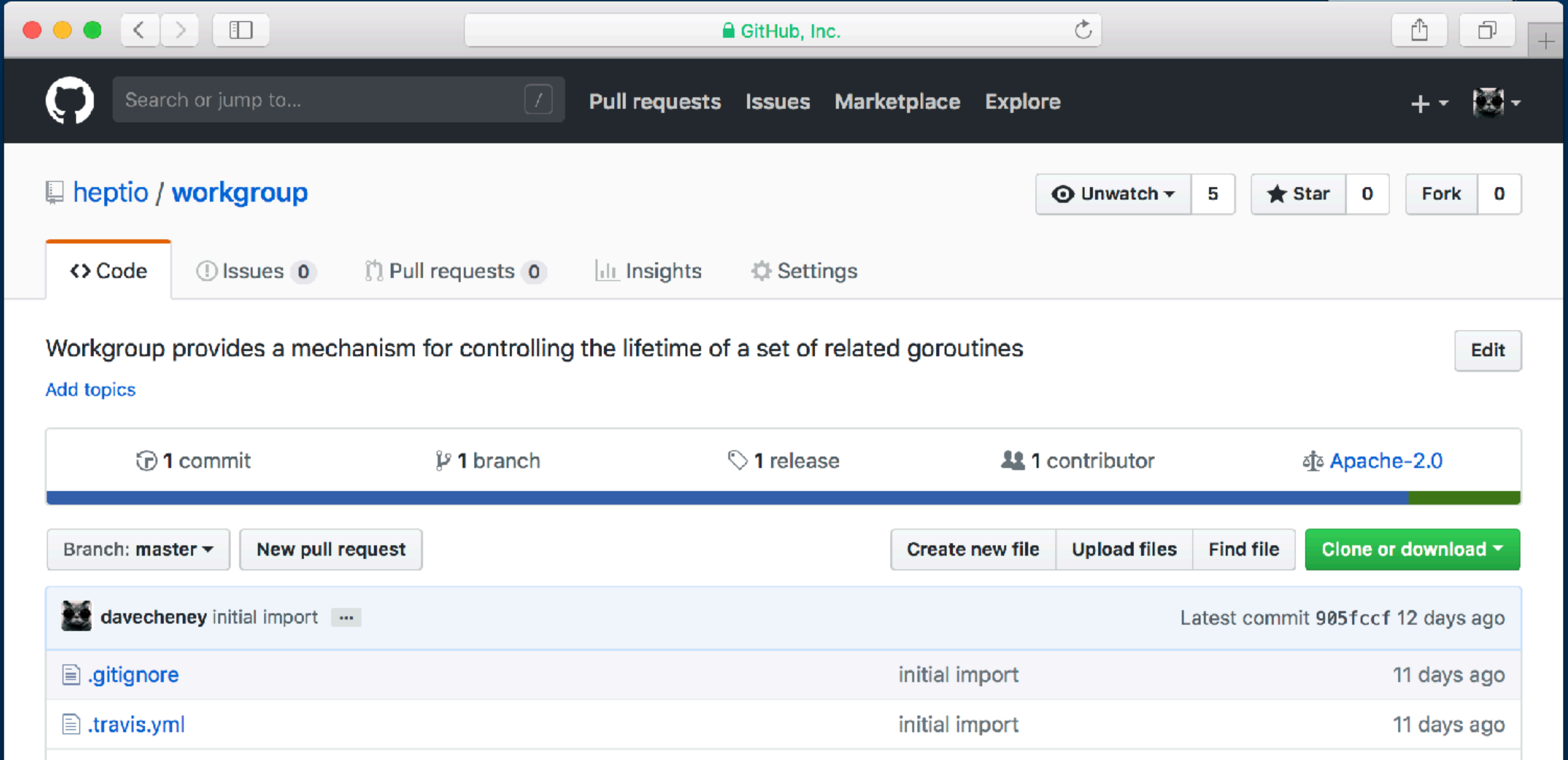
Register the gRPC server

Create individual watchers and register them with the group

Register the /debug/pprof server

Start all the workers, wait until one exits

Now with extra open source



heptio / workgroup

Workgroup provides a mechanism for controlling the lifetime of a set of related goroutines

1 commit | 1 branch | 1 release | 1 contributor | Apache-2.0

davecheney initial import ··· Latest commit 905fccf 12 days ago

.gitignore	initial import	11 days ago
.travis.yml	initial import	11 days ago

Dependency management with dep

Gopkg.toml

```
[[constraint]]
  name = "k8s.io/client-go"
  version = "v8.0.0"

[[constraint]]
  name = "k8s.io/apimachinery"
  version = "kubernetes-1.11.4"

[[constraint]]
  name = "k8s.io/api"
  version = "kubernetes-1.11.4"
```

We don’t commit `vendor/` to our repository

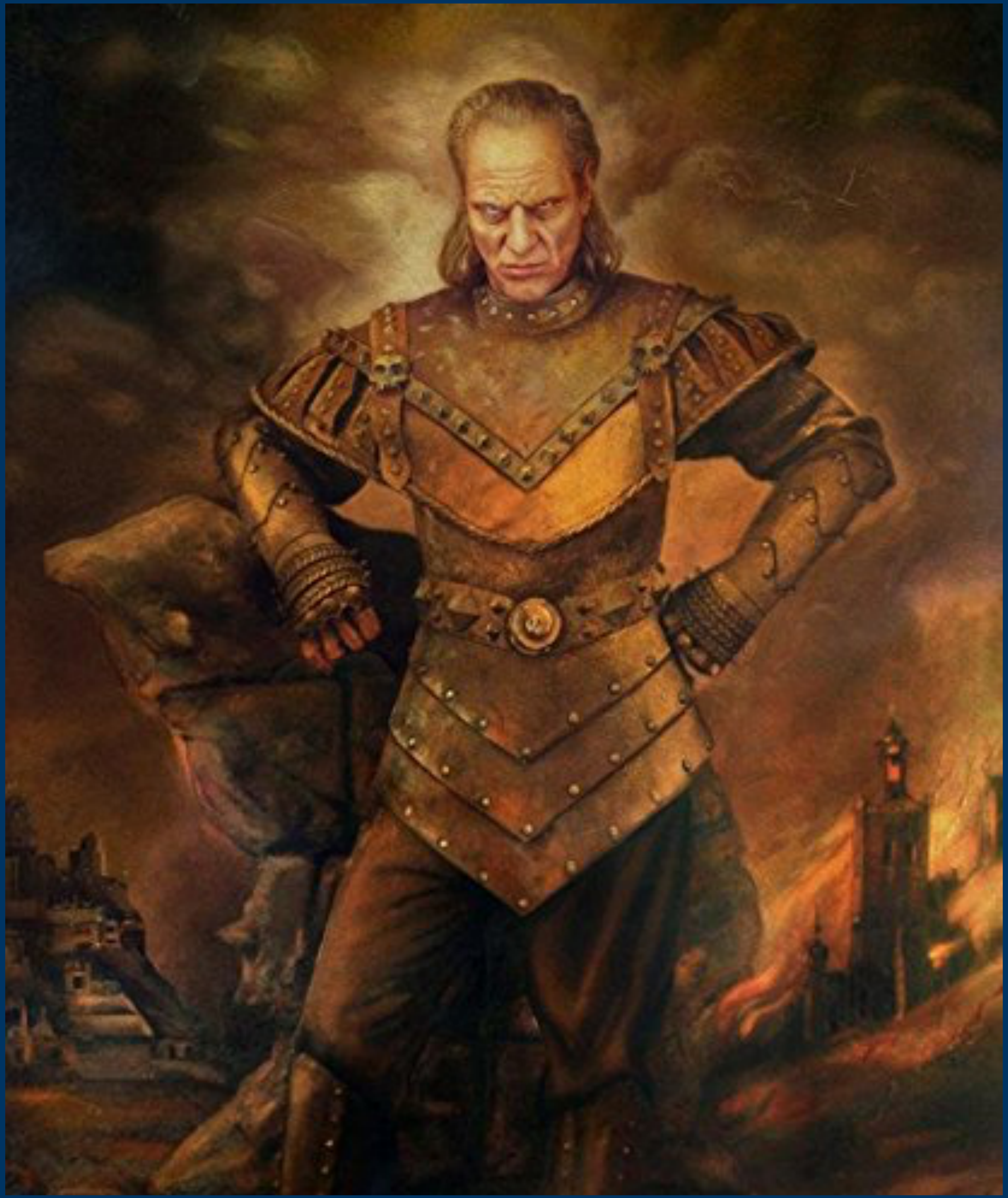
```
% go get -d github.com/heptio/contour
% cd $GOPATH/src/github.com/heptio/contour
% dep ensure -vendor-only
```

If you change branches you may need to run `dep ensure`

Not committing `vendor/` does not protect us against a depdendency going away

What about go modules?

TL;DR the future isn't here yet



Living with Docker

.dockerignore

When you run docker build it copies *everything* in your working directory to the docker daemon 😴

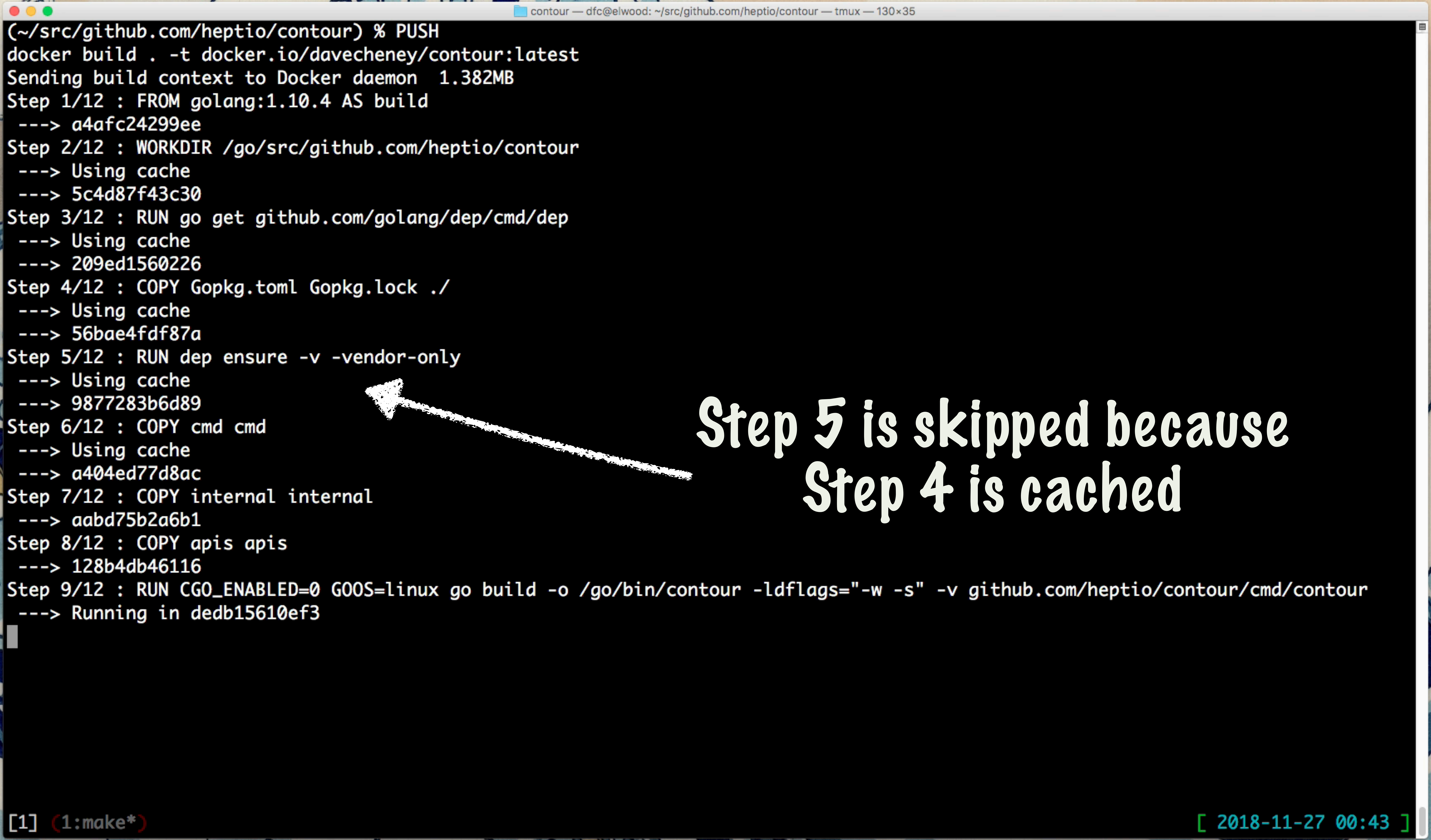
heptio Contour

```
% cat .dockerignore
/.git
/vendor
```

```
% cat Dockerfile
FROM golang:1.10.4 AS build
WORKDIR /go/src/github.com/heptio/contour

RUN go get github.com/golang/dep/cmd/dep
COPY Gopkg.toml Gopkg.lock ./
RUN dep ensure -v -vendor-only

COPY cmd cmd
COPY internal internal
COPY apis apis
RUN CGO_ENABLED=0 GOOS=linux go build -o /go/bin/contour \
  -ldflags=“-w -s" -v github.com/heptio/contour/cmd/contour

FROM alpine:3.8 AS final
RUN apk --no-cache add ca-certificates
COPY --from=build /go/bin/contour /bin/contour
```

only runs if Gopkg.toml or Gopkg.lock have changed

heptio Contour


Terminal Shell Edit View Window Help
Tue 11:43
contour — dfc@elwood: ~/src/github.com/heptio/contour — tmux — 130×35

```
(~/src/github.com/heptio/contour) % PUSH
docker build . -t docker.io/davecheney/contour:latest
Sending build context to Docker daemon  1.382MB
Step 1/12 : FROM golang:1.10.4 AS build
 ---> a4afc24299ee
Step 2/12 : WORKDIR /go/src/github.com/heptio/contour
 ---> Using cache
 ---> 5c4d87f43c30
Step 3/12 : RUN go get github.com/golang/dep/cmd/dep
 ---> Using cache
 ---> 209ed1560226
Step 4/12 : COPY Gopkg.toml Gopkg.lock ./
 ---> Using cache
 ---> 56bae4fdf87a
Step 5/12 : RUN dep ensure -v -vendor-only
 ---> Using cache
 ---> 9877283b6d89
Step 6/12 : COPY cmd cmd
 ---> Using cache
 ---> a404ed77d8ac
Step 7/12 : COPY internal internal
 ---> aabd75b2a6b1
Step 8/12 : COPY apis apis
 ---> 128b4db46116
Step 9/12 : RUN CGO_ENABLED=0 GOOS=linux go build -o /go/bin/contour -ldflags="-w -s" -v github.com/heptio/contour/cmd/contour
 ---> Running in dedb15610ef3

[1] (1:make*)                                                                                          [ 2018-11-27 00:43 ]
```

Step 5 is skipped because
Step 4 is cached

Try to avoid the

`docker build && docker push`

workflow in your inner loop

heptio Contour

Terminal
Shell
Edit
View
Window
Help
Tue 11:22
contour — dfc@elwood: ~/src/github.com/heptio/contour — tmux — 120×33

```
(~/src/github.com/heptio/contour) %

[1] (1:bash*)                                                    [ 2018-11-27 00:22 ]
```

Local development against a live cluster

heptio Contour

Terminal
Shell
Edit
View
Window
Help
Tue 11:25
contour — dfc@elwood: ~/src/github.com/heptio/contour — tmux — 130×35

```
(~/src/github.com/heptio/contour) %
```

```
(~/src/github.com/heptio/contour) %
```

```
[1] (1:bash*)                                                        [ 2018-11-27 00:25 ]
```

Functional Testing

Functional End to End tests are terrible

- Slow …
- Which leads to effort expended to run them in parallel …
- Which tends to make them flakey …
- In my experience end to end tests become a boat anchor on development velocity

So, I put them off as long as I could

But, there are scenarios that unit tests cannot cover …

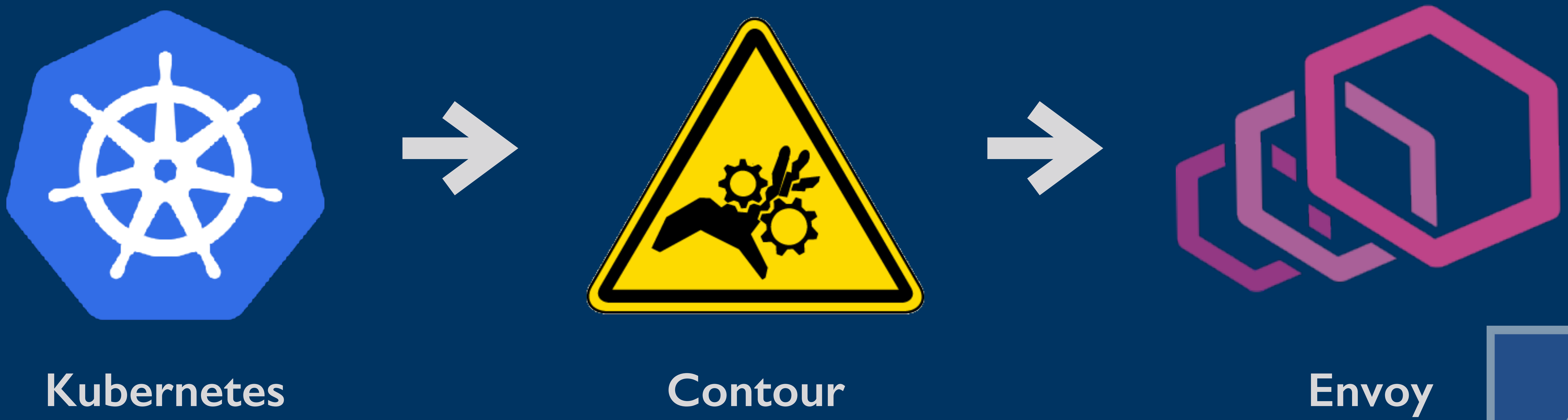
… because there is a moderate impedance mismatch between Kubernetes and Envoy

We need to model the sequence of interactions between Kubernetes and Envoy

What are Contour’s e2e tests not testing?

- We are not testing Kubernetes—we assume it works
- We are not testing Envoy—we hope someone else did that

Contour Architecture Diagram



Kubernetes → Contour → Envoy

heptio
Contour

```
func setup(t *testing.T) (cache.ResourceEventHandler, *grpc.ClientConn, func()) {
        log := logrus.New()
        log.Out = &testWriter{t}

        tr := &contour.Translator{
                FieldLogger: log,
        }

        l, err := net.Listen("tcp", "127.0.0.1:0")
        check(t, err)
        var wg sync.WaitGroup
        wg.Add(1)
        srv := cgrpc.NewAPI(log, tr)
        go func() {
                defer wg.Done()
                srv.Serve(l)
        }()
        cc, err := grpc.Dial(l.Addr().String(), grpc.WithInsecure())
        check(t, err)
        return tr, cc, func() {
                // close client connection
```

Create a contour Translator

Create a gRPC client and dial our server

Create a new gRPC server and bind it to a loopback address

Return a resource handler, client, and shutdown function

Resource handler,
the input

```
// pathological hard case, one service is removed, the other
// is moved to a different port, and its name removed.
func TestClusterRenameUpdateDelete(t *testing.T) {
  rh, cc, done := setup(t)
  defer done()
```

gRPC client, the output

```
  s1 := service("default", "kuard",
    v1.ServicePort{
      Name:       "http",
      Protocol:   "TCP",
      Port:       80,
      TargetPort: intstr.FromInt(8080),
    },
    v1.ServicePort{
      Name:       "https",
      Protocol:   "TCP"
```

Insert s1 into
API server

Query Contour
for the results

Low lights 😒

- Verbose, even with lots of helpers …
- … but at least it’s explicit; after this event from the API, I expect this state.

heptio
Contour

High Lights 😁

- High success rate in reproducing bugs reported in the field.
- Easy to model failing scenarios which enables Test Driven Development 🎉
- Easy way for contributors to add tests.
- Avoid `docker push && k delete po -l app=contour` style debugging

Thank you!

- ☞ github.com/heptio/contour
- ☞ @davecheney
- ☞ dfc@heptio.com

Image: Egon Elbre