How Heptio Built Contour

Any What You Can Learn From Our Experiences

How Heptio Built Contour

Any What You Can Learn From Our Experiences

What does an Ingress Controller do?


INDGANG B
Psykiatrien
i Region Syddanmark

Ingress

To step in

Why can’t I just use a Service type: LoadBalancer?

The Philosophy of Ingress

An ingress controller should take care of the 90% case

The Philosophy of Ingress

The Philosophy of Ingress

Traffic consolidation

The Philosophy of Ingress

Traffic consolidation

TLS management

The Philosophy of Ingress

Traffic consolidation

TLS management

Abstract configuration

The Philosophy of Ingress

Traffic consolidation

TLS management

Abstract configuration

Reverse proxy table stakes

The Philosophy of Ingress

Traffic consolidation

TLS management

Abstract configuration

Reverse proxy table stakes

- Path based routing

The Philosophy of Ingress

Traffic consolidation

TLS management

Abstract configuration

Reverse proxy table stakes

- Path based routing
- HTTP → HTTPS 3xx redirects

The Philosophy of Ingress

Traffic consolidation

TLS management

Abstract configuration

Reverse proxy table stakes

- Path based routing
- HTTP → HTTPS 3xx redirects
- (limited) Request rewriting

What is Contour?

Contour only does what you can describe in the Ingress object

Why did Contour choose Envoy as its foundation?

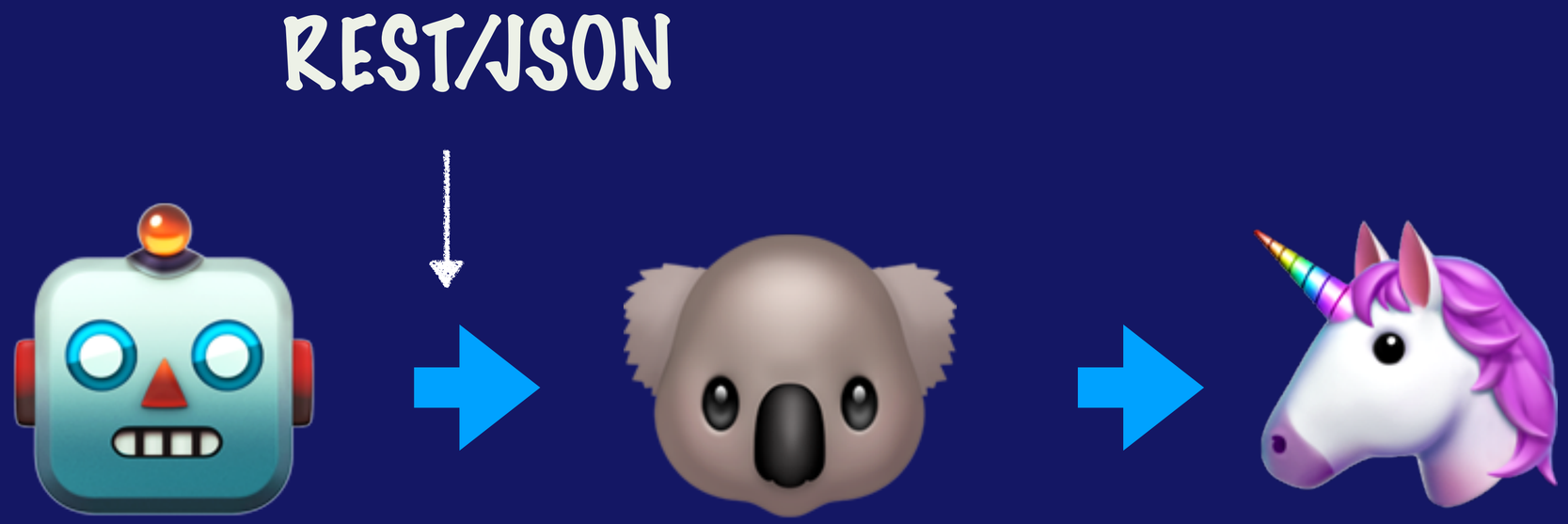
Envoy is the API *client*

Contour is the API *server*

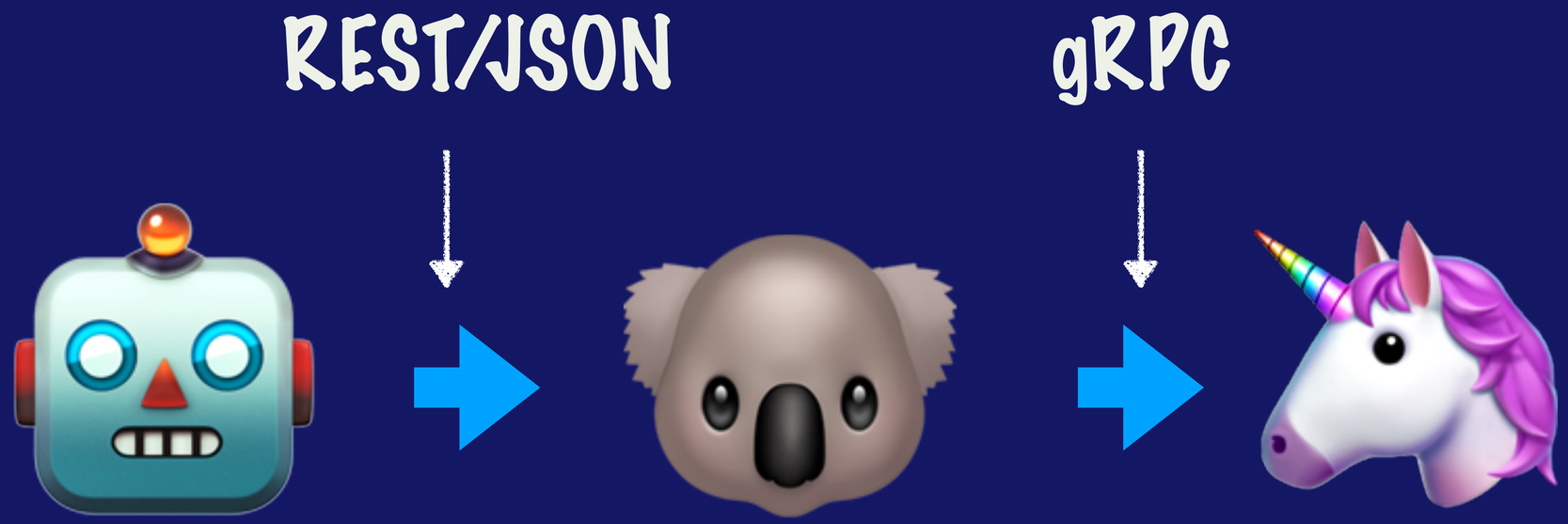
Contour Architecture Diagram

Kubernetes Contour Envoy

Contour Architecture Diagram



Contour Architecture Diagram



Kubernetes and Envoy interoperability

LDS				
RDS				
CDS				
EDS				

Kubernetes and Envoy interoperability

	Ingress			
LDS	😀			
RDS	😀			
CDS				
EDS				

Kubernetes and Envoy interoperability

	Ingress	Service		
LDS	😀			
RDS	😀			
CDS		😀		
EDS				

Kubernetes and Envoy interoperability

	Ingress	Service	Endpoints	
LDS	😀			
RDS	😀			
CDS		😀		
EDS			😀	

Kubernetes and Envoy interoperability

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

Got Service Mesh?

May 4

B4-M1 14:16-14:24

Contour, the project

As of April 30, Contour is around 9900 LOC

As of April 30, Contour is around 9900 LOC

2900 source, 7000 tests

As of April 30, Contour is around 9900 LOC

2900 source, 7000 tests

heptio
Contour

```
contour
├── cmd
|   └── contour
├── internal
|   ├── contour
|   ├── e2e
|   ├── envoy
|   ├── grpc
|   ├── k8s
|   └── workgroup
└── vendor
```

```
contour
├── cmd
|   └── contour
├── internal
|   ├── contour
|   ├── e2e
|   ├── envoy
|   ├── grpc
|   ├── k8s
|   └── workgroup
└── vendor
```

The actual contour command

heptio
Contour

```
contour
├── cmd
|   └── contour
├── internal
|   ├── contour
|   ├── e2e
|   ├── envoy
|   ├── grpc
|   ├── k8s
|   └── workgroup
└── vendor
```

The translator; turns k8s objects into Envoy

heptio Contour

```
contour
├── cmd
|   └── contour
├── internal
|   ├── contour
|   ├── e2e
|   ├── envoy
|   ├── grpc
|   ├── k8s
|   └── workgroup
└── vendor
```

Integration tests

heptio
Contour

```
contour
├── cmd
|   └── contour
├── internal
|   ├── contour
|   ├── e2e
|   ├── envoy
|   ├── grpc
|   ├── k8s
|   └── workgroup
└── vendor
```

Envoy helpers; bootstrap config

```
contour
├── cmd
|   └── contour
├── internal
|   ├── contour
|   ├── e2e
|   ├── envoy
|   ├── grpc
|   ├── k8s
|   └── workgroup
└── vendor
```

gRPC server; implements the xDS protocol

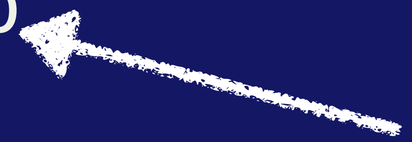
heptio Contour

```
contour
├── cmd
|   └── contour
├── internal
|   ├── contour
|   ├── e2e
|   ├── envoy
|   ├── grpc
|   ├── k8s
|   └── workgroup
└── vendor
```

Kuberneters helpers

heptio
Contour

```
contour
├── cmd
|   └── contour
├── internal
|   ├── contour
|   ├── e2e
|   ├── envoy
|   ├── grpc
|   ├── k8s
|   └── workgroup
└── vendor
```



Goroutine helpers

Consider `internal/` for packages that you don't want other prorjects to depend on

Goroutine management

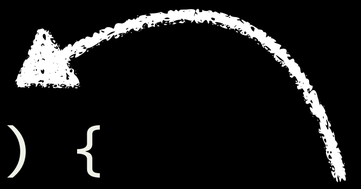
github.com/heptio/contour/internal/workgroup

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

```
// Group manages a set of goroutines with related lifetimes.
type Group struct {
        fn []func(<-chan struct{})
}

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

// Run exectues each function registered with Add in
// its own goroutine.
// Run blocks until each function has returned.
// The first function to return will trigger the closure of
the channel
```

```
// Group manages a set of goroutines with related lifetimes.
type Group struct {
        fn []func(<-chan struct{})
}

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

// Run exectues each function registered with Add in
// its own goroutine.
// Run blocks until each function has returned.
// The first function to return will trigger the closure of
the channel
```

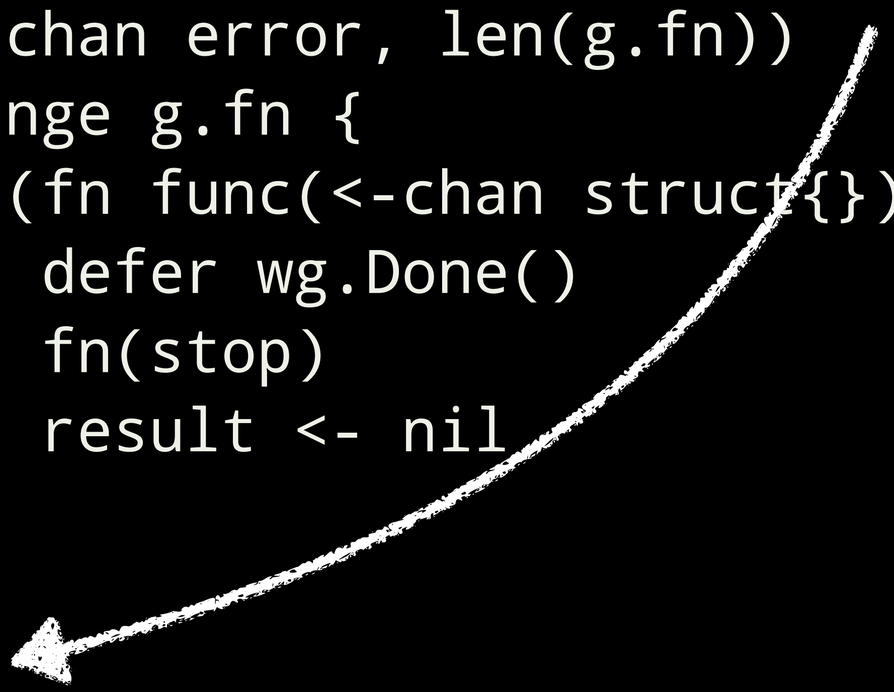


Register functions to be run as coroutines in the group

```
func (g *Group) Run() {
        var wg sync.WaitGroup
        wg.Add(len(g.fn))

        stop := make(chan struct{})
        result := make(chan error, len(g.fn))
        for _, fn := range g.fn {
                go func(fn func(<-chan struct{})) {
                        defer wg.Done()
                        fn(stop)
                        result <- nil
                }(fn)
        }

        <-result    // wait for first goroutine to exit
        close(stop) // ask others to exit
        wg.Wait()   // wait for all goroutines to exit
}
```

```
func (g *Group) Run() {
        var wg sync.WaitGroup
        wg.Add(len(g.fn))

        stop := make(chan struct{})
        result := make(chan error, len(g.fn))
        for _, fn := range g.fn {
                go func(fn func(<-chan struct{})) {
                        defer wg.Done()
                        fn(stop)
                        result <- nil
                }(fn)
        }

        <-result    // wait for first goroutine to exit
        close(stop) // ask others to exit
        wg.Wait()   // wait for all goroutines to exit
}
```

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



```
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{}) {
        addr := net.JoinHostPort(*xdsAddr, strconv.Itoa(*xdsPort))
        l, err := net.Listen("tcp", addr)
        if err != nil {
                log.Errorf("could not listen on %s: %v", addr, err)
                return
```

Make a new 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{}) {
        addr := net.JoinHostPort(*xdsAddr, strconv.Itoa(*xdsPort))
        l, err := net.Listen("tcp", addr)
        if err != nil {
                log.Errorf("could not listen on %s: %v", addr, err)
                return
```

```
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{}) {
        addr := net.JoinHostPort(*xdsAddr, strconv.Itoa(*xdsPort))
        l, err := net.Listen("tcp", addr)
        if err != nil {
                log.Errorf("could not listen on %s: %v", addr, err)
                return
```

Create individual watchers and register them with 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{}) {
        addr := net.JoinHostPort(*xdsAddr, strconv.Itoa(*xdsPort))
        l, err := net.Listen("tcp", addr)
        if err != nil {
                log.Errorf("could not listen on %s: %v", addr, err)
                return
```

Register the /debug/pprof server

```
k8s.WatchIngress(&g, client)
k8s.WatchSecrets(&g, client)

g.Add(debug.Start)

g.Add(func(stop <-chan struct{}) {
        addr := net.JoinHostPort(*xdsAddr, strconv.Itoa(*xdsPort))
        l, err := net.Listen("tcp", addr)
        if err != nil {
                log.Errorf("could not listen on %s: %v", addr, err)
                return
        }
        s := grpc.NewAPI(log, t)
        s.Serve(l)
})

g.Run()
```

Register the gRPC server

```
k8s.WatchIngress(&g, client)
k8s.WatchSecrets(&g, client)

g.Add(debug.Start)

g.Add(func(stop <-chan struct{}) {
        addr := net.JoinHostPort(*xdsAddr, strconv.Itoa(*xdsPort))
        l, err := net.Listen("tcp", addr)
        if err != nil {
                log.Errorf("could not listen on %s: %v", addr, err)
                return
        }
        s := grpc.NewAPI(log, t)
        s.Serve(l)
})

g.Run()
```

Start all the workers,
wait until one exits

Ways To Do Things - Peter Bourgon - Release Party #GoSF

https://www.youtube.com/watch?v=LHe1Cb_Ud_M

Handling concurrency

`github.com/heptio/contour/internal/k8s.Buffer`

Watchers call back to Contour concurrently

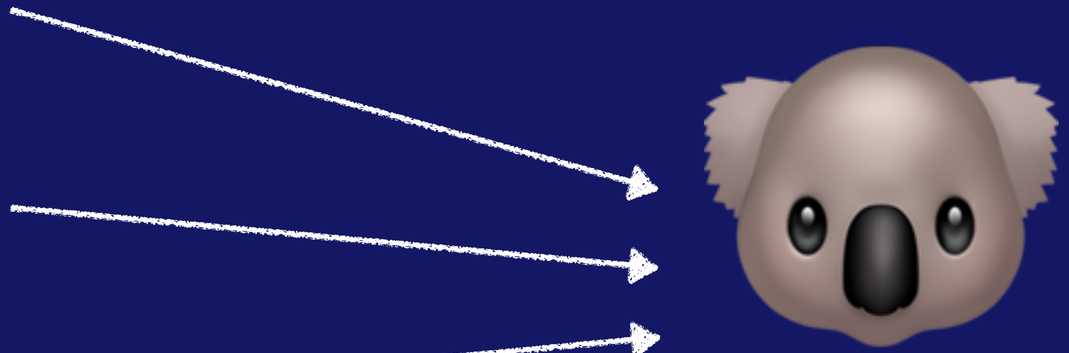
Kubernetes

Ingress

Services

Endpoints

Secrets



Contour

Kubernetes and Envoy interoperability

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

`sync.Mutex`
to the rescue ...

Channels to the rescue

```
func NewBuffer(g *workgroup.Group, rh cache.ResourceEventHandler, size int)
cache.ResourceEventHandler {
        buf := &buffer{
                ev:        make(chan interface{}, size),
                rh:        rh,
        }
        g.Add(buf.loop)
        return buf
}

func (b *buffer) OnAdd(obj interface{}) {
        b.send(&addEvent{obj})
}

func (b *buffer) OnUpdate(oldObj, newObj interface{}) {
        b.send(&updateEvent{oldObj, newObj})
}

func (b *buffer) OnDelete(obj interface{}) {
        b.send(&deleteEvent{obj})
}
```

```
func NewBuffer(g *workgroup.Group, rh cache.ResourceEventHandler, size int)
cache.ResourceEventHandler {
        buf := &buffer{
                ev:         make(chan interface{}, size),
                rh:         rh,
        }
        g.Add(buf.loop)
        return buf
}

func (b *buffer) OnAdd(obj interface{}) {
        b.send(&addEvent{obj})
}

func (b *buffer) OnUpdate(oldObj, newObj interface{}) {
        b.send(&updateEvent{oldObj, newObj})
}

func (b *buffer) OnDelete(obj interface{}) {
        b.send(&deleteEvent{obj})
}
```

Create new buffer; register with group

```
func NewBuffer(g *workgroup.Group, rh cache.ResourceEventHandler, size int)
cache.ResourceEventHandler {
        buf := &buffer{
                ev:         make(chan interface{}, size),
                rh:         rh,
        }
        g.Add(buf.loop)
        return buf
}

func (b *buffer) OnAdd(obj interface{}) {
        b.send(&addEvent{obj})
}

func (b *buffer) OnUpdate(oldObj, newObj interface{}) {
        b.send(&updateEvent{oldObj, newObj})
}

func (b *buffer) OnDelete(obj interface{}) {
        b.send(&deleteEvent{obj})
}
```

Buffer fulfills the ResourceEventHandler interface

```
func (b *buffer) OnAdd(obj interface{}) {
        b.send(&addEvent{obj})
}

func (b *buffer) OnUpdate(oldObj, newObj interface{}) {
        b.send(&updateEvent{oldObj, newObj})
}

func (b *buffer) OnDelete(obj interface{}) {
        b.send(&deleteEvent{obj})
}

func (b *buffer) send(ev interface{}) {
        select {
        case b.ev <- ev:
                // all good
        default:
                b.Printf("event channel is full, len: %v, cap: %v", len(b.ev), cap(b.ev
                b.ev <- ev
        }
}
```

```
func (b *buffer) OnAdd(obj interface{}) {
        b.send(&addEvent{obj})
}

func (b *buffer) OnUpdate(oldObj, newObj interface{}) {
        b.send(&updateEvent{oldObj, newObj})
}

func (b *buffer) OnDelete(obj interface{}) {
        b.send(&deleteEvent{obj})
}

func (b *buffer) send(ev interface{}) {
        select {
        case b.ev <- ev:
                // all good
        default:
                b.Printf("event channel is full, len: %v, cap: %v", len(b.ev), cap(b.ev
                b.ev <- ev
        }
}
```

Send forwards ResourceEvents to a channel

```
func (b *buffer) loop(stop <-chan struct{}) {
        b.Println("started")
        defer b.Println("stopped")

        for {
                select {
                case ev := <-b.ev:
                        switch ev := ev.(type) {
                        case *addEvent:
                                b.rh.OnAdd(ev.obj)
                        case *updateEvent:
                                b.rh.OnUpdate(ev.oldObj, ev.newObj)
                        case *deleteEvent:
                                b.rh.OnDelete(ev.obj)
                        default:
                                b.Printf("unhandled event type: %T: %v", ev, ev)
                        }
                case <-stop:
                        return
                }
        }
}
```

```
func (b *buffer) loop(stop <-chan struct{}) {
        b.Println("started")
        defer b.Println("stopped")

        for {
                select {
                case ev := <-b.ev:
                        switch ev := ev.(type) {
                        case *addEvent:
                                b.rh.OnAdd(ev.obj)
                        case *updateEvent:
                                b.rh.OnUpdate(ev.oldObj, ev.newObj)
                        case *deleteEvent:
                                b.rh.OnDelete(ev.obj)
                        default:
                                b.Printf("unhandled event type: %T: %v", ev, ev)
                        }
                case <-stop:
                        return
                }
        }
}
```

Loop takes events off the channel, calls the backing handler, until the group tells it to stop

Dependency management with dep

Gopkg.toml

```
[[constraint]]
  name="k8s.io/client-go"
  version=“v7.0.0"

[[constraint]]
  name="k8s.io/api"
  branch=“release-1.10"

[[constraint]]
  name="k8s.io/apimachinery"
  branch="release-1.10"
```

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
```

Living with Docker

.dockerignore

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

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

```
% cat Dockerfile
FROM golang:1.10
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
RUN CGO_ENABLED=0 GOOS=linux go install -ldflags="-w -s" -v
github.com/heptio/contour/cmd/contour

FROM alpine:latest
RUN apk --no-cache add ca-certificates
COPY --from=0 /go/bin/contour /bin/contour
```

```
% cat Dockerfile
FROM golang:1.10
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
RUN CGO_ENABLED=0 GOOS=linux go install -ldflags="-w -s" -v
github.com/heptio/contour/cmd/contour

FROM alpine:latest
RUN apk --no-cache add ca-certificates
COPY --from=0 /go/bin/contour /bin/contour
```

```
% cat Dockerfile
FROM golang:1.10
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
RUN CGO_ENABLED=0 GOOS=linux go install -ldflags="-w -s" -v
github.com/heptio/contour/cmd/contour

FROM alpine:latest
RUN apk --no-cache add ca-certificates
COPY --from=0 /go/bin/contour /bin/contour
```

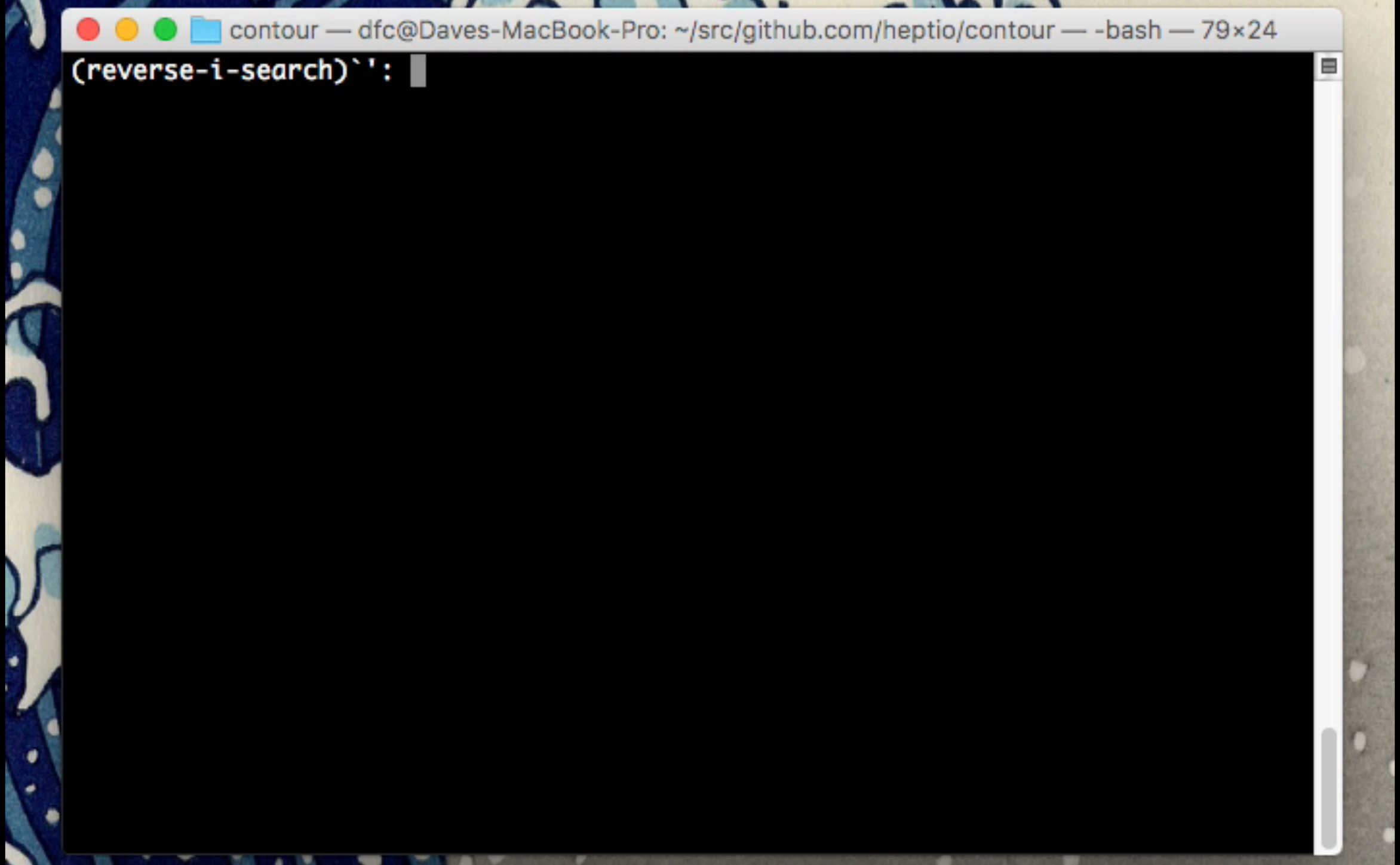
```
% cat Dockerfile
FROM golang:1.10
WORKDIR /go/src/github.com/heptio/contour

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
RUN CGO_ENABLED=0 GOOS=linux go install -ldflags="-w -s" -v
github.com/heptio/contour/cmd/contour

FROM alpine:latest
RUN apk --no-cache add ca-certificates
COPY --from=0 /go/bin/contour /bin/contour
```

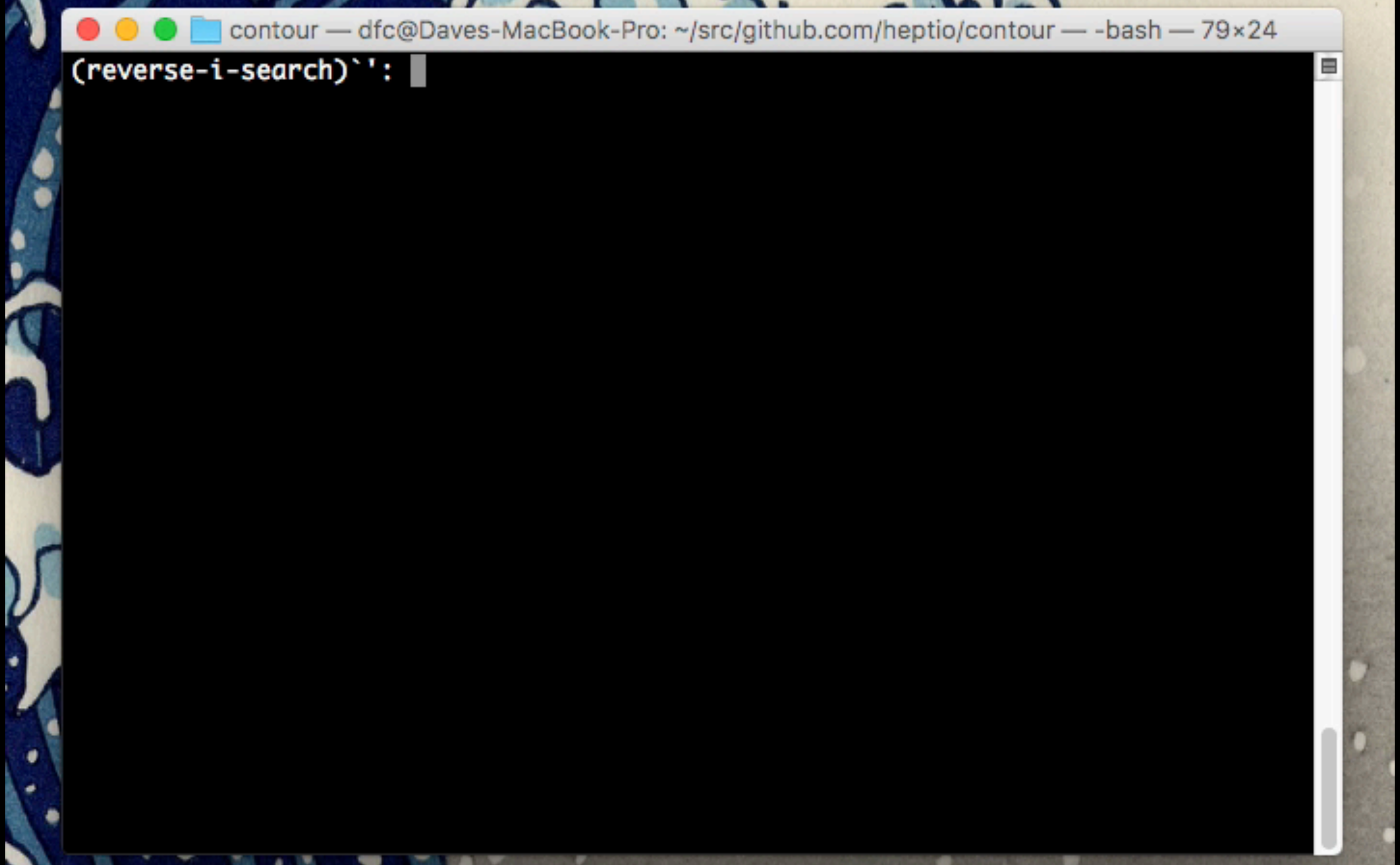
only runs if Gopkg.toml or Gopkg.lock have changed

heptio Contour

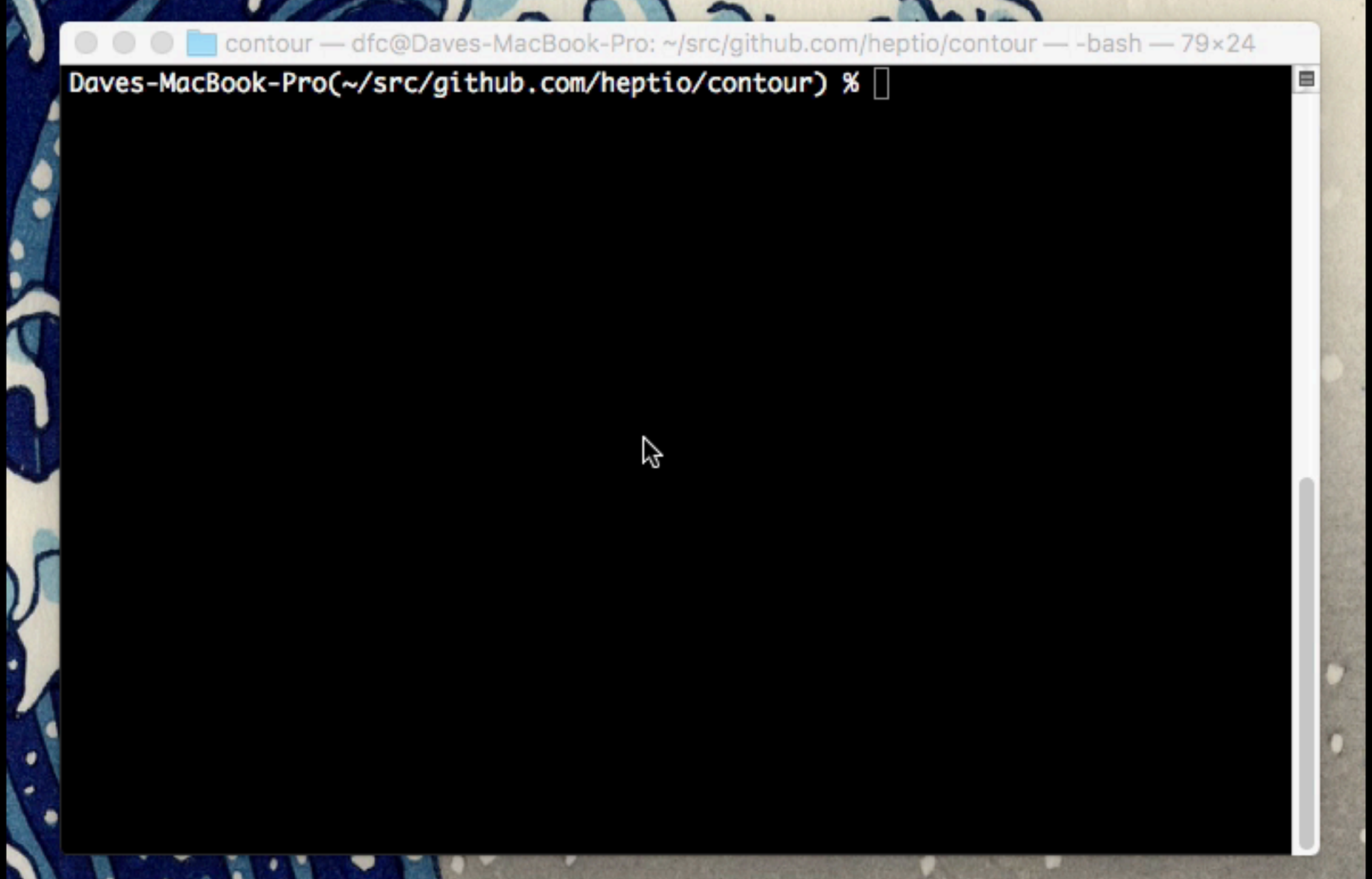
```
% cat Dockerfile
FROM golang:1.10
WORKDIR /go/src/github.com/heptio/contour

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
RUN CGO_ENABLED=0 GOOS=linux go install -ldflags="-w -s" -v
github.com/heptio/contour/cmd/contour

FROM alpine:latest
RUN apk --no-cache add ca-certificates
COPY --from=0 /go/bin/contour /bin/contour
```

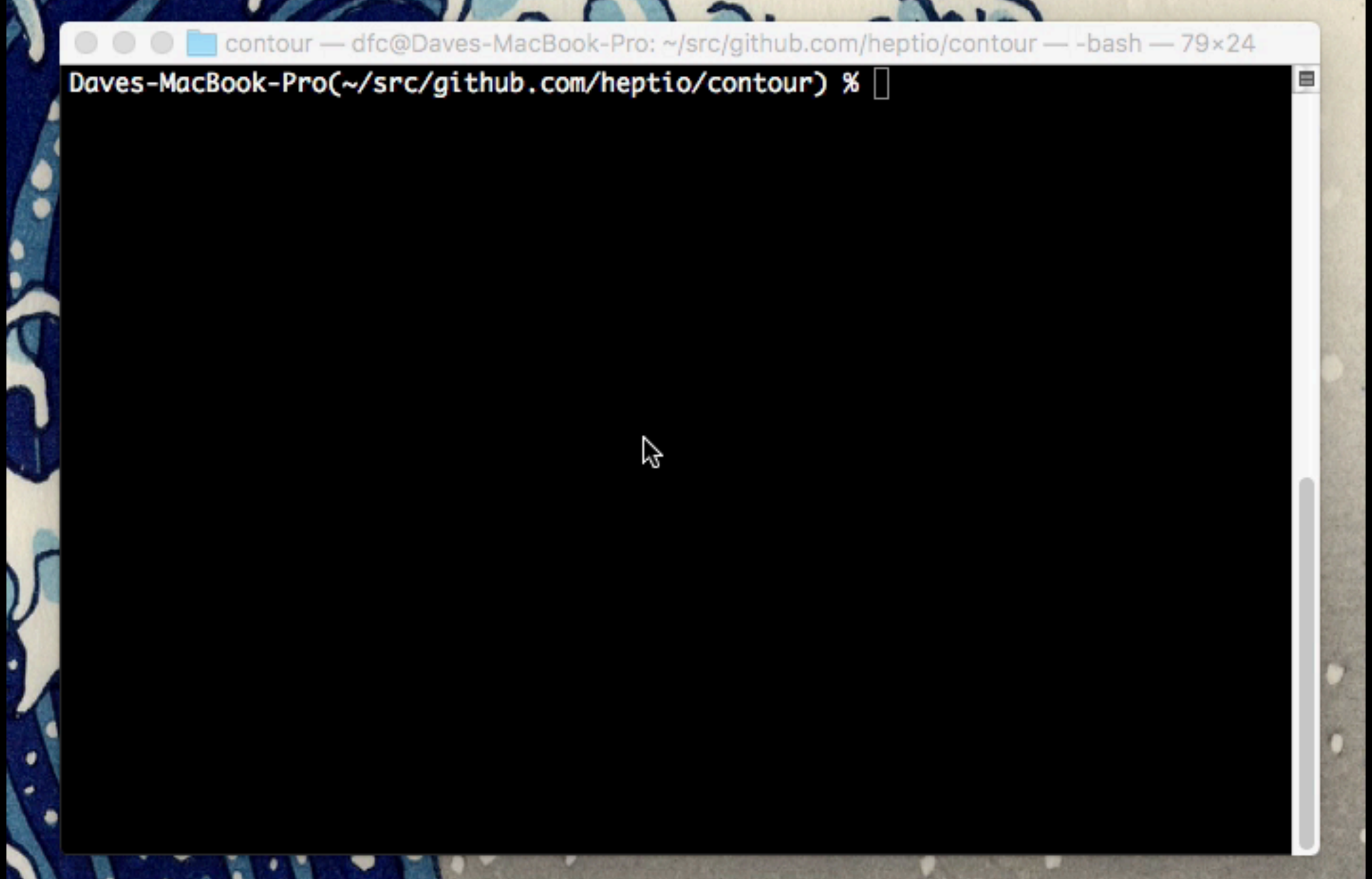
```
% cat Dockerfile
FROM golang:1.10
WORKDIR /go/src/github.com/heptio/contour

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
RUN CGO_ENABLED=0 GOOS=linux go install -ldflags="-w -s" -v
github.com/heptio/contour/cmd/contour

FROM alpine:latest
RUN apk --no-cache add ca-certificates
COPY --from=0 /go/bin/contour /bin/contour
```


contour — dfc@Daves-MacBook-Pro: ~/src/github.com/heptio/contour — -bash — 79×24
(reverse-i-search)`':

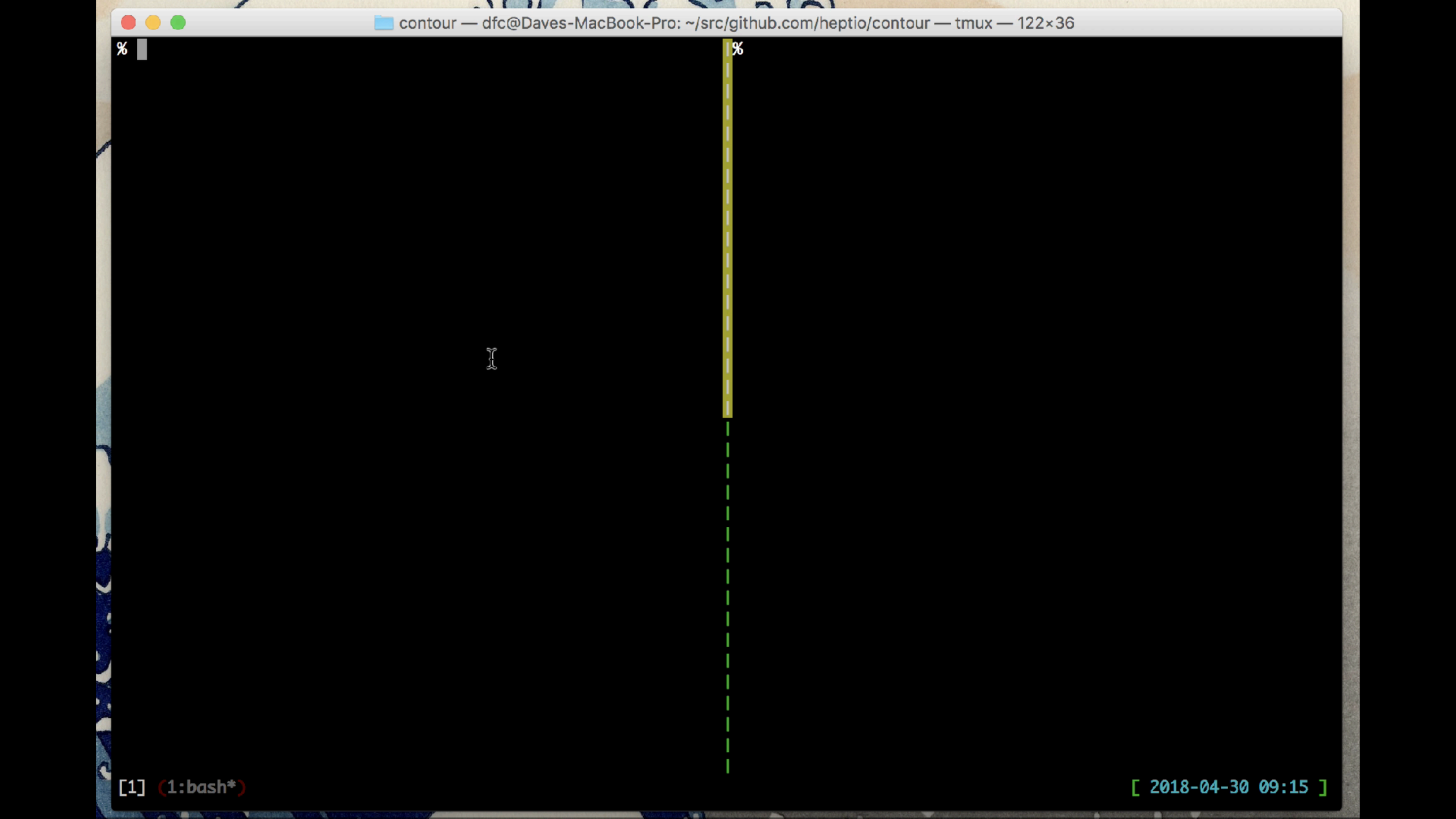

contour — dfc@Daves-MacBook-Pro: ~/src/github.com/heptio/contour — -bash — 79×24
(reverse-i-search)`':

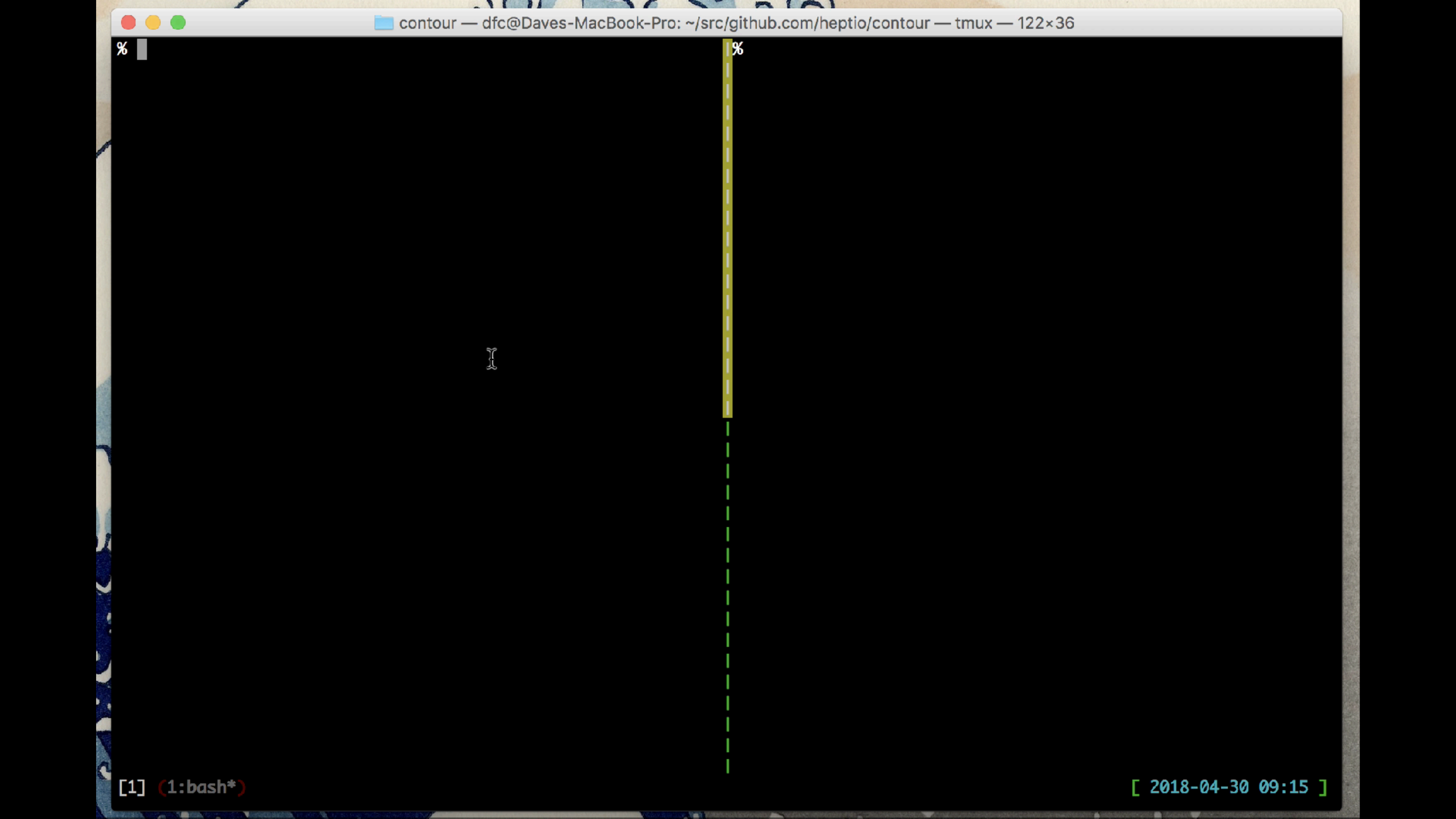
Try to avoid the
`docker build && docker push`
workflow in your inner loop


contour — dfc@Daves-MacBook-Pro: ~/src/github.com/heptio/contour — -bash — 79×24
Daves-MacBook-Pro(~/src/github.com/heptio/contour) %


contour — dfc@Daves-MacBook-Pro: ~/src/github.com/heptio/contour — -bash — 79×24
Daves-MacBook-Pro(~/src/github.com/heptio/contour) %

Local development against a live cluster


contour — dfc@Daves-MacBook-Pro: ~/src/github.com/heptio/contour — tmux — 122×36
%
%
[1] (1:bash*)
[2018-04-30 09:15]


contour — dfc@Daves-MacBook-Pro: ~/src/github.com/heptio/contour — tmux — 122×36
%
%
[1] (1:bash*)
[2018-04-30 09:15]

Functional Testing

~~Functional~~ End to End tests are terrible

~~Functional~~ End to End tests are terrible

- Slow …

~~Functional~~ End to End tests are terrible

- Slow …
- Which leads to effort expended to run them in parallel …

~~Functional~~ End to End tests are terrible

- Slow …
- Which leads to effort expended to run them in parallel …
- Which tends to make them flakey …

~~Functional~~ End to End tests are terrible

- Slow …
- Which leads to effort expended to run them in parallel …
- Which tends to make them flakey …
- IMO end to end tests become a boat anchor on 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

Functional Testing Requirements

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

Contour Architecture Diagram

Kubernetes Contour Envoy

What are Contour's e2e tests *not* testing?

What are Contour's e2e tests *not* testing?

- We are *not* testing Kubernetes (we assume it works)

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

Contour Architecture Diagram

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
```

```
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

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

```
        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
                cc.Close()

                // shut down listener, stop server and wait for it to stop
                l.Close()
                srv.Stop()
                wg.Wait()
        }
}
```

```
        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
                cc.Close()

                // shut down listener, stop server and wait for it to stop
                l.Close()
                srv.Stop()
                wg.Wait()
        }
}
```

Create a gRPC client and dial our server

```
        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
                cc.Close()

                // shut down listener, stop server and wait for it to stop
                l.Close()
                srv.Stop()
                wg.Wait()
        }
}
```

Return a resource handler, client, and shutdown function

```
// 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()

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

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()

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

Resource handler, the input

gRPC client, the output

```
// 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()

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

```
s1 := service("default", "kuard",
  v1.ServicePort{
    Name:       "http",
    Protocol:   "TCP",
    Port:       80,
    TargetPort: intstr.FromInt(8080),
  },
  v1.ServicePort{
    Name:       "https",
    Protocol:   "TCP",
    Port:       443,
    TargetPort: intstr.FromInt(8443),
  },
)

rh.OnAdd(s1)
assertEqual(t, &v2.DiscoveryResponse{
```

```
s1 := service("default", "kuard",
   v1.ServicePort{
      Name:       "http",
      Protocol:   "TCP",
      Port:       80,
      TargetPort: intstr.FromInt(8080),
   },
   v1.ServicePort{
      Name:       "https",
      Protocol:   "TCP",
      Port:       443,
      TargetPort: intstr.FromInt(8443),
   },
)

rh.OnAdd(s1)
assertEqual(t, &v2.DiscoveryResponse{
```

Insert s1 into API server

```
      Port:       443,
      TargetPort: intstr.FromInt(8443),
    },
  )

  rh.OnAdd(s1)
  assertEqual(t, &v2.DiscoveryResponse{
    VersionInfo: "0",
    Resources: []types.Any{
      any(t, cluster("default/kuard/443", "default/kuard/https")),
      any(t, cluster("default/kuard/80", "default/kuard/http")),
      any(t, cluster("default/kuard/http", "default/kuard/http")),
      any(t, cluster("default/kuard/https", "default/kuard/https")),
    },
    TypeUrl: clusterType,
    Nonce:   "0",
  }, fetchCDS(t, cc))
```

```
      Port:       443,
      TargetPort: intstr.FromInt(8443),
   },
)

rh.OnAdd(s1)
assertEqual(t, &v2.DiscoveryResponse{
   VersionInfo: "0",
   Resources: []types.Any{
      any(t, cluster("default/kuard/443", "default/kuard/https")),
      any(t, cluster("default/kuard/80", "default/kuard/http")),
      any(t, cluster("default/kuard/http", "default/kuard/http")),
      any(t, cluster("default/kuard/https", "default/kuard/https")
   },
   TypeUrl: clusterType,
   Nonce:   "0",
}, fetchCDS(t, cc))
```

Query Contour
for the results

```
        any(t, cluster("default/kuard/80", "default/kuard/http")),
        any(t, cluster("default/kuard/http", "default/kuard/http")),
        any(t, cluster("default/kuard/https", "default/kuard/https")),
    },
    TypeUrl: clusterType,
    Nonce:   "0",
}, fetchCDS(t, cc))

// cleanup and check
rh.OnDelete(s1)
assertEqual(t, &v2.DiscoveryResponse{
    VersionInfo: "0",
    Resources:   []types.Any{},
    TypeUrl:     clusterType,
    Nonce:       "0",
}, fetchCDS(t, cc))
}
```

Low lights 😒

Low lights 😒

- Verbose, even with lots of helpers …

Low lights 😒

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

High Lights 😄

High Lights 😄

- High success rate in reproducing bugs reported in the field.

High Lights 😄

- High success rate in reproducing bugs reported in the field.
- Easy way for contributors to add tests.

High Lights 😄

- High success rate in reproducing bugs reported in the field.
- Easy way for contributors to add tests.
- Easy to model failing scenarios which enables Test Driven Development 🎉

High Lights 😄

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

Thank you for listening!

Questions?

@davecheney – dfc@heptio.com