



KUBERNATIC

Marvin Beckers

# Ephemeral Containers in Action

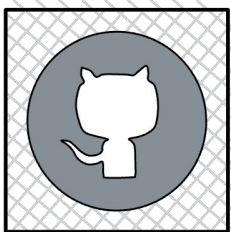
Running a Go Debugger in Kubernetes

SEP 11–13, 2023

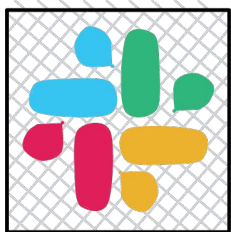
CONTAINER  
days

## Marvin Beckers

- Team Lead in Engineering @ Kubermatic
- Sysadmin turned software engineer
- Working with Go and Kubernetes since 2018



GitHub



Slack

embik



# Agenda

1. Basics of Debugging (Go)
2. Remote Debugging
3. A Primer on Ephemeral Containers
  - a. Command Line Tooling
4. Delve as Ephemeral Container
5. Remote Debugging in VS Code
6. Live Demo

# Basics of Debugging (Go)



# DELVE

A Debugger for the Go Programming Language

# Are We Capable (of Debugging)?

## CAP\_SYS\_PTRACE

- \* Trace arbitrary processes using `ptrace(2)`;
- \* apply `get_robust_list(2)` to arbitrary processes;
- \* transfer data to or from the memory of arbitrary processes using `process_vm_readv(2)` and `process_vm_writev(2)`;
- \* inspect processes using `kcmp(2)`.

`man capabilities 7`

## Compile Flags You *Might* Need

```
$ go build -gcflags="all=-N -l"
```

```
-gcflags '[pattern=]arg list'
```

arguments to pass on each go tool compile invocation.

```
-N disable optimizations
```

```
-l disable inlining
```

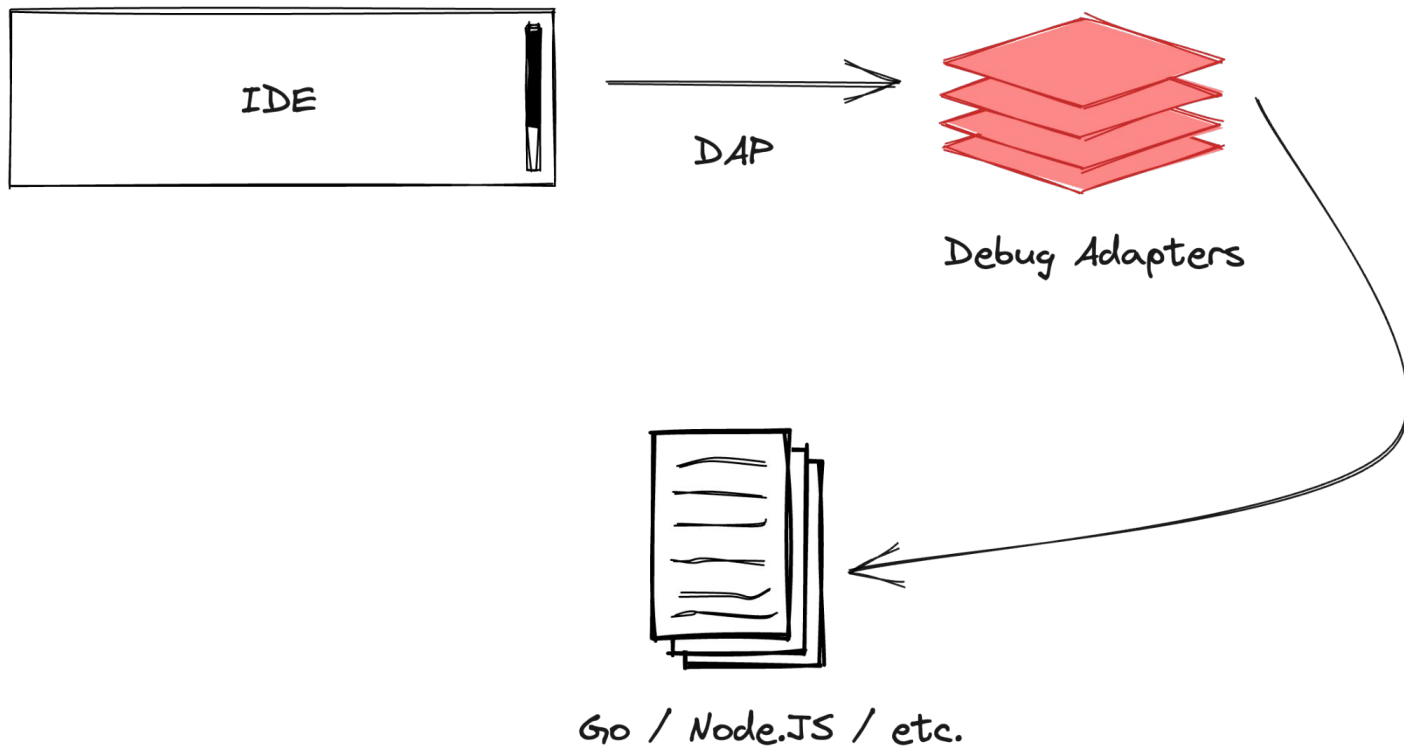
**Debugging can work without this, but e.g. variables might disappear!**

# Remote Debugging

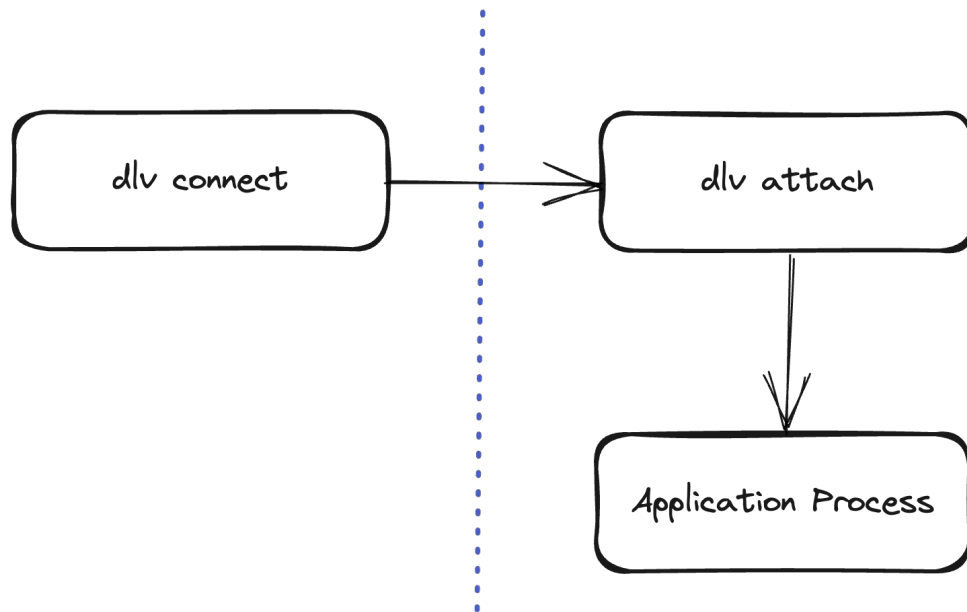




# DAP - Debug Adapter Protocol

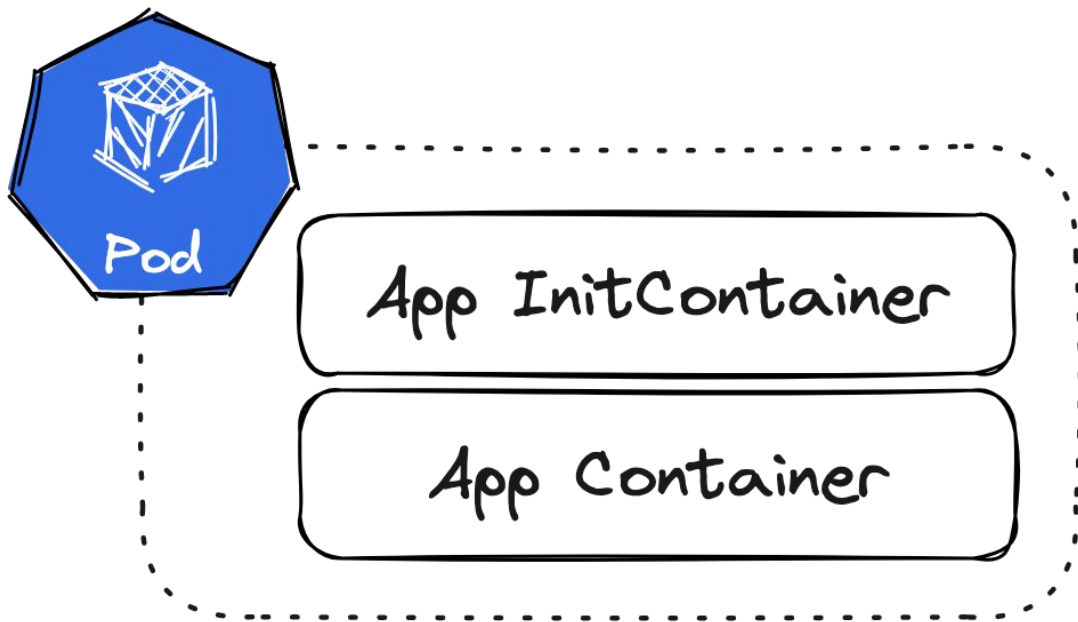


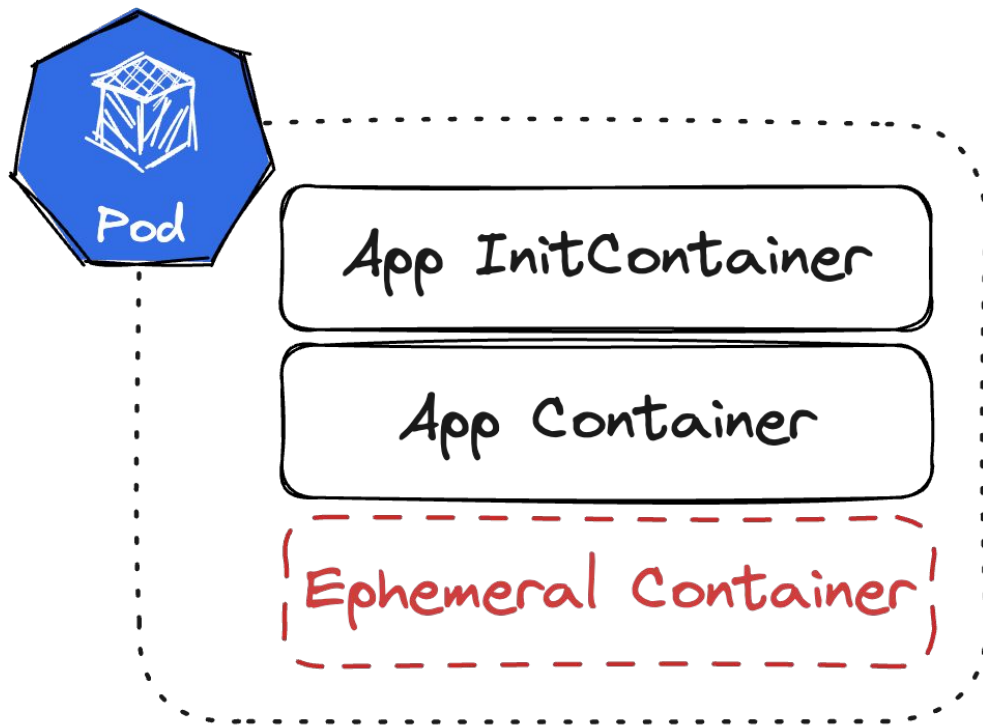
## Headless Delve Instance



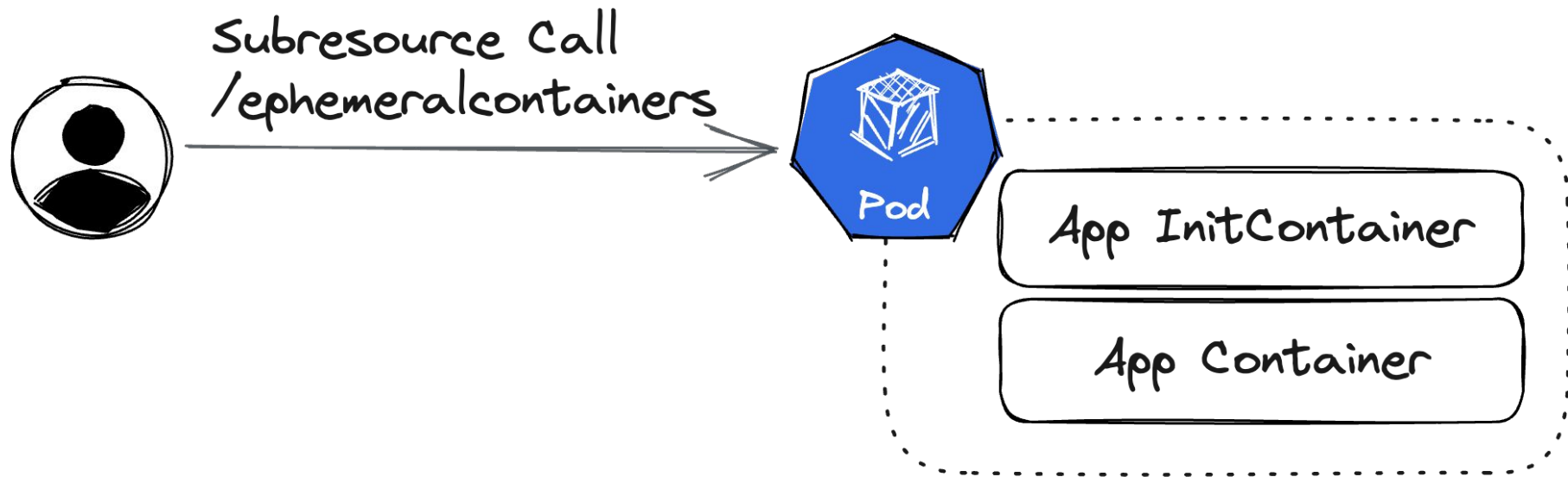
```
$ dlv attach <Target PID> --listen=:2345 --headless=true
```

# A Primer on Ephemeral Containers





With Kubernetes 1.25+



```
client.Pods(namespace).UpdateEphemeralContainers(...)
```

# kubectl debug

```
$ kubectl debug <pod> -it \  
  --image=busybox \  
  --target=<container>
```



Defaulting debug container name to debugger-jwl89.  
If you don't see a command prompt, try pressing enter.

```
/ #
```

```
/ #
```

```
/ # ps waux
```

PID	USER	TIME	COMMAND
1	root	0:00	/bin/app
30	root	0:00	sh
36	root	0:00	ps waux

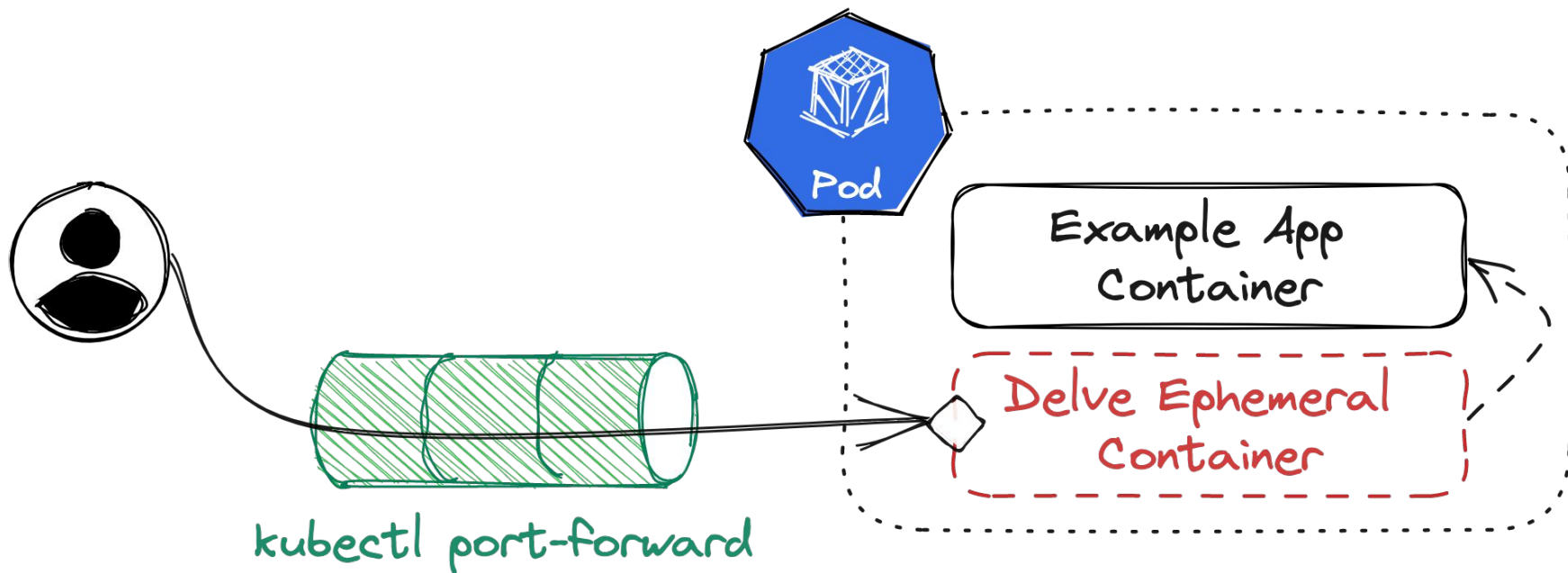
# Find the right `-profile`

... because you don't want apps to have `CAP_SYS_PTRACE`

# Delve as Ephemeral Container

## We End Up With ...

```
$ kubectl debug \  
  --profile=general \  
  --target=sample-app \  
  --image=quay.io/embik/dlv:v1.20.1 \  
  sample-app-[...] \  
  -- dlv --listen=127.0.0.1:2345 --headless=true  
      --accept-multiclient --api-version=2 attach 1
```



```
$ dlv connect localhost:2345
```

# Remote Debugging in VS Code

*... or any editor with DAP plugin  
(neovim, anyone?)*

## .vscode/launch.json

```
{  
  "name": "Remote Attach",  
  "type": "go",  
  "request": "attach",  
  "debugAdapter": "dlv-dap",  
  "mode": "remote",  
  "substitutePath": [  
    { "from": "${workspaceFolder}", "to": "/build" }  
  ],  
  "port": 2345,  
  "host": "127.0.0.1"  
}
```

# Application Dockerfile

```
FROM golang:1.19.8 as builder
```

```
WORKDIR /build
```

```
COPY . .
```

```
RUN CGO_ENABLED=0 go build -gcflags="-N -l" -o app .
```

```
FROM alpine:3.16
```

```
LABEL maintainer="marvin@kubermatic.com"
```

```
COPY --from=builder /build/app /bin/app
```

```
ENTRYPOINT ["/bin/app"]
```





KUBERNATIC

Let's See This in Practice

# Quick Live Demo

```
former coreinformers.PodInformer{
    record.NewBroadcaster()
    logging(klog.Infof)
    tRecordingToSink(&v1core.EventRecorder{
        && kubeClient.CoreV1().RESTClient,
        registerMetricAndTrackRateLimit
    })
}

controller{
    kubeClient,
    controller.RealPodController{
        client: kubeClient,
        recorder: eventBroadcaster.NewRecorder(
            klog, controller.NewControllerExp
            workqueue.NewNamedRateLimit
            eventBroadcaster.NewRecorder
        )
    }
}

former.Informer().AddEventHandler(cache
    AddFunc: func(obj interface{}) {
        jm.enqueueController(obj, true)
    },
    UpdateFunc: jm.updateJob,
    DeleteFunc: func(obj interface{}) {
        jm.enqueueController(obj, true)
    },
}
```



VARIABLES

Locals

> w: net/http.ResponseWriter(\*net/http.respon...  
> req: \*net/http.Request {Method: "GET", URL:...  
response: "bad-override"

WATCH

CALL STACK

- > [Go 3] runtime.gopark PAUSED
- > [Go 4] runtime.gopark PAUSED
- > [Go 5] runtime.gopark PAUSED
- > \* [Go 8] main.appHan... PAUSED ON BREAKPOINT

main.appHandler	main.go	13
net/http.HandlerFunc.ServeHTTP	server.go	
net/http.(*ServeMux).ServeHTTP	server.go	

BREAKPOINTS

main.go > appHandler

```
1 package main
2
3 import (
4     "fmt"
5     "net/http"
6     "os"
7 )
8
9 func appHandler(w http.ResponseWriter, req *http.Request) {
10     response := "this is a test"
11     response = "bad-override"
12
13     fmt.Fprintf(w, "%s\n", response)
14 }
15
16 func main() {
17     http.HandleFunc("/app", appHandler)
18
19     if err := http.ListenAndServe(":8080", nil); err != nil {
20         fmt.Printf("error serving webserver: %s", err.Error())
21         os.Exit(1)
22     }
23 }
```

Type 'dlv help' for list of commands.

Locals:

```
▶ w net/http.ResponseWriter = net/http.  
▶ req *net/http.Request = *net/http.Req  
  response string = "bad-override"
```

main.go:

```
13 fmt.Fprintf(w, "%s\n", response)
```

\* [Go 33] main.appHandler (Thread 10):

```
main.appHandler main.go:13  
net/http.HandlerFunc.ServeHTTP server.g  
net/http.(*ServeMux).ServeHTTP server.g  
net/http.serverHandler.ServeHTTP server  
net/http.(*conn).serve server.go:1991  
net/http.(*Server).Serve.func3 server.g
```

No Expressions

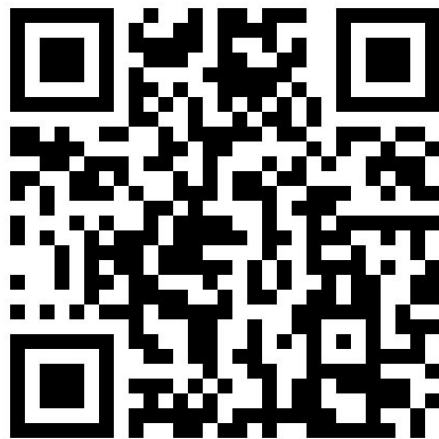
```
6 | "os"  
7 | )  
8 |  
9 | func appHandler(w http.ResponseWriter, req *http.Request) {  
10 |     response := "this is a test"  
11 |     response = "bad-override"  
12 |  
13 |     fmt.Fprintf(w, "%s\n", response)  
14 | }  
15 |  
16 | func main() {  
17 |     http.HandleFunc("/app", appHandler)  
18 |  
19 |     if err := http.ListenAndServe(":8080", nil); err != nil {  
20 |         fmt.Printf("error serving webserver: %s", err.Error())  
21 |         os.Exit(1)  
22 |     }  
23 | }
```

▶ ⚡ ↺ ⬆ ⬇ ⬅ ⬅

Type 'dlv help' for list of commands.

**\$ git log**

# Thank You!



GitHub Repository 