Debugging Kubernetes Controllers from IDE
Surendhar Ravichandran
Sr. Software Engineer, F5 Networks
• Background
• Short Introduction to Debugger
• Patterns to debug kubernetes controllers
• Demo
• Summary
• Started working on controllers six months ago
• Extensively used print statements
• People get used to it and move forward
• “New Hires” experience is bad
What’s wrong with print statements

- Seems simple at the beginning but silently eats developers time
- Not reliable
- Lacks big picture
- Very little information
- Extremely difficult to understand multi threaded applications
- Only works forward in time
- If not reviewed properly, it will end up in production code
The Debugger
What is a debugger anyway?

• Debugger is equivalent of Dr. Strange’s time stone for Developers
• It allows you to stop a program execution, move forward and backward in time.
• You will have access to all the information about the program sequenced
Debugger Model

Debugger

App

TCP
Patterns
Think out of the box

- Controller doesn’t need to run inside a container
- All we need is a connection to the Kubernetes API
- Client-go out-of-cluster configuration example
- Elevated access in your local operating system
- `kubectl` set context
- Controller code fetches Kubernetes config from home directory and communicate with the API automatically
- 20% of work; 80% of productivity
Demo
In-Cluster Controllers
Inside the kubernetes

- You may need to debug in customer’s kubernetes
- Your controller needs some resources running inside kubernetes
- Controller communication with Kubernetes API is automatic
- Package Debugger inside your controller image
- Expose Debugger TCP port
Adding Debugger Support

Production Dockerfile

FROM golang AS builder
ENV CGO_ENABLED 0
ADD ./go/src/controller
RUN go get k8s.io/apimachinery/pkg/api/errors &&
    go get k8s.io/apimachinery/pkg/apis/meta/v1 &&
    go get k8s.io/client-go/kubernetes &&
    go get k8s.io/client-go/rest
RUN go build -o /controller controller

FROM alpine AS runner
WORKDIR /
COPY --from=builder /controller /
ENTRYPOINT ["/controller"]

Debug Dockerfile

FROM golang AS builder
ENV CGO_ENABLED 0
ADD ./go/src/controller
RUN go get k8s.io/apimachinery/pkg/api/errors &&
    go get k8s.io/apimachinery/pkg/apis/meta/v1 &&
    go get k8s.io/client-go/kubernetes &&
    go get k8s.io/client-go/rest
RUN go get github.com/go-delve/delve/cmd/dlv
RUN go build -gcflags "all=-N -l" -o /controller controller

FROM alpine AS runner
WORKDIR /
COPY --from=builder /controller /
COPY --from=builder /go/bin/dlv /
ENTRYPOINT ["/dlv",
    "--listen=:40000",
    "--headless=true",
    "--api-version=2",
    "--accept-multiclient",
    "exec", "/controller"
EXPOSE 40000
Exposée Debugger Port

```bash
kubectl run controller-debug --image=ssurenr/samplecontroller:debug
kubectl expose deployment controller-debug --type="NodePort" --port 40000
```
Adding Debugger Support

.vscode/launch.json

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "K8SDebug",
      "type": "go",
      "request": "launch",
      "mode": "remote",
      "program": "${fileDirname}",
      "port": 32393,
      "host": "192.168.99.100",
      "env": {},
      "args": []
    }
  ]
}
```
Demo
Useful networking tips

- Proxies
  - SSH Proxy
  - Kubectl proxy
  - Telepresence
- VPN
- VxLAN
Gotchas

• Some debuggers require you to build the controller from inside the go path to work correctly
• In some IDE, communicating over a remote host is a premium feature
• Create a different work flow for debugging in your CI/CD Build process
  • Create a separate Make file action
  • Keep the debugging Docker file separate
Summary

• Avoid using printf debugging
• Use a debugger
• Keep controller outside the k8s cluster for debugging
• Create a workflow for in-cluster controller
• Keep your production and debugging controllers separate
• Multiple ways to overcome networking issues
Links

https://github.com/ssurenr/controller

https://github.com/kubernetes/client-go
Thank you
Surendhar

“Happy Controlling”