Kubernetes & Golang

A retrospective

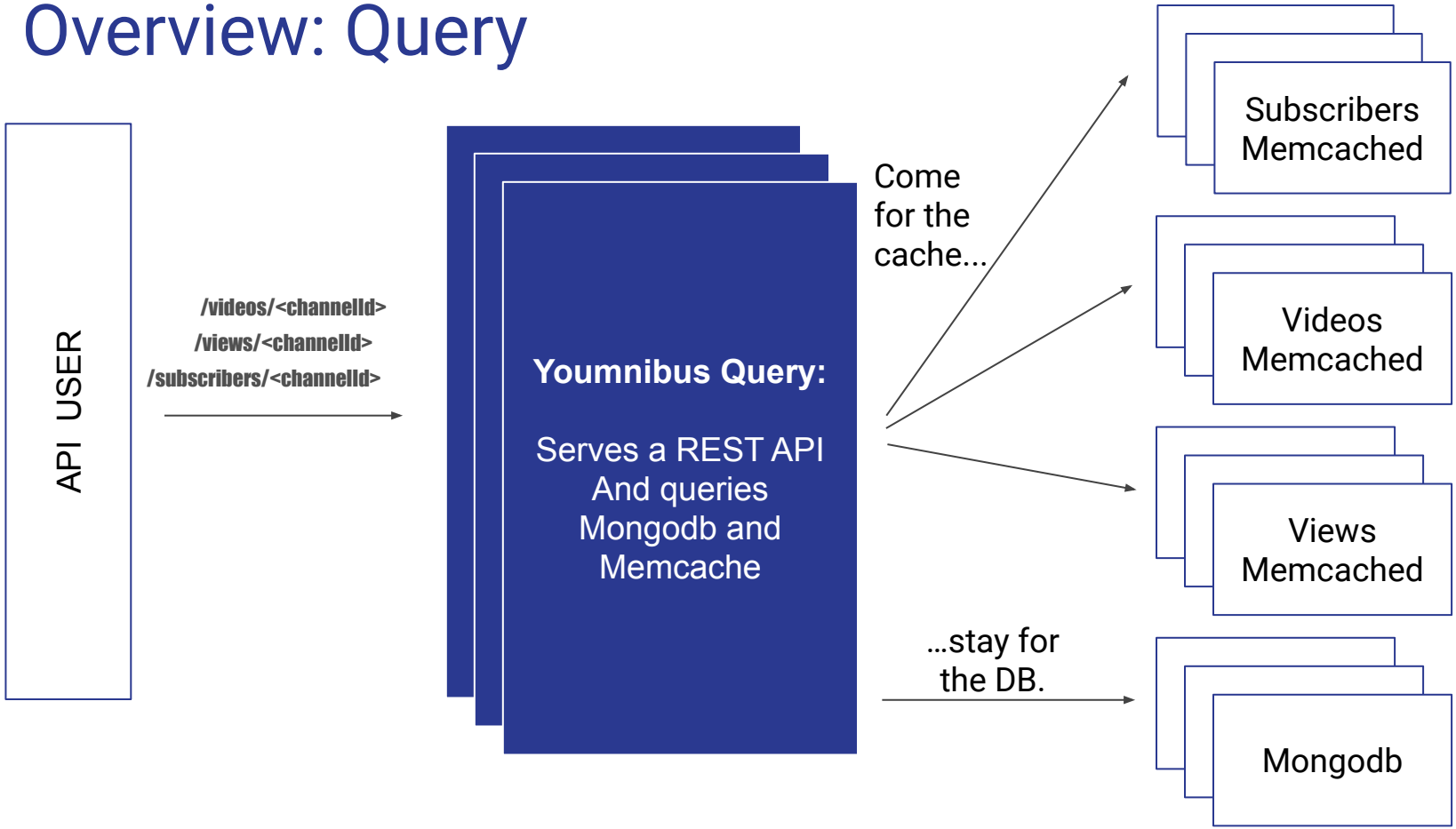
My project

Youmnibus = YouTube + Omnibus. It is a system for capturing and querying time series YouTube data.

The stack:

- YouTube Data API
- Google Kubernetes Engine
- Mongodb
- RabbitMQ
- Golang
- Memcached

Overview: Query



Overview: Query

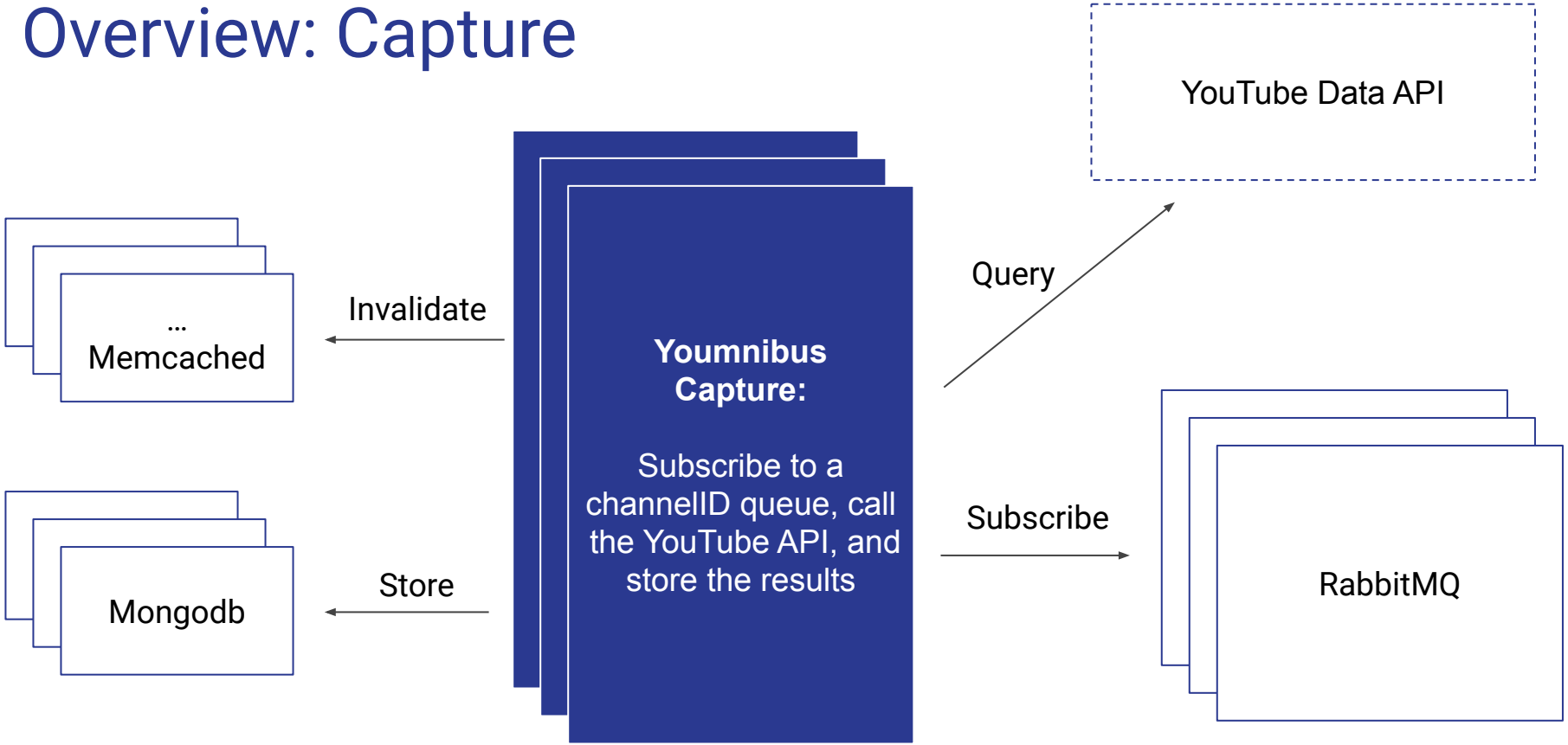
Quirks:

- JSON serialization with base Golang kinda sucks

```
type SubscribersAt struct {
    SubscriberCount uint64 `json:"subscriberCount,omitempty,string"`
    At              string `json:"at,omitempty"`
}
```

- I use Gin as a REST framework. It’s meant to be a relatively lightweight and high performance router. You can plug in middleware for e.g. logging and authentication - I’ve just used the defaults.

Overview: Capture

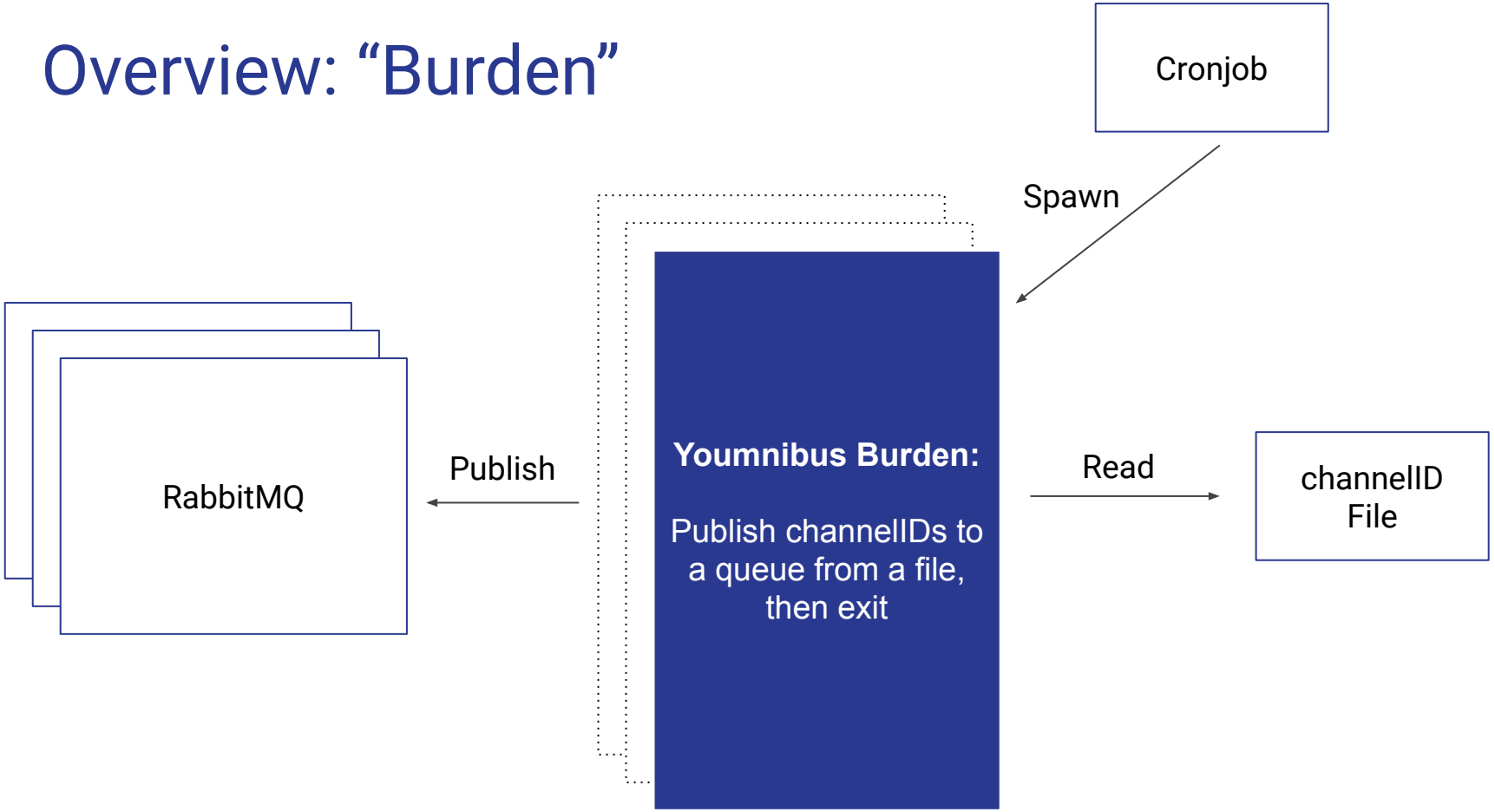


Overview: Capture

Quirks:

- I authenticate and authorize with the YouTube API using an API key. If your system is querying/updating more personal user info, then there is an Oauth option.
- You can create all the RabbitMQ needed resources from the Golang client itself. I.e. A queue and a dead letter exchange. The first application to request the queue connection will construct it, the rest will just use it.
- Golang concurrency + RabbitMQ worker queues work really nicely - Golang subscribes to a (Golang) channel which is populated asynchronously. It’s reliable and the channel doesn’t use any resources while idle.

Overview: “Burden”



Overview: Burden

Quirks:

- Because Golang applications are statically linked (to C libs) they work great for worker applications: Golang apps start up quickly and use little base memory.
  - You can also compile *with* the C libs, so that the binary has no dependencies whatsoever.

Environment Evolution

Local

“I'm too young to die”

Get the applications running in separate terminals.

Docker

“Hey, not too rough”

Get the applications running in Docker, and create a build pipeline using Docker Hub and GitHub.

Kubernetes

“Hurt me plenty”

Deploy load balanced applications on Google Kubernetes Engine with volume claims, secrets and config maps.

Local

Quirks:

- I configure my applications using environment variables which have as many common sense defaults as possible, e.g. MONGO_PORT: 27017. This makes it easy to run the application locally - by just running the executable - but also easy to configure the app in both Docker and Kubernetes.
- I find it is helpful to log the config at the start of the application, it serves as a kind of minimal manual about how it can be configured.

Docker

Quirks:

- A Makefile is a great way of automating common tasks for your application, including:
  - Building and installing the application locally
  - Building / Deploying / Pushing docker images
  - ...and anything else which can be run in a few lines in the terminal.
  - Plus it has bash completion

Docker

Quirks:

- E.g. make install docker-build docker-push

  Is equivalent to:

```
go build ./...
go install ./...
docker build -t youmnibus-burden -f docker/Dockerfile.youmnibus-burden .
docker build -t youmnibus-capture -f docker/Dockerfile.youmnibus-capture .
docker build -t youmnibus-query -f docker/Dockerfile.youmnibus-query .
docker tag youmnibus-burden:latest lpulles/youmnibus-burden:0.1
docker tag youmnibus-capture:latest lpulles/youmnibus-capture:0.1
docker tag youmnibus-query:latest lpulles/youmnibus-query:0.1
docker push lpulles/youmnibus-burden:0.1
docker push lpulles/youmnibus-capture:0.1
docker push lpulles/youmnibus-query:0.1
```

Docker

Here’s a nice Dockerfile for Golang apps:

```
FROM golang:alpine
RUN apk update && apk add --no-cache git

ENV GOPATH /go
ENV GOBIN /go/bin
ENV CGO_ENABLED 0
WORKDIR /app
RUN mkdir -p /go/bin

COPY ./go.mod .
RUN go mod download
RUN go mod verify

COPY . .
RUN go build ./...
RUN go install ./...

FROM scratch
COPY --from=0 /go/bin/youmnibus-burden /go/bin/youmnibus-burden
ENTRYPOINT ["/go/bin/youmnibus-burden"]
```

Docker

Quirks:

- Some aspects of the minimal image to consider
  - The resulting image is about ~25MB large
  - You can’t run a terminal inside the container, so you can’t list files or check what environment variables the container is currently using. This makes it difficult to debug, though also harder to attack.
  - The benefits/pitfalls of this approach don’t really relate to the size of the image, because if I instead used e.g an ubuntu image, I would only download the application layer of the image, and would only hold in memory the application layer while running the container.
  - So, do this at your discretion.

Kubernetes

Quirks:

- Constructing secrets for strings (e.g. passwords) can be deceptively tricky.
  - To construct a secret for a password, you need to provide a base64 encoded version of the string.
  - For those with some bash experience, this might seem trivial e.g. we just run
    echo “duck” | base64
  - The trick is, the echo command inserts a newline character at the end of the string, which changes the actual password.
  - You need:
    echo **-n** “duck” | base64
  - This caused many hours of frustration.

Kubernetes

Quirks:

- Connecting to the instances of a stateful set can be deceptively tricky.
  - A stateful set is a deployment of an application where the instances aren't load balanced and are not automatically killed for new deployments. E.g. a mongodb deployment.
  - For mongodb, you can use a connection string in your apps like mongo://user:pass@host1,host2,host3 where the client will select a host accordingly. So what DNS address do we give the application for host1, host2 and host3?
  - It's NOT mongod-0, mongod-1 and mongod-2 i.e. the names of the individual pods
  - It's NOT given by an environment variable - unlike normal services which are.
  - What it actually is, is mongod-0.mongodb-service.youmnibus.svc.cluster.local, mongod-1.mongodb-service.youmnibus.svc.cluster.local, mongod-2.mongodb-service.youmnibus.svc.cluster.local (where youmnibus is the kubernetes namespace I'm using).
  - This caused many hours of frustration.

Kubernetes

Quirks:

- The initial stages of setting up Kubernetes is in general deceptively tricky and causes many hours of frustration.
- BUT, when it finally works, it’s awesome!

Demo

https://github.com/liampulles/youmnibus