Introduction to Watermill
Simple Go Event-Driven application in 20 minutes

FOSDEM 2023
Robert Laszczak

roblaszczak
Introduction to Watermill

Simple Go Event-Driven application in 2015 minutes

FOSDEM 2023
Robert Laszczak

roblassczak
Hey, I’m Robert

- Principal Engineer at /id
- Blogging at threedots.tech
- @roblaszczak at 🥇 🦸
- @roblaszczak.techsub.social at 🕵️‍♂️
- robert [at] threedotslabs.com
- Author of Watermill
PROBLEM?
BUILDING EVENT-DRIVEN APPLICATIONS IS NOT EASY
Event driven concepts

- Consumer groups
- Partitioning
- Message ordering
- At-least-once delivery
- Message Ack and Nack support
- Poison queues
- Not losing any message
How to make building event-driven applications easy in Go?
12

Officially Supported Pub/Subs
HTTP concepts that you usually don’t think about

- TLS
- 7 layers of network stack
- Connection pooling
- Redirect handling
- Handling different protocol versions
- Handling network transient problem
Watermill is not a framework

Watermill is a library
HOW?
Unix philosophy (1978)

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams message, because that is a universal interface.
type Message struct {
    UUID string
    Metadata map[string]string
    Payload []byte
}
type Publisher interface {

    Publish(topic string, messages ...*Message) error

    Close() error

}

type Subscriber interface {

    Subscribe(ctx context.Context, topic string) (<-chan *Message, error)

    Close() error

}
type HandlerFunc func(msg *Message) ([][]*Message, error)
Middlewares

- Timeout
- Instant Ack
- Correlation
- Poison
- Retry

- Throttle
- Random fail
- Duplicator
- Recoverer
- Ignore Errors
Testing
func TestPublishSubscribe(t *testing.T) {
    features := tests.Features{
        ConsumerGroups: true,
        ExactlyOnceDelivery: false,
        GuaranteedOrder: false,
        Persistent: true,
    }
}

tests.TestPubSub(
    t,
    features,
    createPubSub,
    createPubSubWithConsumerGrup,
)
var stressTestTestsCount = 10

func TestPubSubStressTest(
    t *testing.T,
    features Features,
    pubSubConstructor PubSubConstructor,
    consumerGroupPubSubConstructor ConsumerGroupPubSubConstructor,
) {
    for i := 0; i < stressTestTestsCount; i++ {
        t.Run(fmt.Sprintf("%d", i), func(t *testing.T) {
            t.Parallel()
            TestPubSub(t, features, pubSubConstructor, consumerGroupPubSubConstructor)
        })
    }
}
How fast is Watermill?
<table>
<thead>
<tr>
<th>Pub/Sub</th>
<th>Publish (messages / s)*</th>
<th>Subscribe (messages / s)*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kafka</td>
<td>70,252</td>
<td>117,529</td>
</tr>
<tr>
<td>NATS</td>
<td>76,208</td>
<td>38,169</td>
</tr>
<tr>
<td>SQL (MySQL)</td>
<td>6,989</td>
<td>143</td>
</tr>
<tr>
<td>Google Cloud Pub/Sub</td>
<td>7,416</td>
<td>39,591</td>
</tr>
<tr>
<td>AMQP</td>
<td>2,408</td>
<td>10,608</td>
</tr>
</tbody>
</table>

*1 instance, 1 process, Docker Compose
The first rule of live coding
Don’t do livecoding

The first rule of live coding
Not covered

- It’s hard to create production grade app in 15 minutes :) (even HTTP based)
- Kafka and Google Cloud Pub/Sub internals
- At-least-once delivery
- CQRS component
That’s a lot!

Where I should start?
Documentation

- Getting started
  Watermill up and running

- Message
  Message is one of core parts of Watermill

- Pub/Sub
  Publishers and Subscribers

- Message Router
  The Magic Glue of Watermill

- CQRS Component
  Command Query Responsibility Segregation (CQRS) Component

- Implementing custom Pub/Sub
  Bring Your Own Pub/Sub

- Metrics
  Monitor Watermill in realtime

- Middlewares
  Add generic functionalities to your handlers in an inobtrusive way

- Troubleshooting
  When something goes wrong
Thanks for all contributors!
Watermill v1.2 🎉

Online Release party: March 1st
THANKS!

https://threedots.tech/fosdem2023/