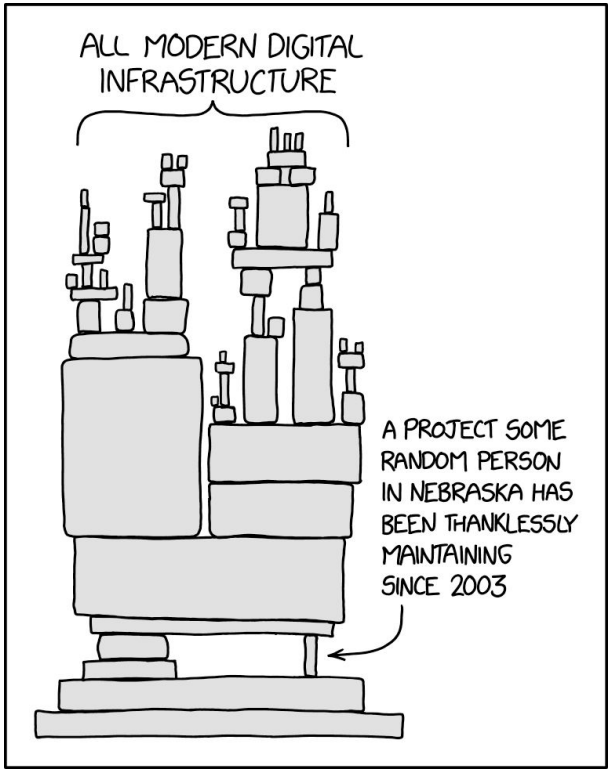
Data engineering at scale: Real-world case studies with Benthos

Mihai Todor
Jan 19th, 2024

About me

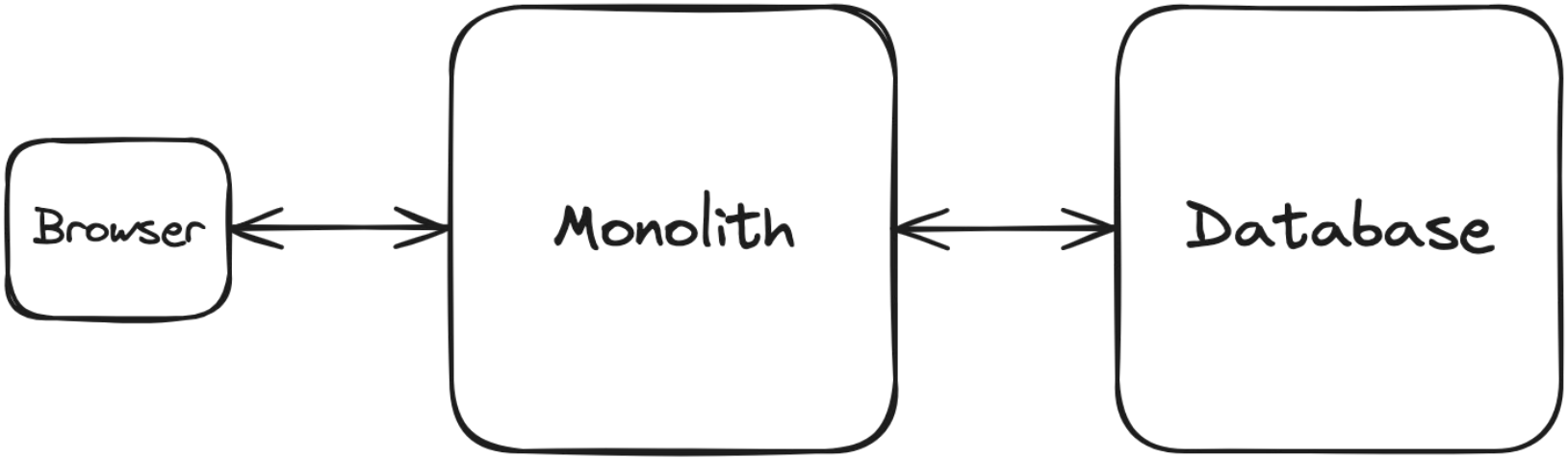
- Principal Software Engineer, currently freelancing
- 16+ years of experience in industry and academia
- Backend microservices and infrastructure / scalability / public cloud SaaS
- Golang, Python, C++
- Open Source contributor
- https://www.linkedin.com/in/mtodor
- https://github.com/mihaitodor

What do SaaS platforms look like behind the scenes?



Source: https://xkcd.com/2347

Let’s build a LinkedIn clone

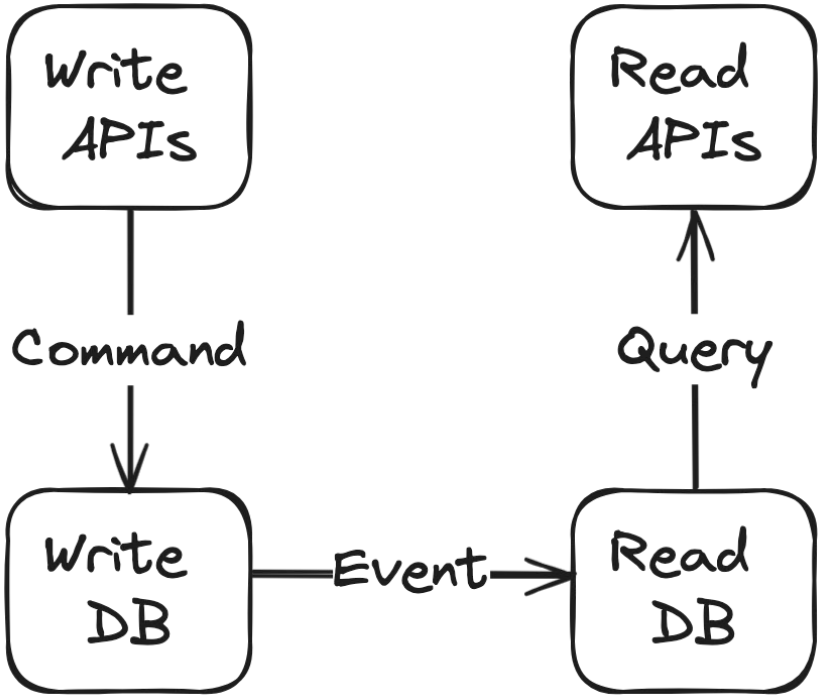


Divide the LinkedIn clone monolith into microservices

- Product feature services
  - Timeline
  - Users
  - Interactions
  - Messaging
  - Payments
  - Ads
  - …

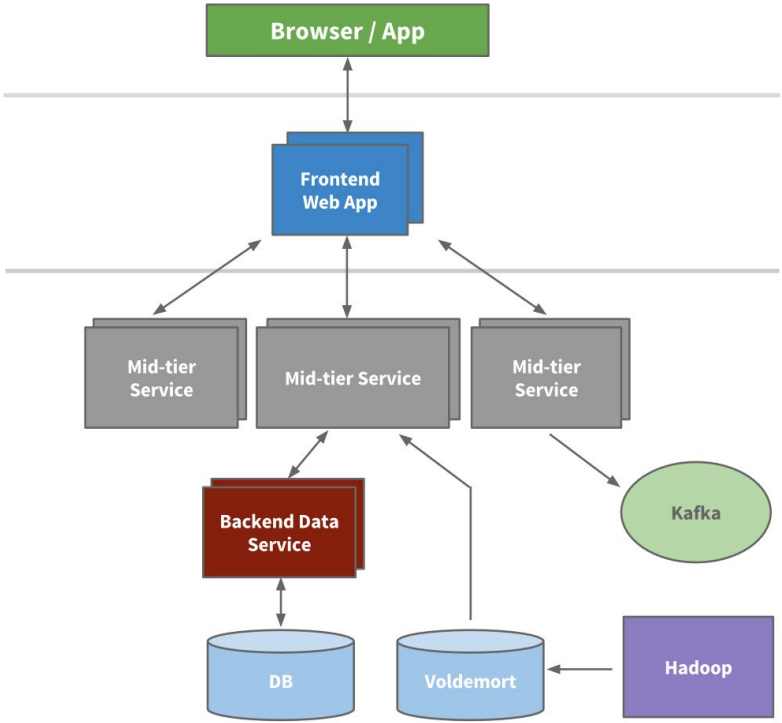
- Auxiliary services
  - Schedulers and orchestrators
  - Databases
  - Storage
  - Analytics
  - Audit log
  - APIs
  - …

CQRS - Command query responsibility segregation



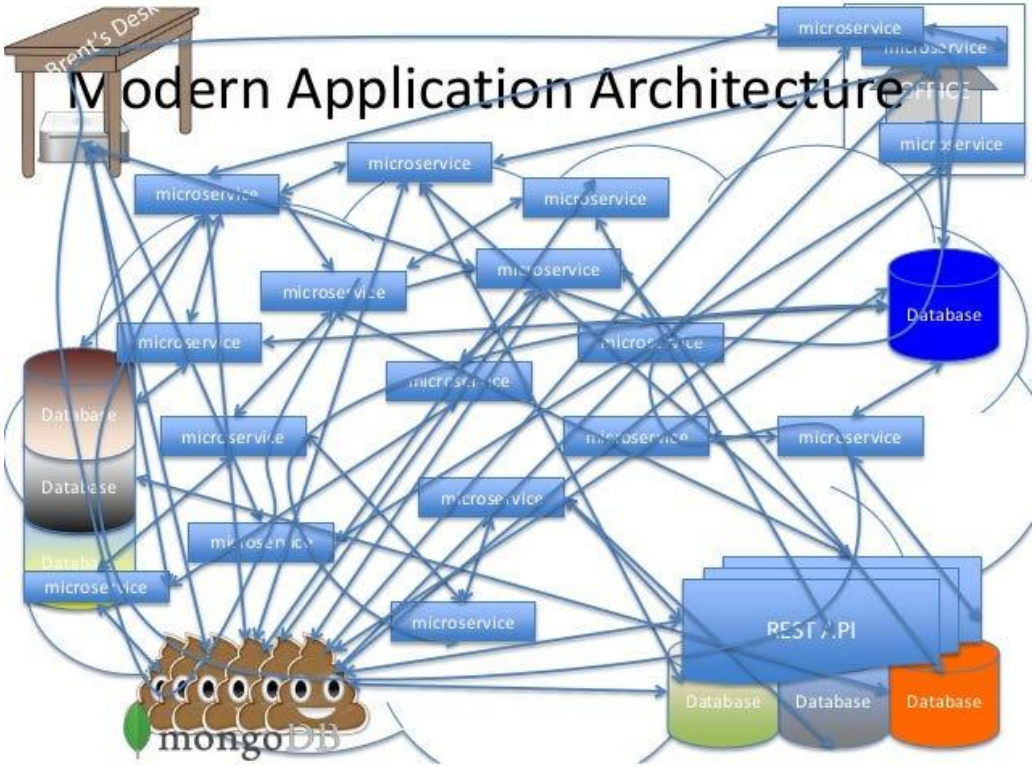
Details: https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/cqrs-pattern.html

LinkedIn’s high level system architecture (2015)



Source: https://engineering.linkedin.com/architecture/brief-history-scaling-linkedin

LinkedIn clone microservice architecture



Source: https://www.slideshare.net/danveloper/microservices-the-right-way

What else?

- Code repositories
- CI/CD (Continuous integration and continuous delivery/deployment)
- Monitoring and alerting
- Container image registry
- CDNs (content delivery network)
- VPCs & Firewalls
- …

Users JSON REST API service for our LinkedIn clone

- Use favourite programming language
- Import HTTP libraries
- Import the JSON libraries
- Import the database client libraries
- Write a bunch of glue code
- Build a Docker image
- Run it as a Docker container in Kubernetes

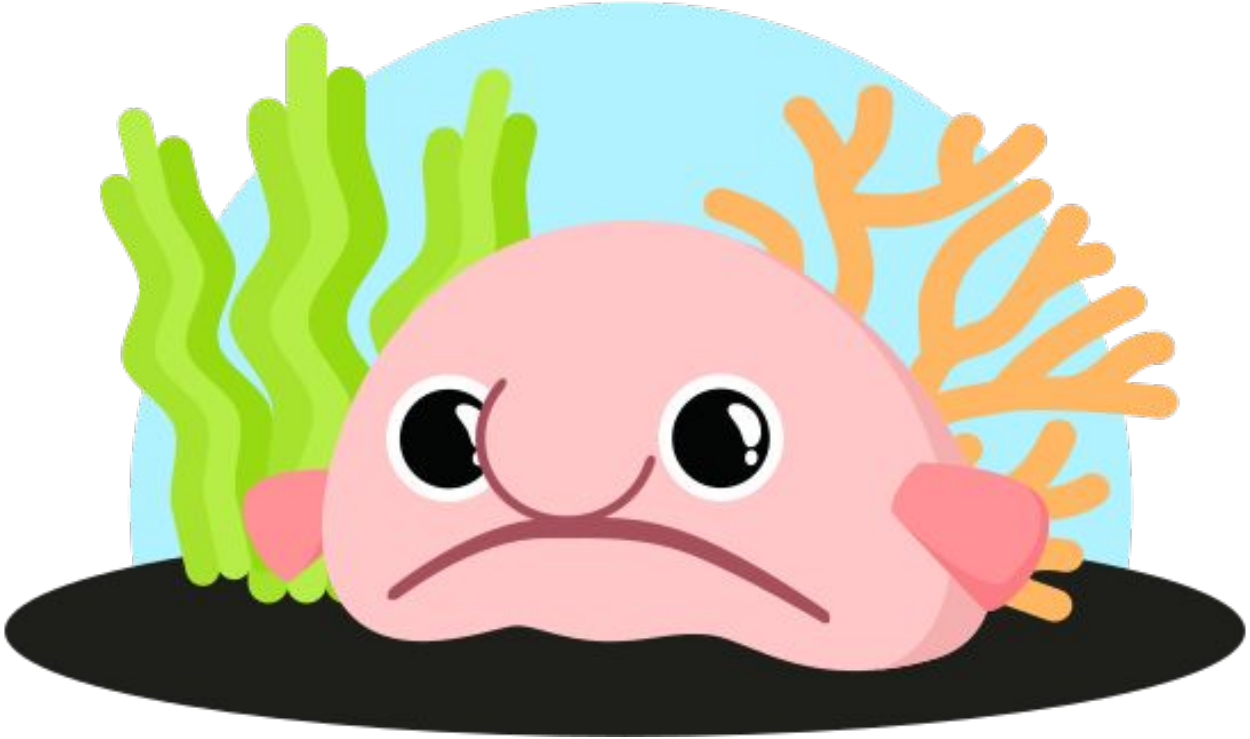
What’s missing?

- Resource limits (memory, CPU, disk and network on individual nodes aren’t infinite)
- Metrics
- Logging
- Retries
- Rate limiting and throttling
- Batching
- Caching
- Scaling (high availability, fault tolerance, etc)
- Security
- Extensibility and maintainability

Twelve-factor apps: https://12factor.net

1. Codebase
2. Dependencies
3. Config
4. Backing services
5. Build, release, run
6. Processes
7. Port binding
8. Concurrency
9. Disposability
10. Dev/prod parity
11. Logs
12. Admin processes

What is Benthos? https://www.benthos.dev

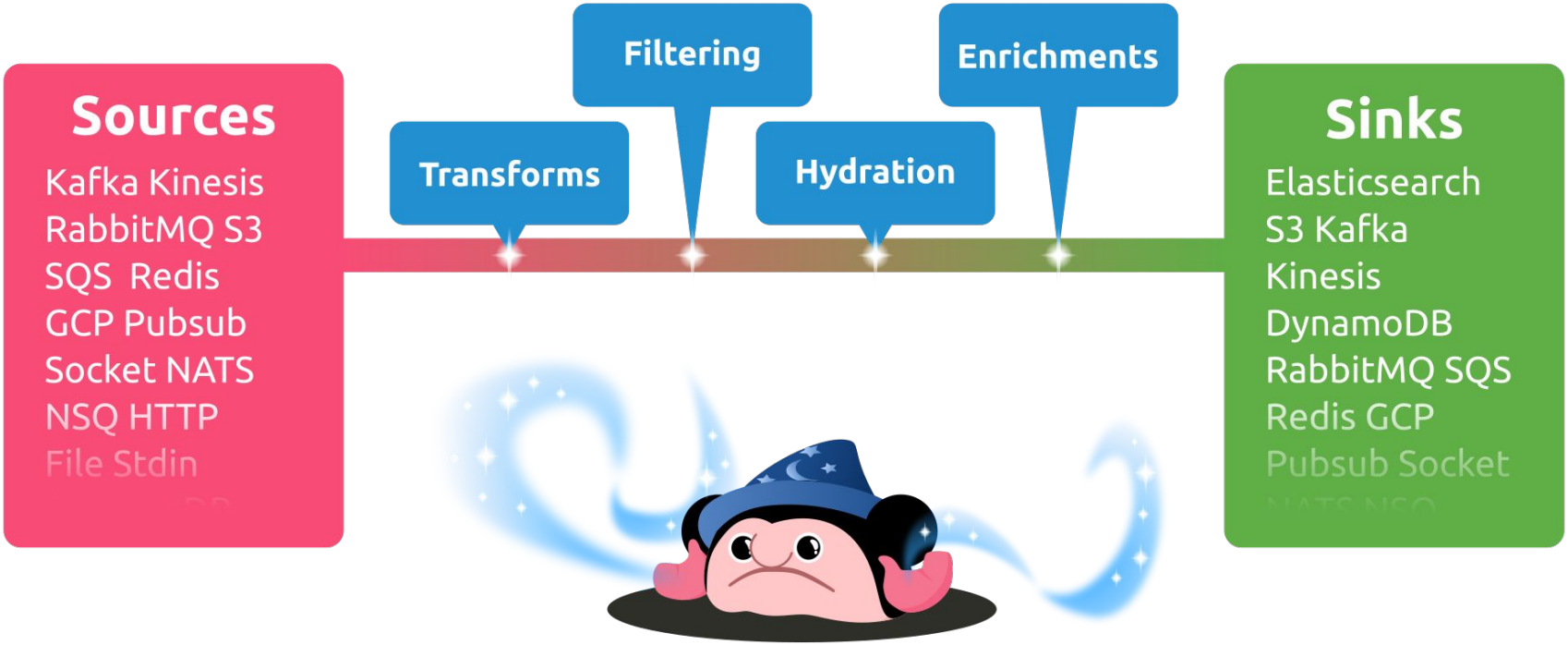


Hint: It has nothing to do with deep sea fish, except for the logo

Disclaimers

- Thoughts expressed here are my own
- Benthos is owned and maintained by Ash: https://twitter.com/Jeffail
- Ash designed and built Benthos from scratch
- I am a contributor to the project

What is Benthos for?



“Fancy stream processing made operationally mundane” - Ash Jeffs

Boringly easy to use

```
# Install
curl -Lsf https://sh.benthos.dev | bash

# Make a config
benthos create stdin/mapping/stdout > ./config.yaml

# Run
benthos -c ./config.yaml
```

Features

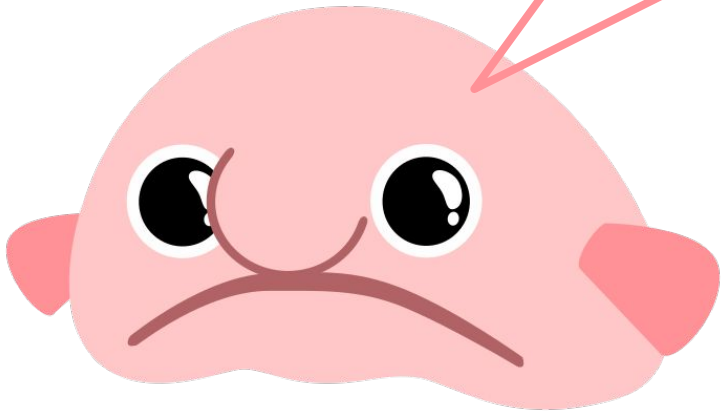
- Declarative YAML-based configuration
- Single message transforms
- Stateless
- At least once delivery
- Metrics and logging
- Custom Plugins
- Written in Go

Bloblang

Custom DSL for arbitrary data transforms

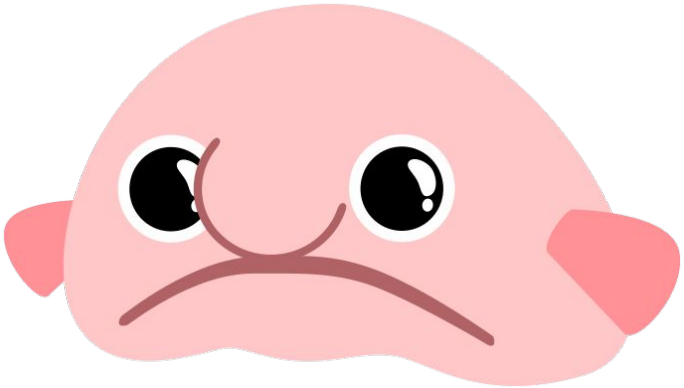
```
root.new_doc = match this.doc {
 this.type == "article" => this.article
 this.type == "comment" => this.comment
 _ => this
}
```

```
{
 "doc": {
   "type": "article",
   "article": {
     "id": "foo",
     "content": "qux"
   }
 }
}
```



```
{
 "new_doc": {
   "id": "foo",
   "content": "qux"
 }
}
```

Deployment models

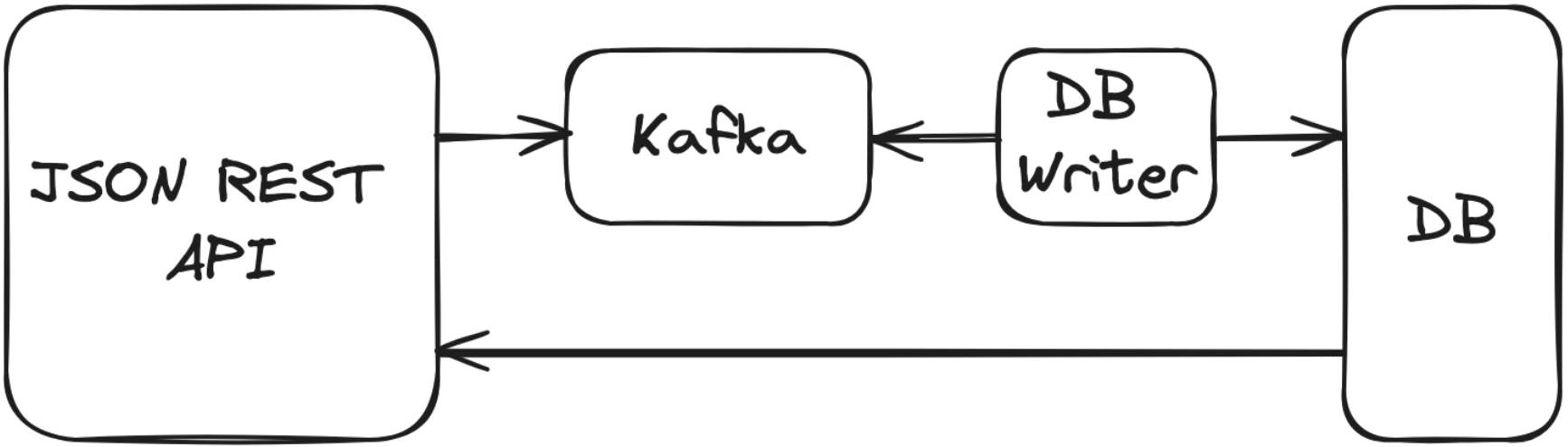


Standalone CLI app

Serverless

Kubernetes

Case study: JSON REST API

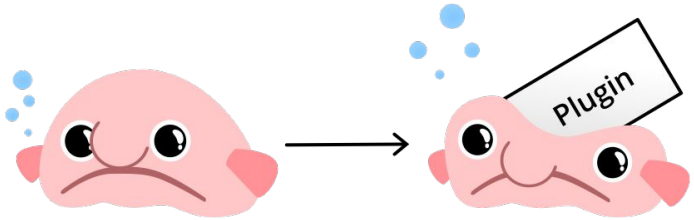


Importing Benthos as a library

```go
package main

import (
    "context"

    "github.com/benthosdev/benthos/v4/public/service"

    // Import minimal Benthos components
    _ "github.com/benthosdev/benthos/v4/public/components/io"
    _ "github.com/benthosdev/benthos/v4/public/components/pure"
)

func main() {
    service.RunCLI(context.Background())
}
```

Source: https://github.com/benthosdev/benthos-plugin-example

Writing a custom Benthos plugin




```go
type processor struct{}

func (r *processor) Process(ctx context.Context, m *service.Message) (service.MessageBatch, error) {
   println("foobar")
   return nil, nil
}

func (r *processor) Close(ctx context.Context) error { return nil }

func init() {
   _ = service.RegisterProcessor("foobar",
       service.NewConfigSpec(),
       func(conf *service.ParsedConfig, mgr *service.Resources) (service.Processor, error) {
           return &processor{}, nil
       },
   )
}
// > ./benthos create -s stdin/foobar/stdout > config.yaml
// > ./benthos -c config.yaml
```

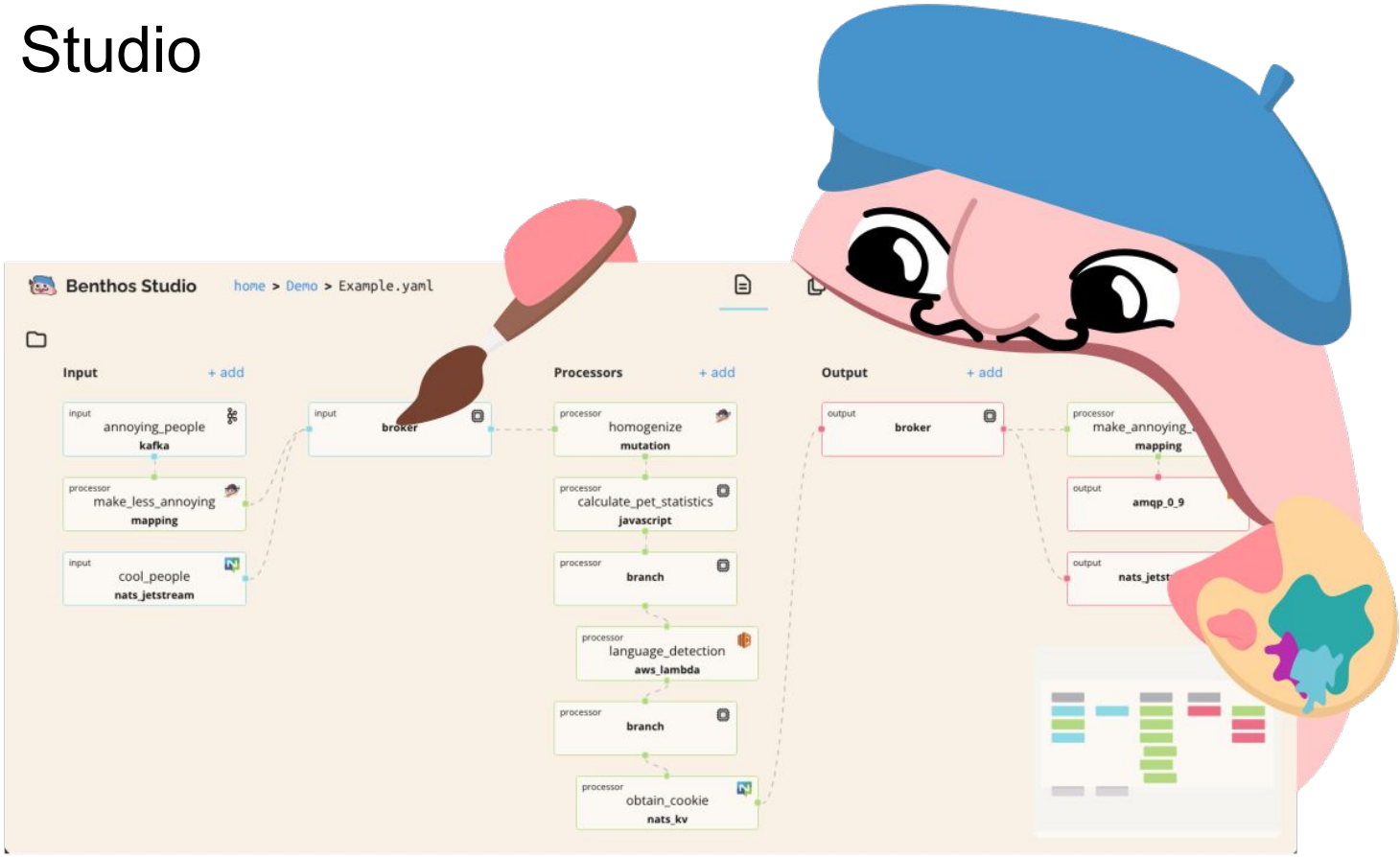
Future enhancements and goodies

More adapters

Custom plugins

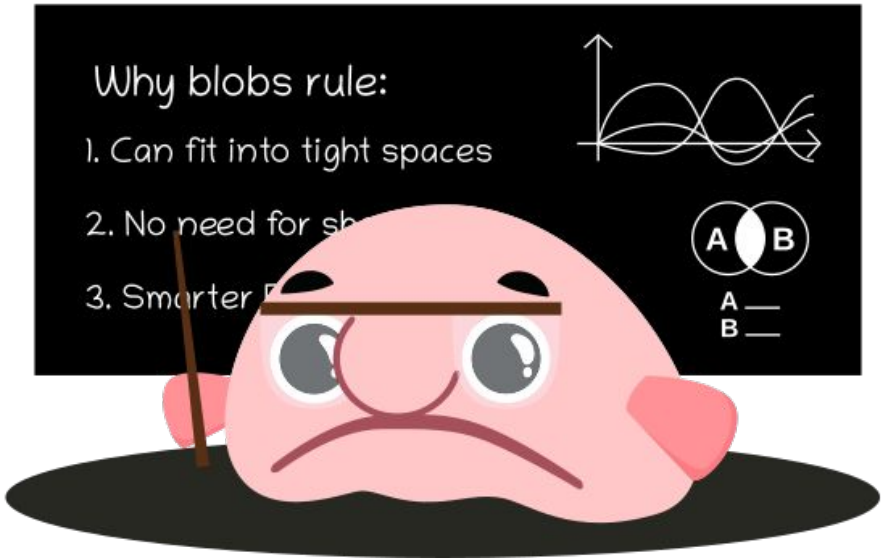
Benthos Studio

Benthos Studio



Why Open Source?

- Built on the shoulders of giants
- Community-driven features, enhancements and bug fixes
- High quality standards enforced uniformly
- Open issue tracker and permanent change history
- Avoids vendor-driven lock-in

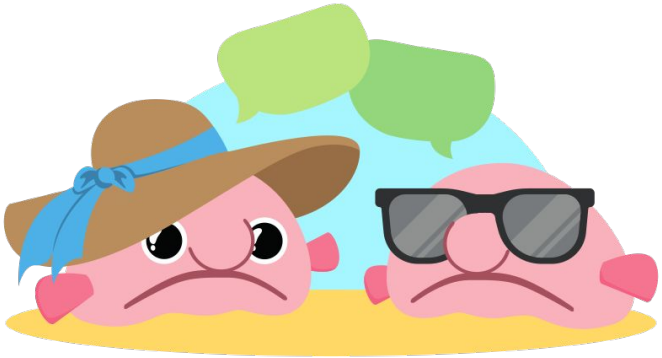


Open Source Needs You!

Community https://www.benthos.dev/community

Discord

https://discord.gg/6VaWjzP



slack

https://invite.slack.golangbridge.org

#benthos channel

Thank you!

Happy to answer your questions!