JUST EAT Takeaway.com

GopherCon UK 2025

15/08/2025

Hi

https://ainsley.dev

@ainsleydev

linkedin.com/in/ainsleyclark

github.com/ainsleyclark


luno

How Just Eat uses tooling to deploy Go microservices in minutes

JET Connect

Scaling Up

Demo

Going Further

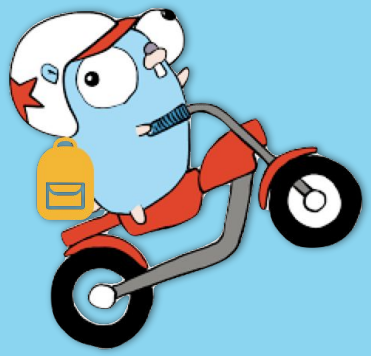
Results

JET Connect

What's JET Connect?

- Founded in 2013 as an integration platform.
- Unified order and menu processing.
- Partners are able to use one unified system.

POS
(Point of Sale)



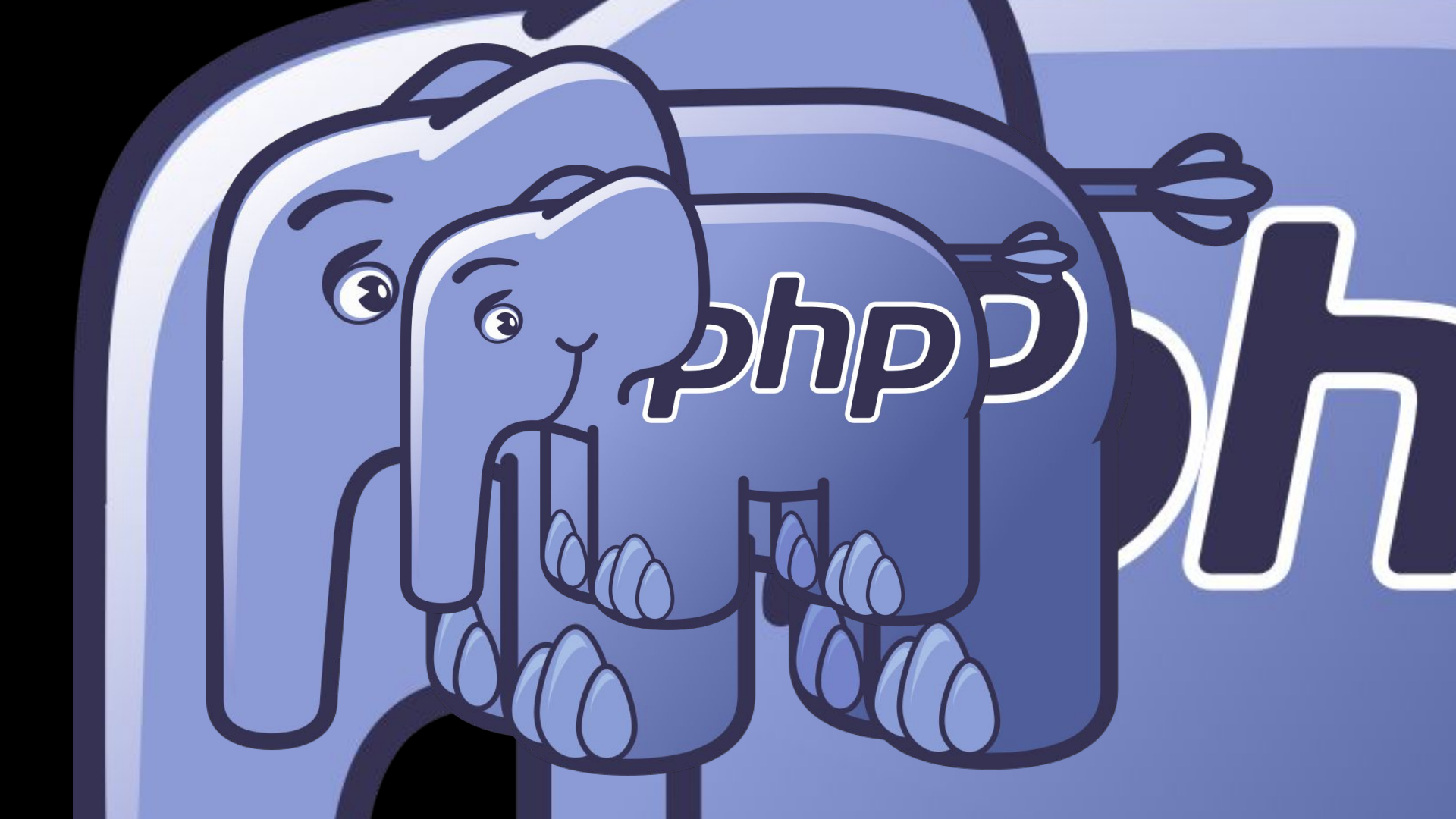
Just Eat 🧡 Go

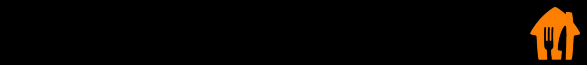
100+ Microservices

100+ Microservices

JET Connect

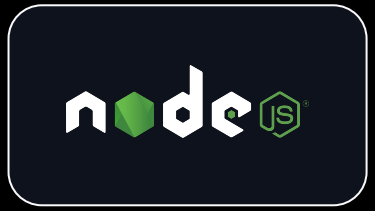
Scaling Up


php

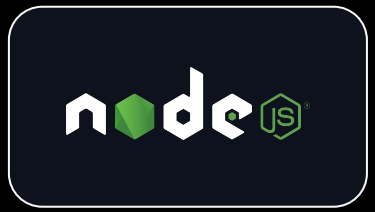


grpc

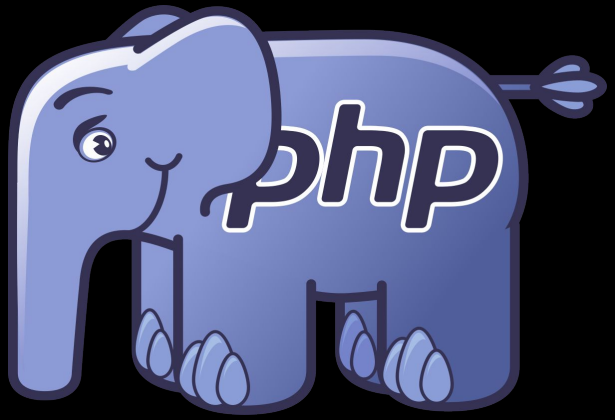
```
package ordering;

service OrderSender {
    rpc SendOrder(OrderRequest) returns (OrderResponse);
}
```



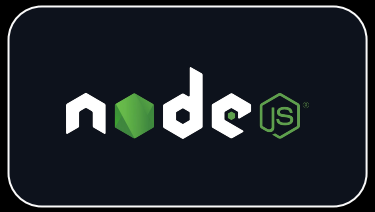
Dominos



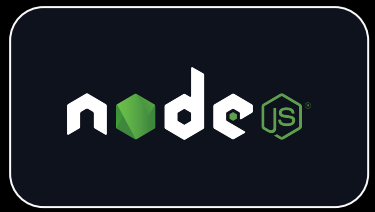
KFC



Monolith



McDonald’s



Subway

Problem Areas

- Maintenance
- Updates
- No E2E testing
- Needed a way to easily create & update services.

Switching To Go

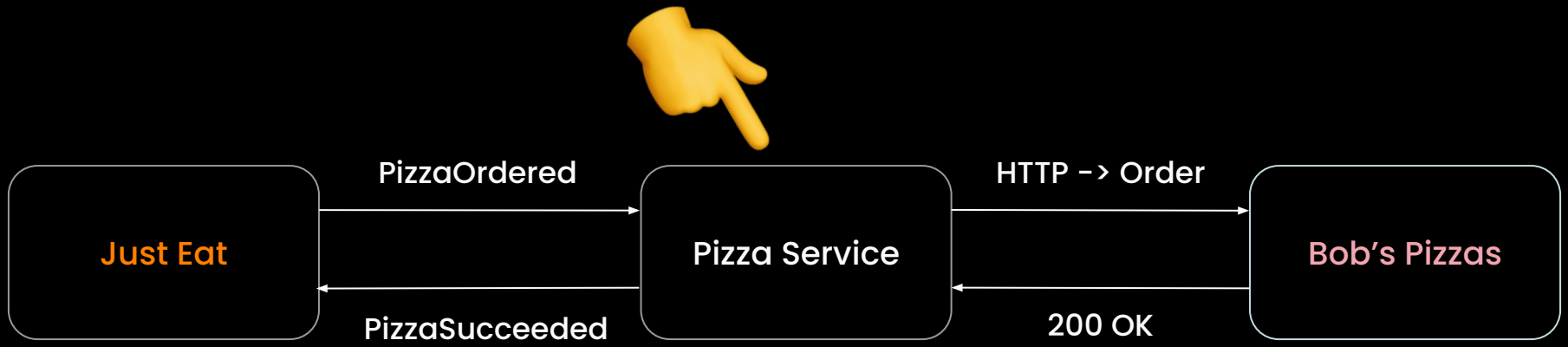
- Needed something that scaled.
- So we switched to Go.
- But needed to come up with a plan and prerequisites for a tool.

Prerequisites

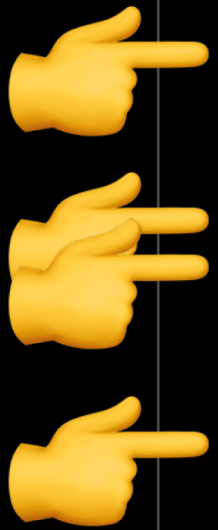
- Generate and update a service in seconds.
- Define HTTP APIs with industry standard docs.
- Subscribe to events with as little code as possible.
- Abstract as much of the infra as possible.
- All CI/CD auto generated so rolling out was easy.
- All changes to main deployed to production, safely (E2E).
- Developers can run services locally with ease.
- Patterns that are used frequently abstracted.


GoKit
Microservice Development Tooling

Demo


Just Eat
PizzaOrdered
PizzaSucceeded
Pizza Service
HTTP -> Order
200 OK
Bob's Pizzas

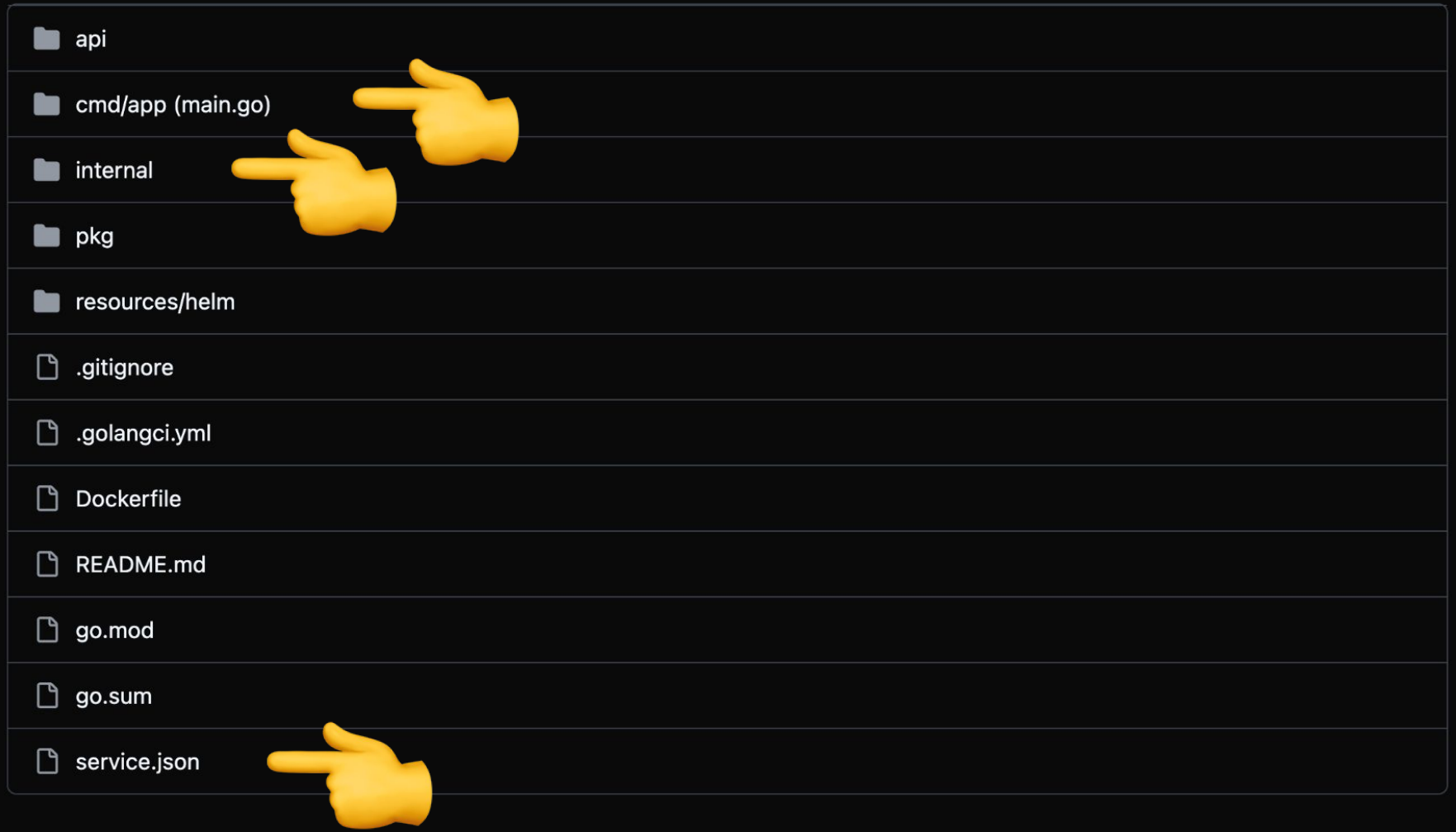

go-kit.sh
go-kit new pizza-service

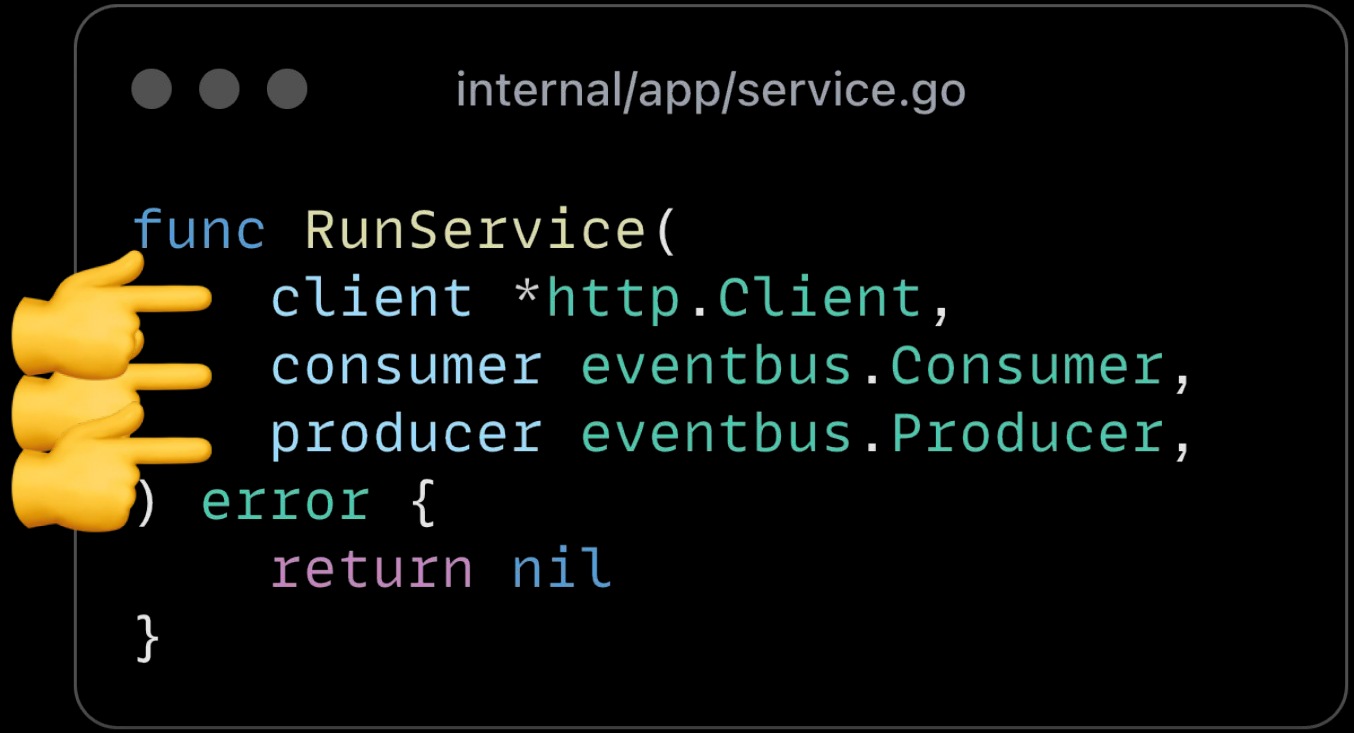


internal/app/service.go

What APIs will your service produce or consume?

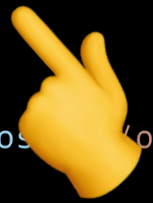
```
  [ ]  HTTPS
  [x]  Events: Consume events
  [x]  Events: Produce events
  [ ]  Database
  [ ]  Object Storage
> [x]  HTTP Client
```


api
cmd/app (main.go)
internal
pkg
resources/helm
.gitignore
.golangci.yml
Dockerfile
README.md
go.mod
go.sum
service.json


internal/app/service.go
func RunService(
client *http.Client,
consumer eventbus.Consumer,
producer eventbus.Producer,
) error {
return nil
}

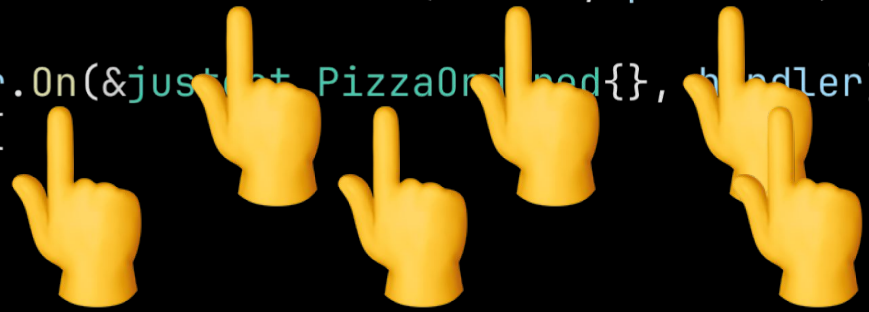
internal/handlers/pizza-ordererd.go

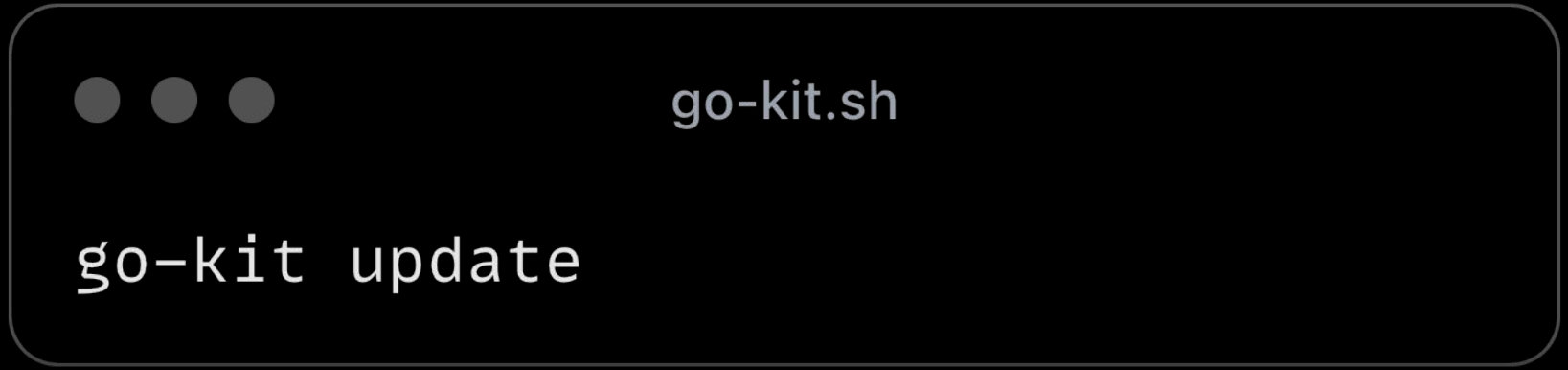
```go
func PizzaHandler(
    client *http.Client,
    producer eventbus.Produce
) eventbus.EventHandler {
    return func(ctx context.Context, event any, headers ...eventbus.Header) error {
        ev := event.(*justeat.PizzaOrdered)

        // Tell Partner about Pizza...
        order := Order{}
        err := client.Post(http.MethodPos    "/order", bytes.NewBuffer(order.Bytes()))
        if err != nil {
            return err
        }

        producer.Emit(ctx, &justeat.PizzaSucceeded{
            RestaurantID: "1234",
        })
    }
}
```

internal/app/service.go

```go
func RunService(
    client *http.Client,
    consumer eventbus.Consumer,
    producer eventbus.Producer,
) error {
    handler := handlers.PizzaHandler(client, producer)

    err := consumer.On(&just.PizzaOrdered{}, handler)
    if err != nil {
        return err
    }

    return nil
}
```




go-kit.sh
go-kit update

service.json

```
{
  "event_bus": {
    "consumes_topics": [
      {
        "name": "pizza-ordered",
        "go_name": "justeat.PizzaOrdered",
        "timeout": 30,
        "retry_count": 5,
        "worker_pool": 25,
        "replica_count": 3,
        "pod_resources": {
          "cpu": "small",
          "memory": "small"
        },
        "consumer_lag_alert": {
          "number_of_messages": 50,
          "time_period": "2m"
        }
      }
    ]
  }
}
```

Events

Consumes

Name	CPU	Memory	Worker Pool
justeat.PizzaOrdered	small 50m	small 50Mi	25

Produces

Name
justeat.PizzaSucceeded

Deployment

Pull Request

Add a title

Creating Pizza Service

Add a description

Write | Preview

Context

<!-- Required: What is this PR achieving? Provide a brief description on a high level of the problem or feature being addressed in this PR. Include any relevant background information that helps explain why these changes are necessary. -->

Changes

<!-- Required: Describe the changes you have made in this PR. Focus on what has been added, removed, or modified. -->

Refactors

<!-- Describe any refactoring done as part of this PR. Explain the purpose and how it improves the codebase. -->

Markdown is supported | Paste, drop, or click to add files

Capability Testing

Drift Detection

service.json

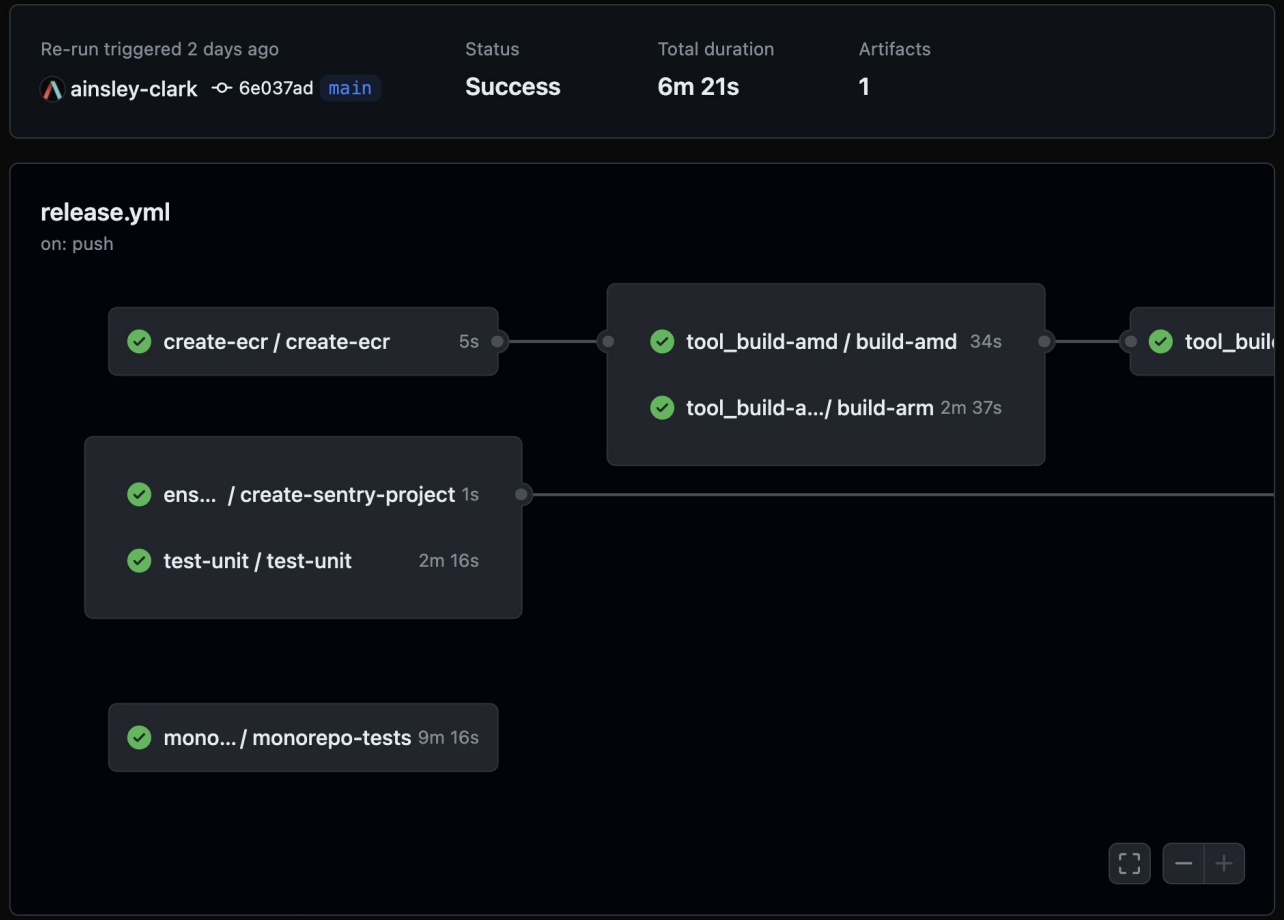
```
consumer.OnAll(map[proto.Message]eventbus.EventHandler{
    &justeat.MyEvent{}: func(ctx context.Context, event any, headers ...eventbus.Header) error {
        return ni
    },
})
```

github-actions bot commented 6 minutes ag

Generated by go-kit ci: Service Validation

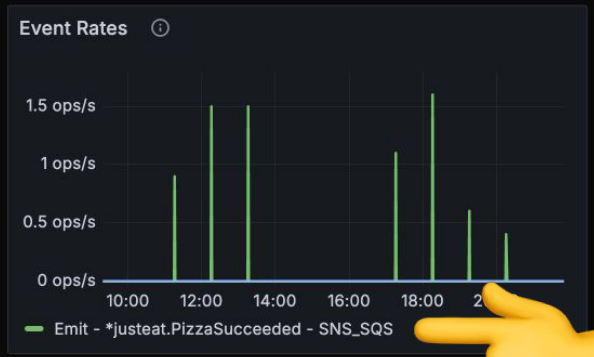
Consumer topic "my-event" in service.go is not a used topic in service.json
Consumer topics used in service.go

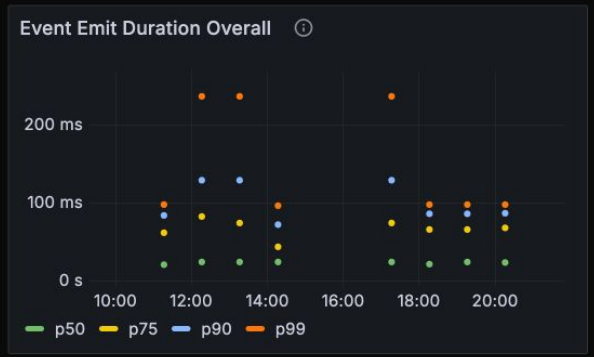
Rollout


Re-run triggered 2 days ago
ainsley-clark
6e037ad
main
Status
Success
Total duration
6m 21s
Artifacts
1
release.yml
on: push
create-ecr / create-ecr 5s
tool_build-amd / build-amd 34s
tool_build-a.../ build-arm 2m 37s
tool_buil
ens... / create-sentry-project 1s
test-unit / test-unit 2m 16s
mono... / monorepo-tests 9m 16s

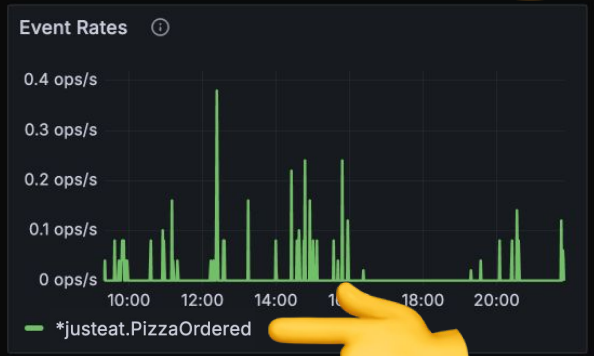
Metrics

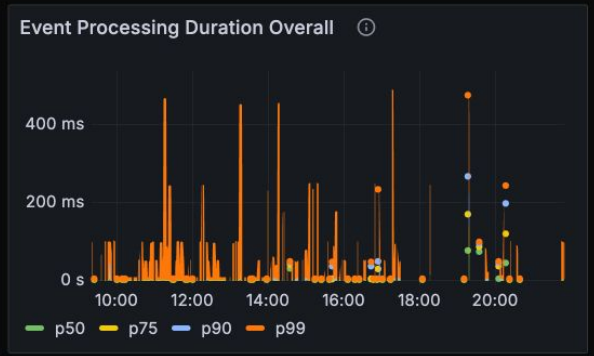
Events Produced





Events Consumed

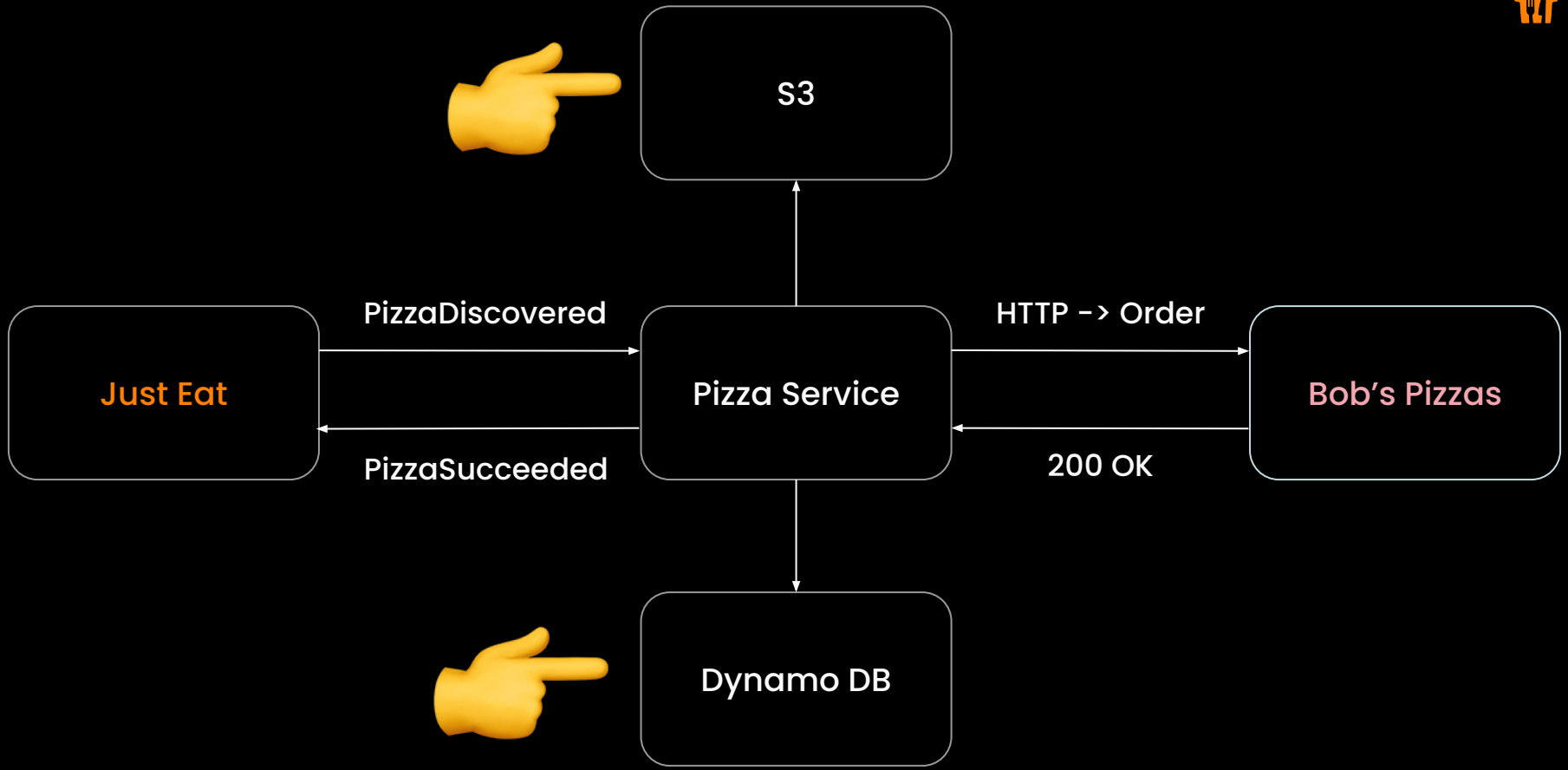




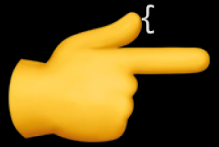
Recap

- Ran go-kit new
- Created one handler
- Attached it to a consumer
- And we’ve deployed a service with metrics, workflows and much more.

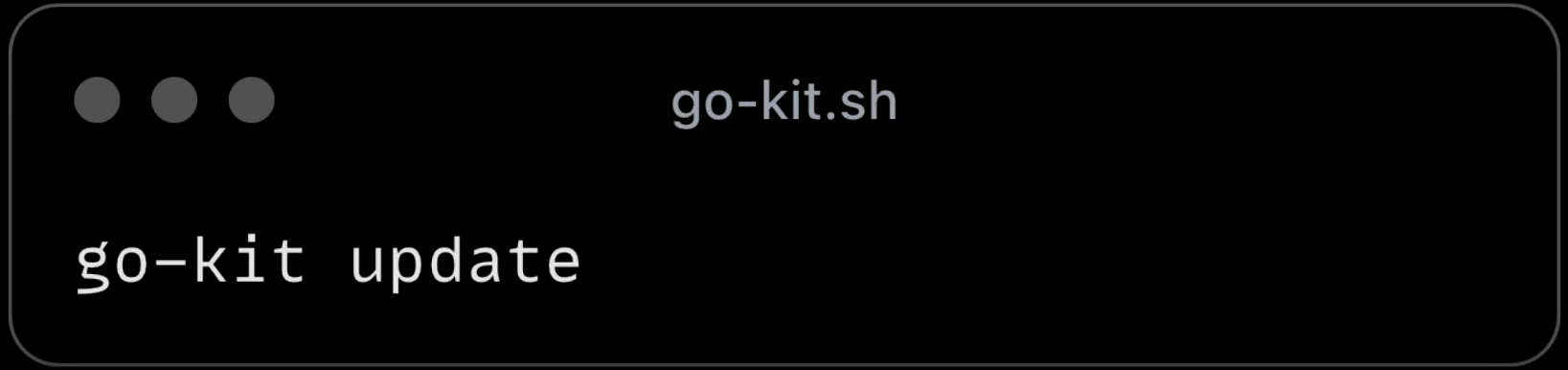
Going Further


S3
PizzaDiscovered
HTTP -> Order
Just Eat
Pizza Service
Bob's Pizzas
PizzaSucceeded
200 OK
Dynamo DB

service.json



```
{
    "resources": {
        "databases": [
            {
                "name": "analytics",
                "description": "Keeps track of basket amounts of orders",
                "attributes": null,
                "indexes": null
            }
        ],
        "object_stores": [
            {
                "name": "orders",
                "description": "Stores the order payload before injecting",
                "object_expire_days": 3
            }
        ]
    }
}
```


go-kit.sh
go-kit update

DynamoDB Tables

Name	Description	Composite
analytics	Keeps track of basket amounts of orders	❌

Stores

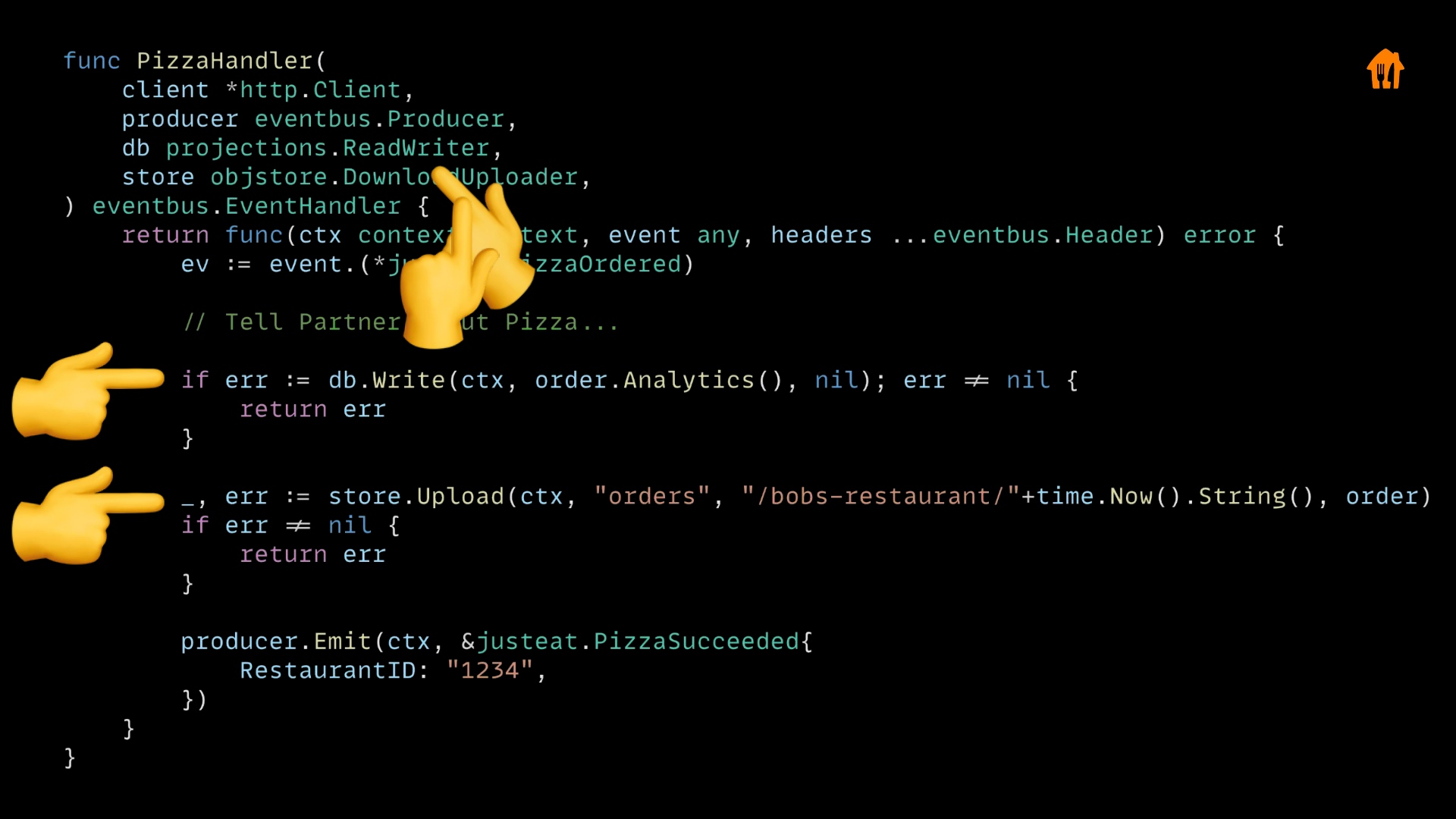
Name	Description	Type	🕐 Expiry
orders	Stores the order payload before injecting	object store	3 days

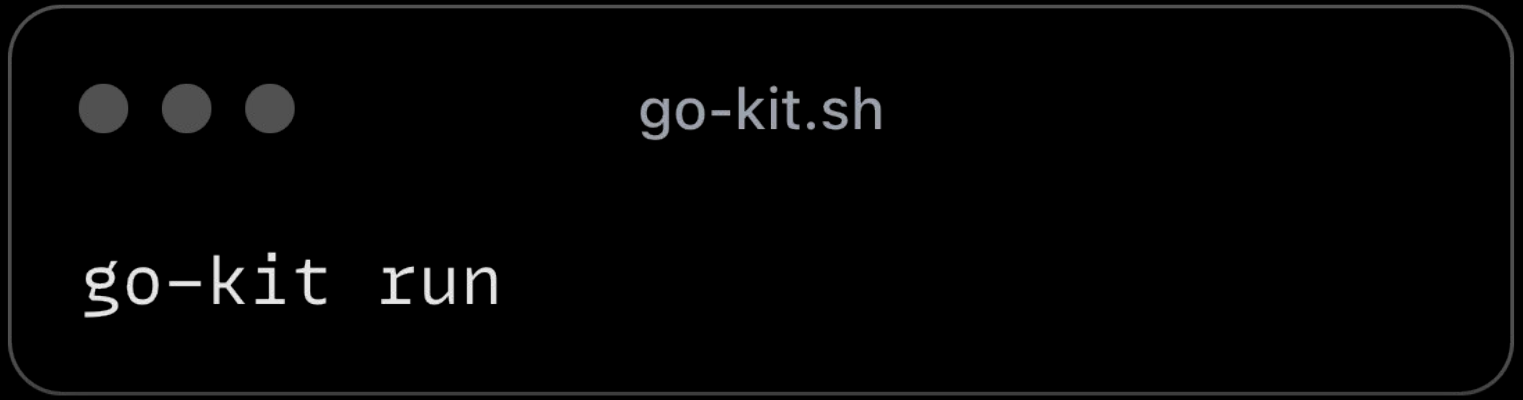
internal/app/service.go

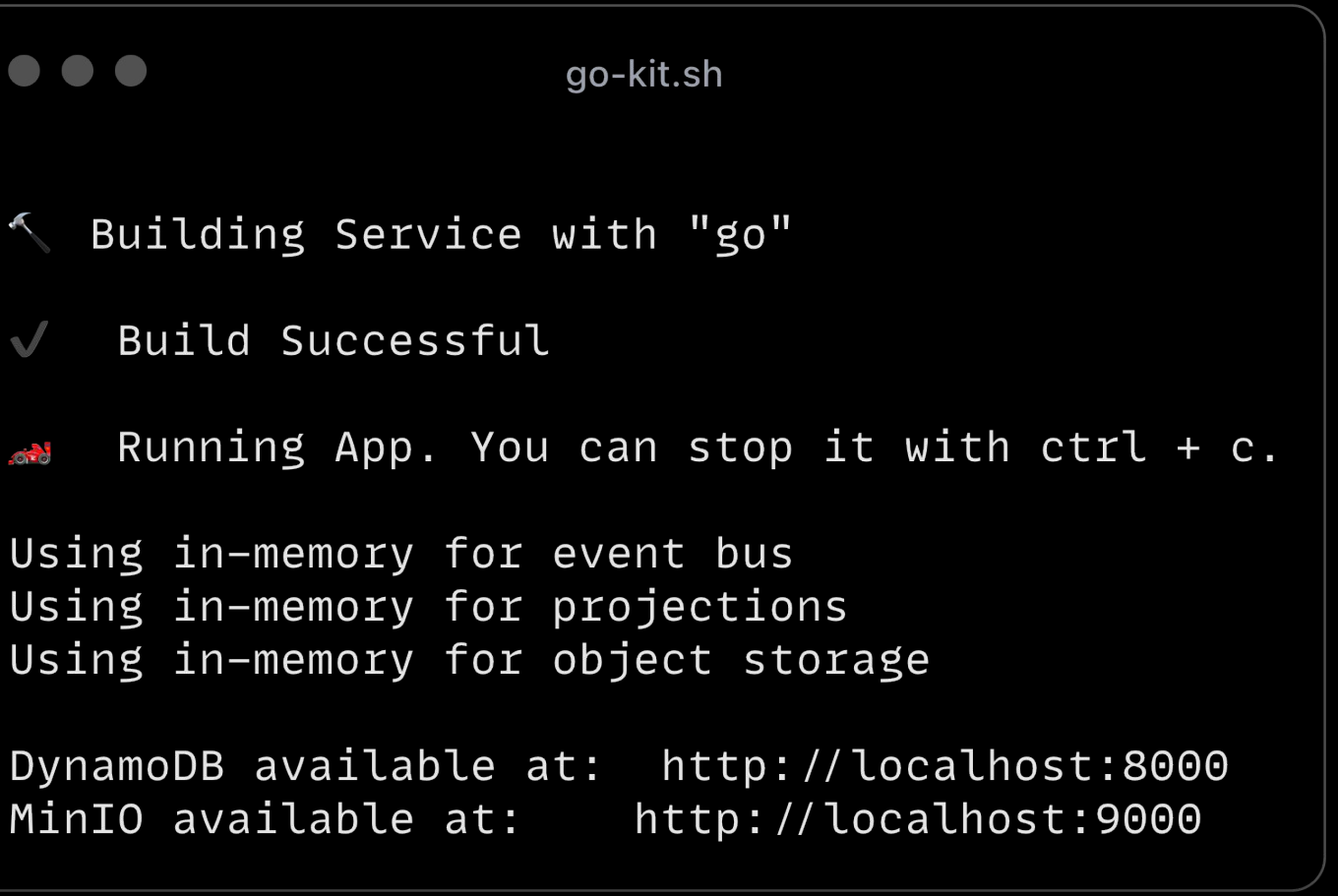
```go
func RunService(
    client *http.Client,
    consumer eventbus.Consumer,
    producer eventbus.Producer,
    db projections.ReadWriter,
    store objstore.DownloadUploader,
) error {
    handler := handlers.PizzaHandler(client, producer, db, store)

    err := consumer.On(&justeat.PizzaOrdered{}, handler)
    if err != nil {
        return err
    }

    return nil
}
```

go-kit

```go
type ReadWriter interface {
    Reader
    Writer
}

type Reader interface {
    Read(ctx context.Context, id string, projection any) error
}

type Writer interface {
    Write(ctx context.Context, data any, expires *time.Time) error
}
```

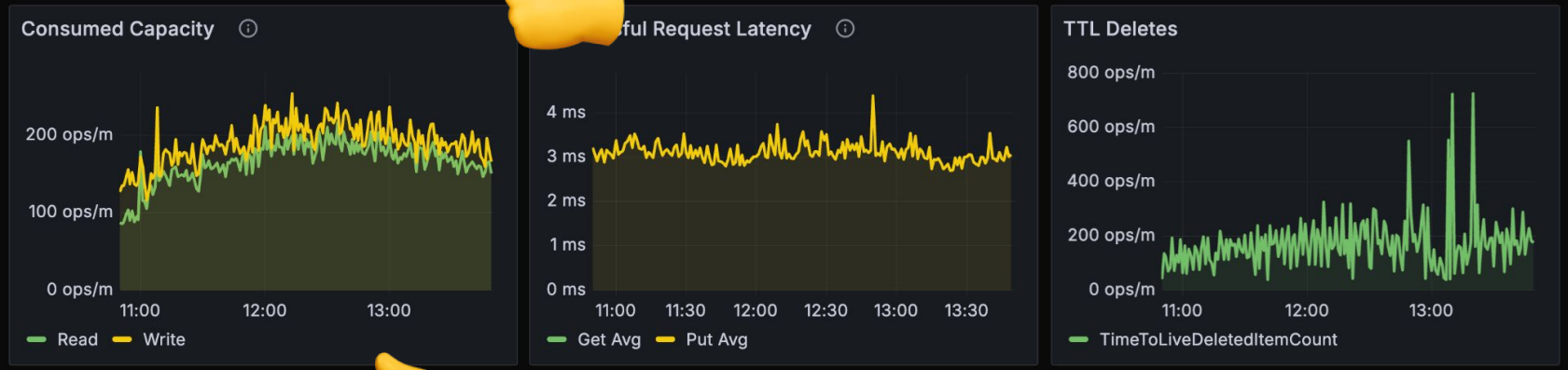

func PizzaHandler(
client *http.Client,
producer eventbus.Producer,
db projections.ReadWriter,
store objstore.DownloadUploader,
) eventbus.EventHandler {
return func(ctx context.Context, event any, headers ...eventbus.Header) error {
ev := event.(*justeat.PizzaOrdered)
// Tell Partner About Pizza...
if err := db.Write(ctx, order.Analytics(), nil); err != nil {
return err
}
_, err := store.Upload(ctx, "orders", "/bobs-restaurant/"+time.Now().String(), order)
if err != nil {
return err
}
producer.Emit(ctx, &justeat.PizzaSucceeded{
RestaurantID: "1234",
})
}
}


go-kit.sh
go-kit run

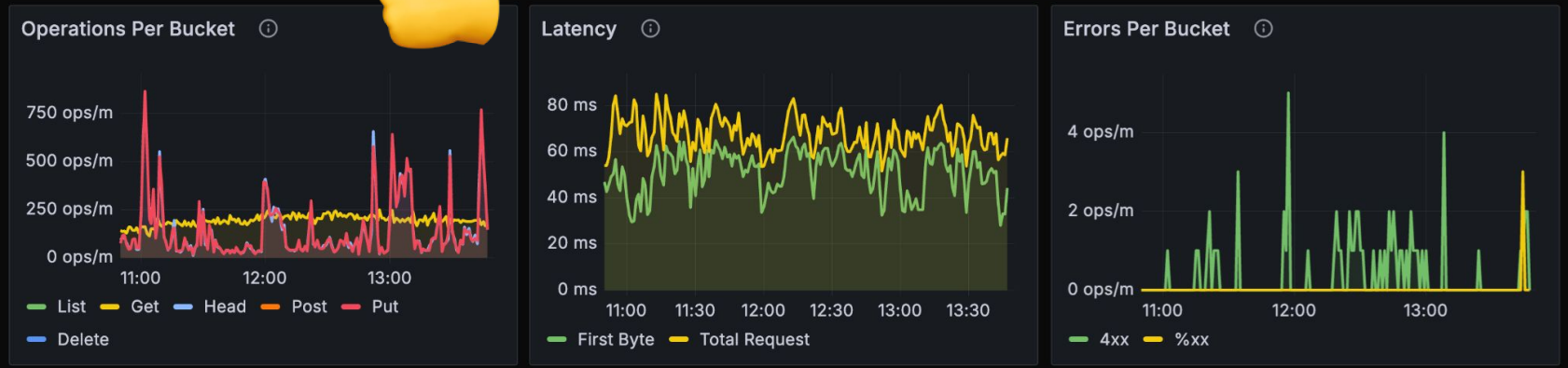

go-kit.sh
🔨 Building Service with "go"
✔ Build Successful
🏎 Running App. You can stop it with ctrl + c.
Using in-memory for event bus
Using in-memory for projections
Using in-memory for object storage
DynamoDB available at: http://localhost:8000
MinIO available at: http://localhost:9000

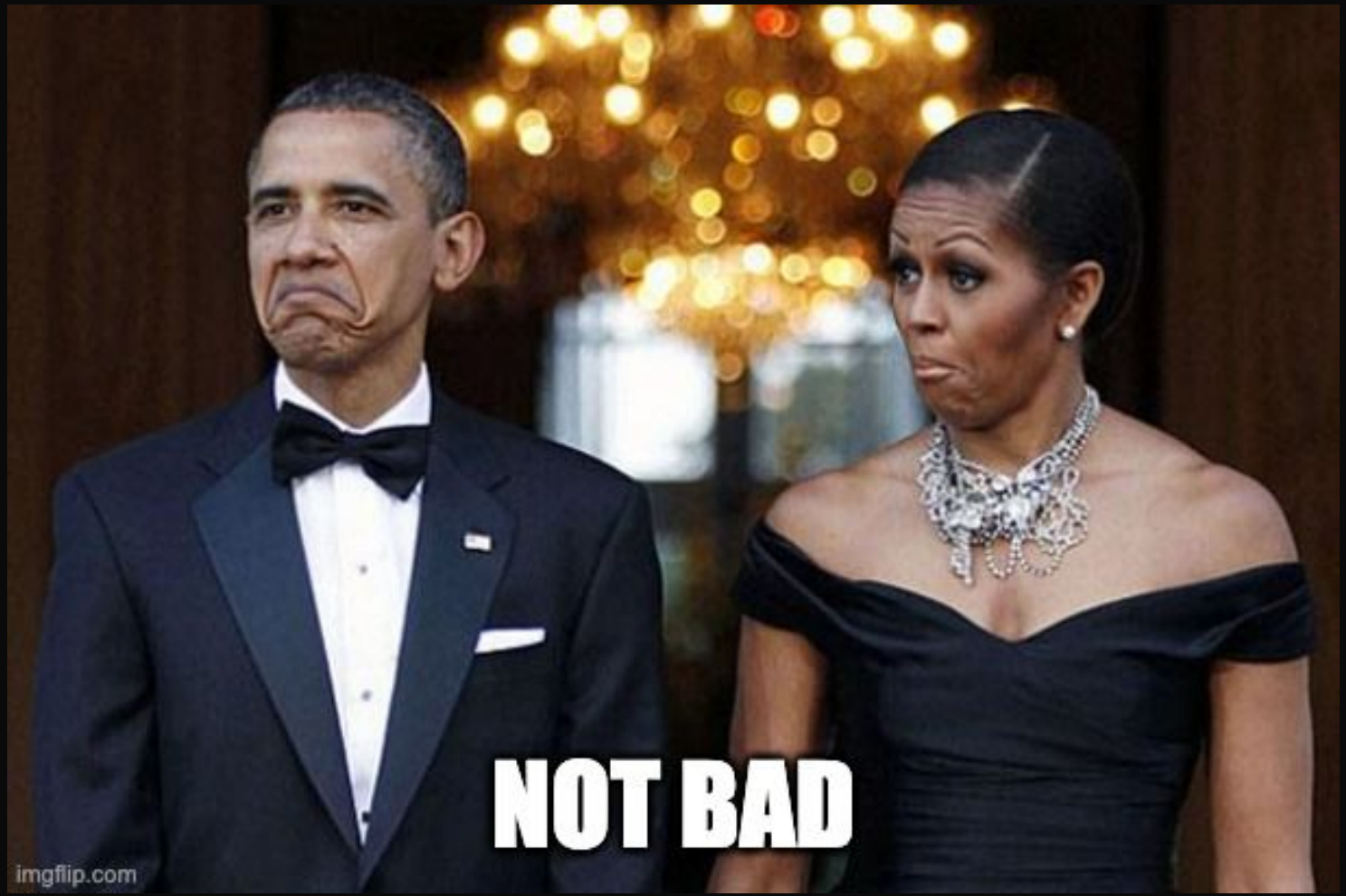
Deploy…

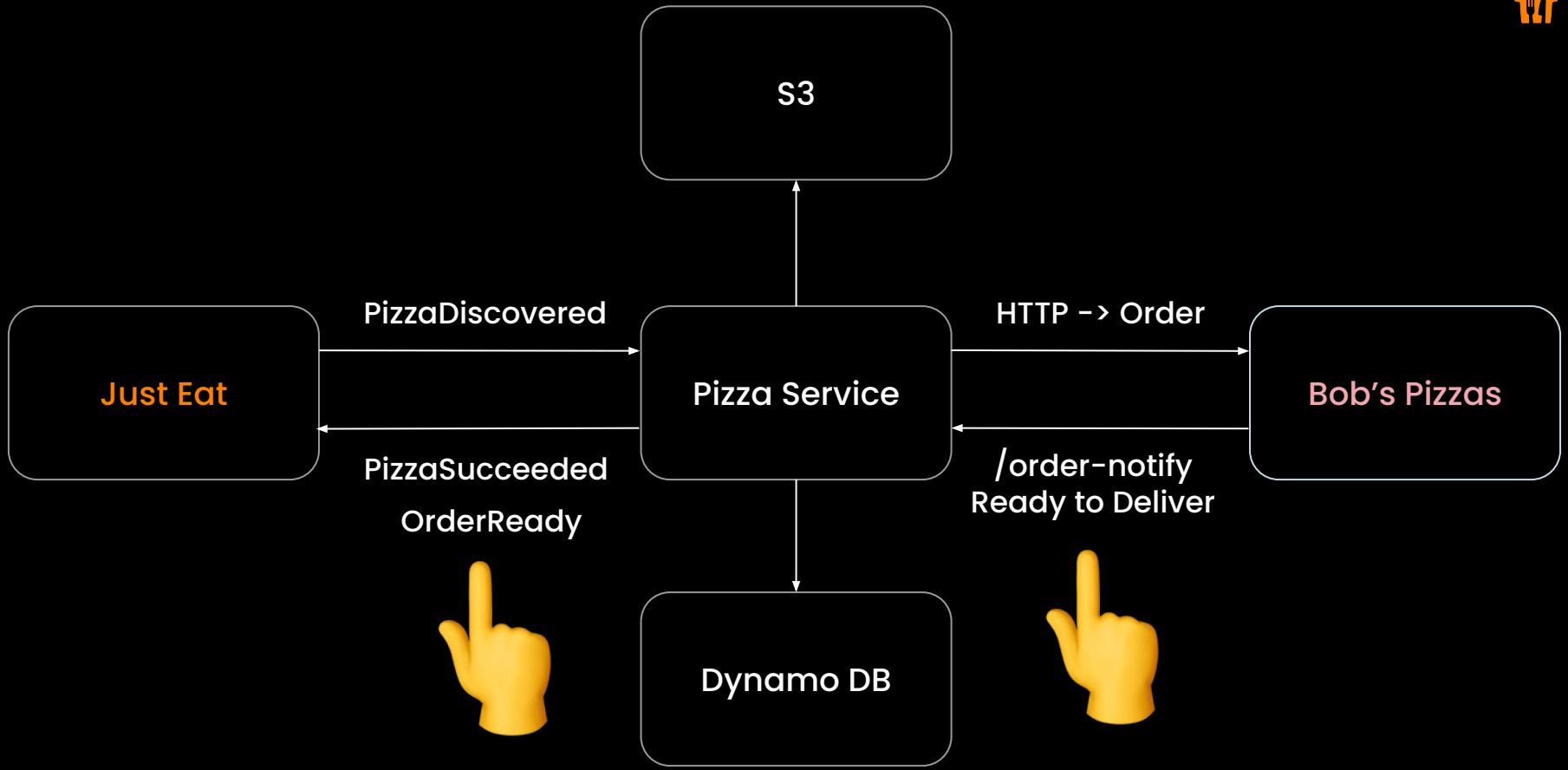
Resource - DynamoDB - Analytics



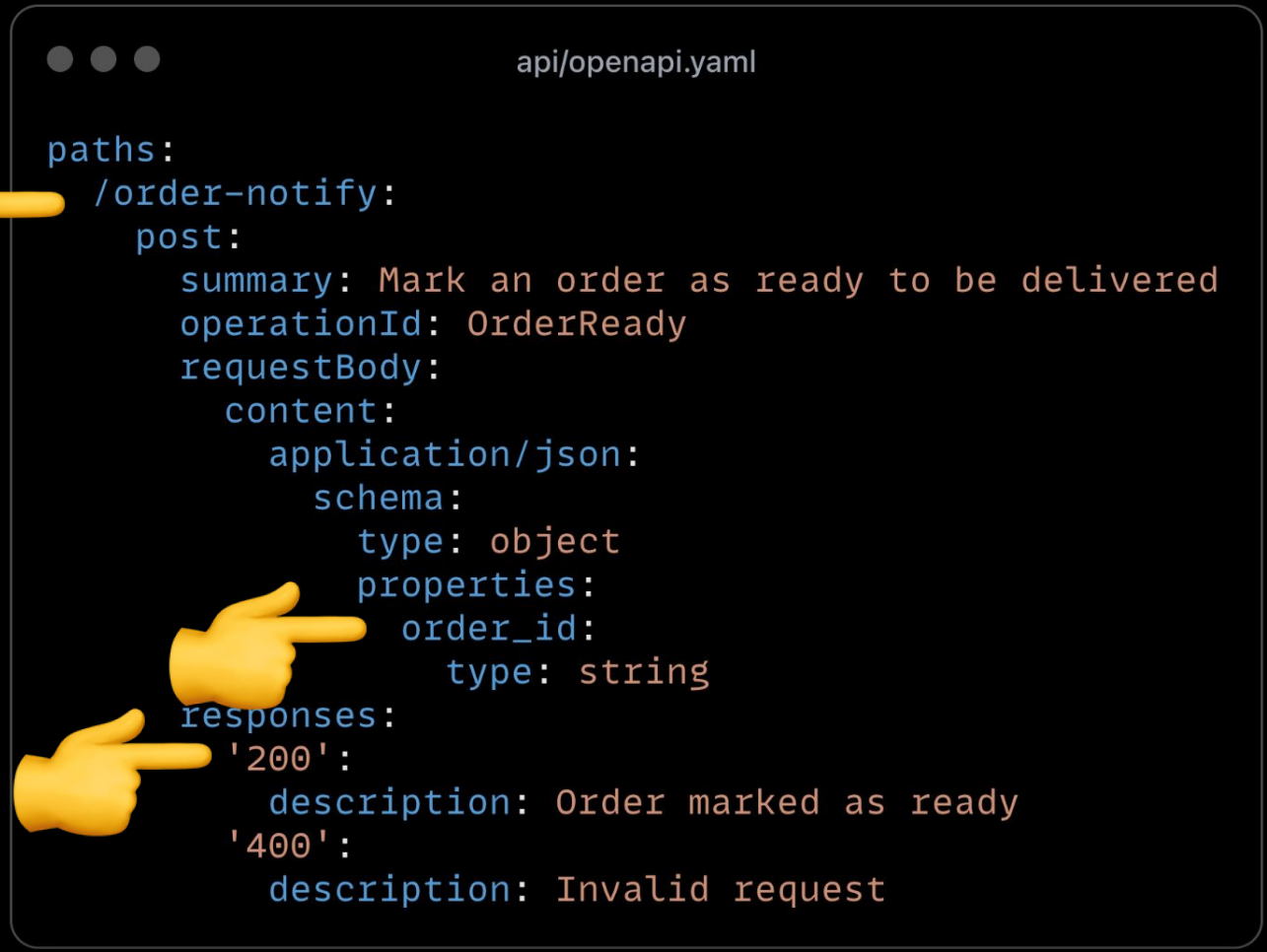
Resource - S3 - Orders

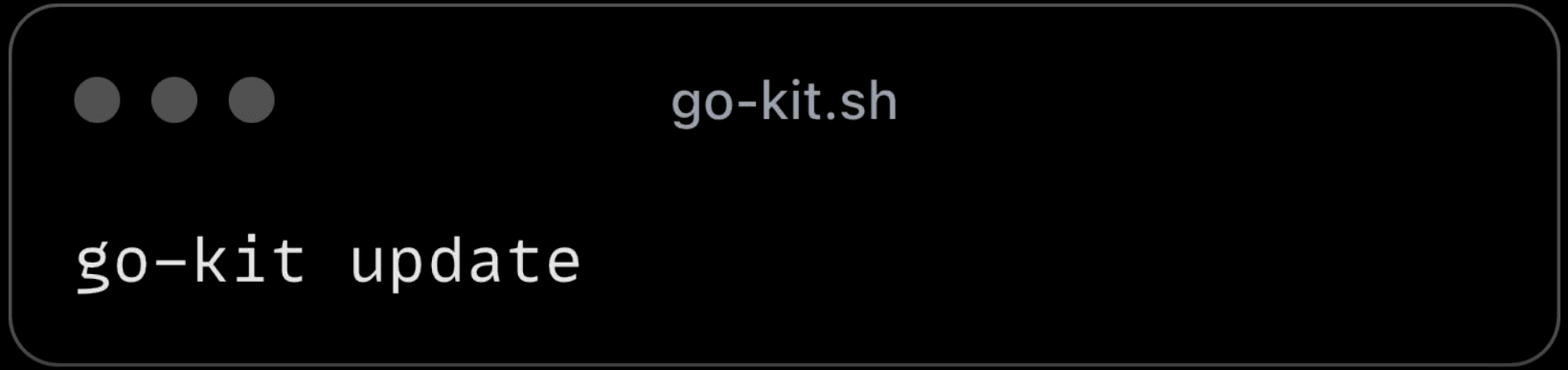


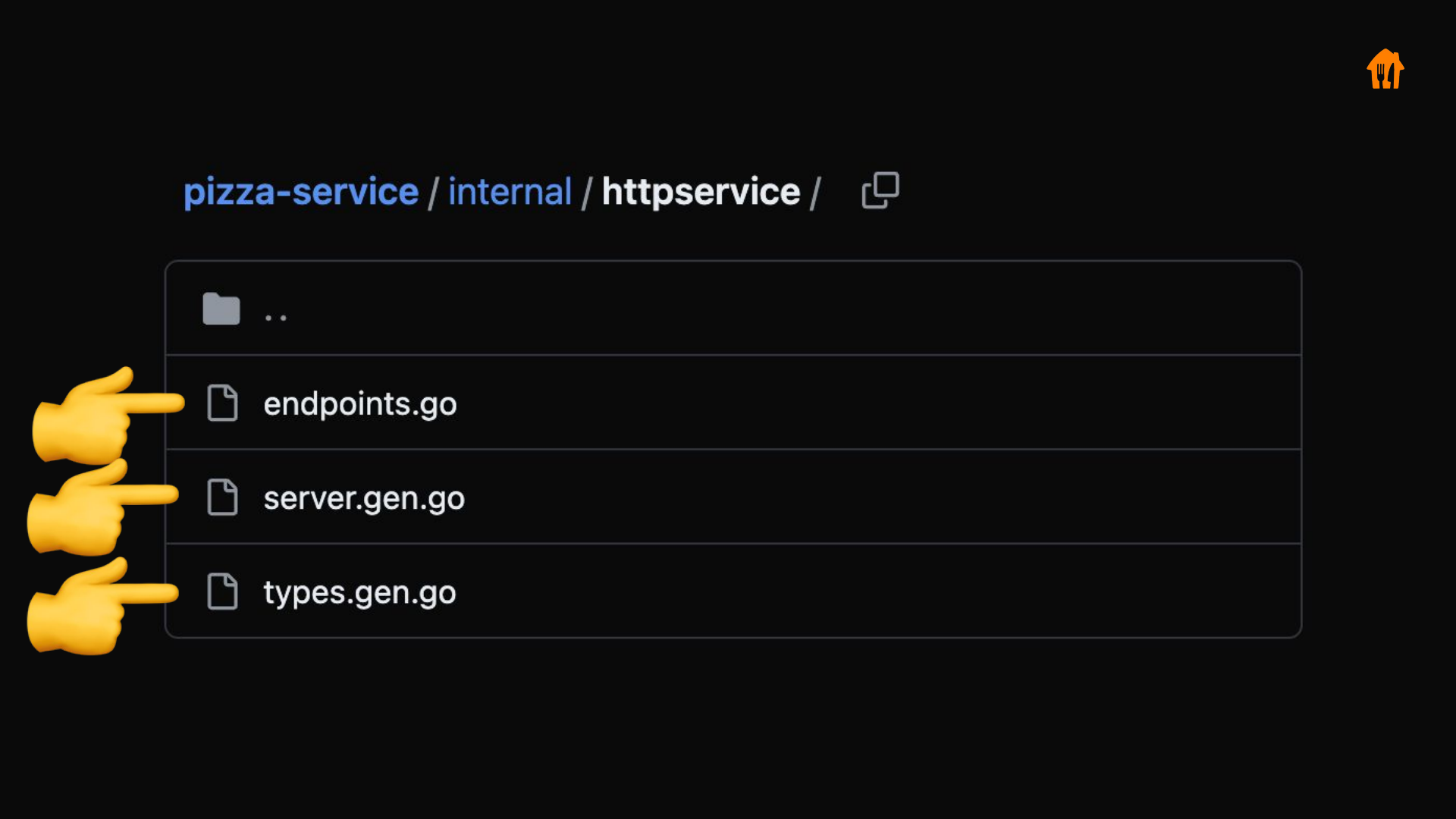

NOT BAD
imgflip.com


S3
Just Eat
PizzaDiscovered
PizzaSucceeded
OrderReady
Pizza Service
HTTP -> Order
/order-notify
Ready to Deliver
Bob's Pizzas
Dynamo DB


service.json
{
"https": {
"enabled": true,
"type": "api",
"timeout_second": 5,
"pod_resources": {
"cpu": "small",
"memory": "small"
},
"min_replicas": 3,
"max_replicas": 50
}
}


api/openapi.yaml
paths:
/order-notify:
post:
summary: Mark an order as ready to be delivered
operationId: OrderReady
requestBody:
content:
application/json:
schema:
type: object
properties:
order_id:
type: string
responses:
'200':
description: Order marked as ready
'400':
description: Invalid request


go-kit.sh
go-kit update


pizza-service / internal / httpservice /
..
endpoints.go
server.gen.go
types.gen.go

internal/app/service.go

```go
// ServerInterface represents all server handlers.
type ServerInterface interface {
    // Mark an order as ready
    // (POST /order)
    OrderReady(ctx echo.Context) error
}

type ServerInterfaceWrapper struct {
    Handler ServerInterface
}
```

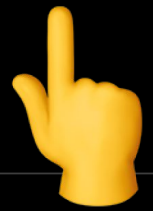
internal/httpservice/endpoints.go

```go
func (h HTTPHandler) OrderReady(c echo.Context) error {
    ctx := c.Request().Context()
    logger := log.WithContext(ctx)

    var in OrderReadyJSONBody
    if err := c.Bind(in); err != nil {
        return echo.NewHTTPError(http.StatusBadRequest, err.Error())
    }

    logger.Info("Order has been marked as ready: " + in.OrderId)

    h.Producer.Emit(ctx, &justeat.OrderReady{
        OrderId: in.OrderId,
    })
}
```

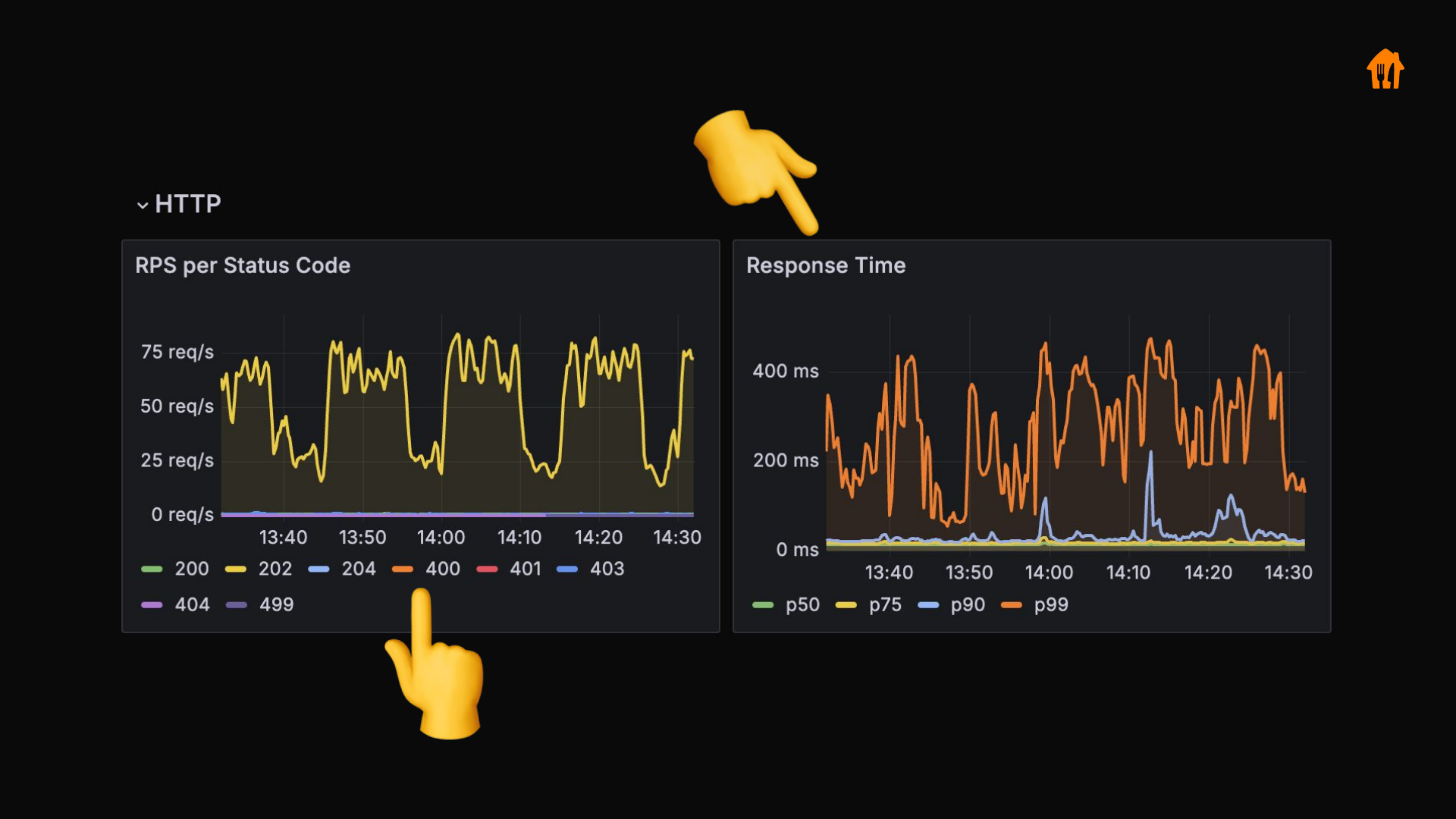


internal/app/service.go

```go
func RunService(
    client *http.Client,
    consumer eventbus.Consumer,
    producer eventbus.Producer,
    db projections.ReadWriter,
    store objstore.DownloadUploader,
    http *echo.Echo,
) error {
    handler := handlers.PizzaHandler(client, producer, db, store)

    err := consumer.On(&justeat.PizzaOrdered{}, handler)
    if err != nil {
        return err
    }

    httpservice.RegisterHandlers(http, httpservice.HTTPHandler{
        Producer: producer,
    })

    return nil
}
```



Deploy…


HTTP
RPS per Status Code
75 req/s
50 req/s
25 req/s
0 req/s
13:40
13:50
14:00
14:10
14:20
14:30
200
202
204
400
401
403
404
499
Response Time
400 ms
200 ms
0 ms
13:40
13:50
14:00
14:10
14:20
14:30
p50
p75
p90
p99

Product Guy 11:25

Sweet!

Results

Go Kit helps us to

- Update our entire estate (150+ services) in one PR.
- Have naturally occurring SRE & updates over time.
- Create first class documentation without intervention
- Focus on business problems, not deployment.
- CI/CD updates and changes easily implemented.
- Helps us with patterns for file structure.
- Have extremely robust end to end testing.
- Scale to hundreds of millions of orders a month.

Thank you!
Questions?