Event-Driven Microservices with NATS Streaming

Shiju Varghese

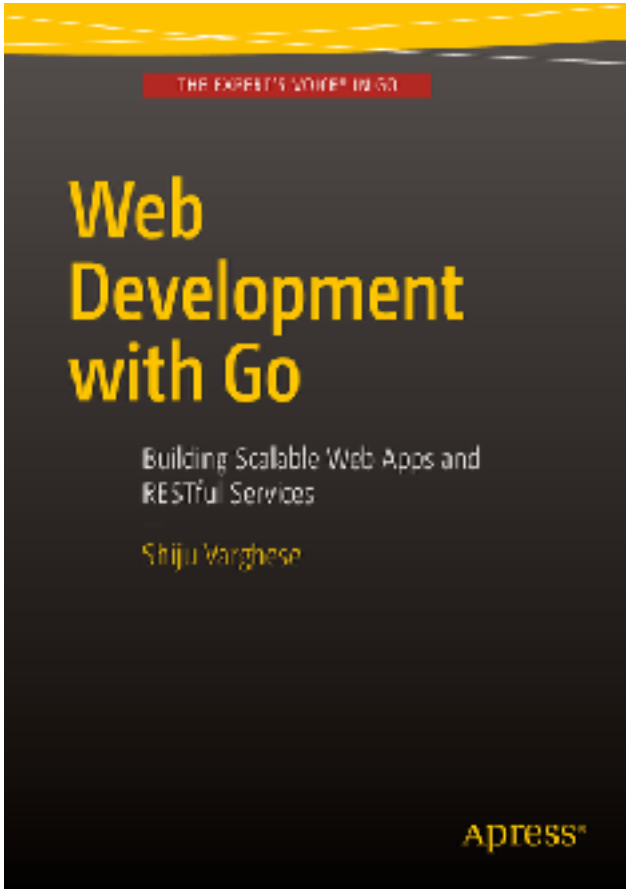
Consulting Solutions Architect

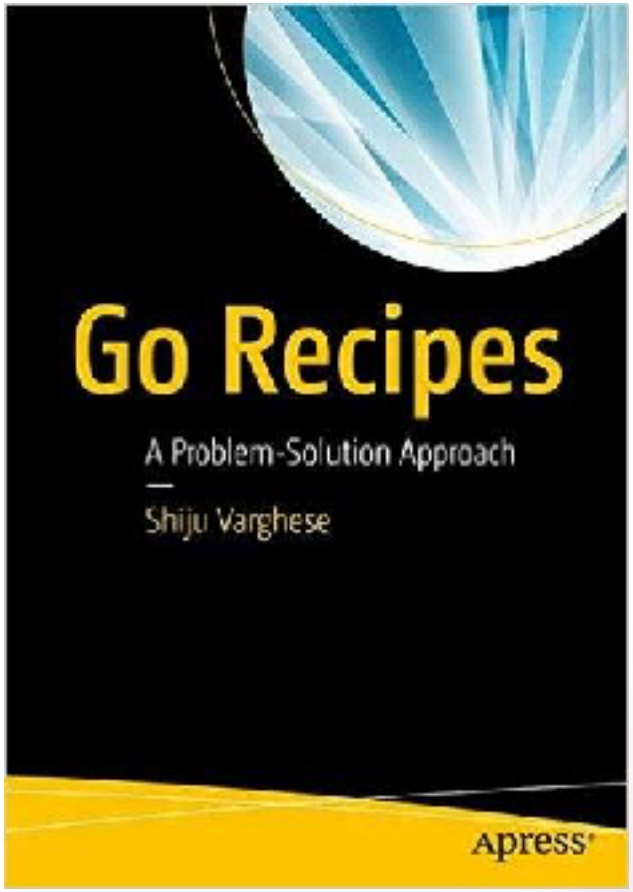
https://medium.com/@shiuvar

About Me

- Consulting Solutions Architect
- Consulting and Training on Golang and Cloud-Native Distributed Systems
- Early adopter of Go programming language
- Published Author
- Honoured with Microsoft MVP award seven times
- Blog: medium.com/@shijuvar |Twitter: @shijucv

Author


Web
Development
with Go
Building Scalable Web Apps and
RESTful Services
Shiju Varghese
Apress

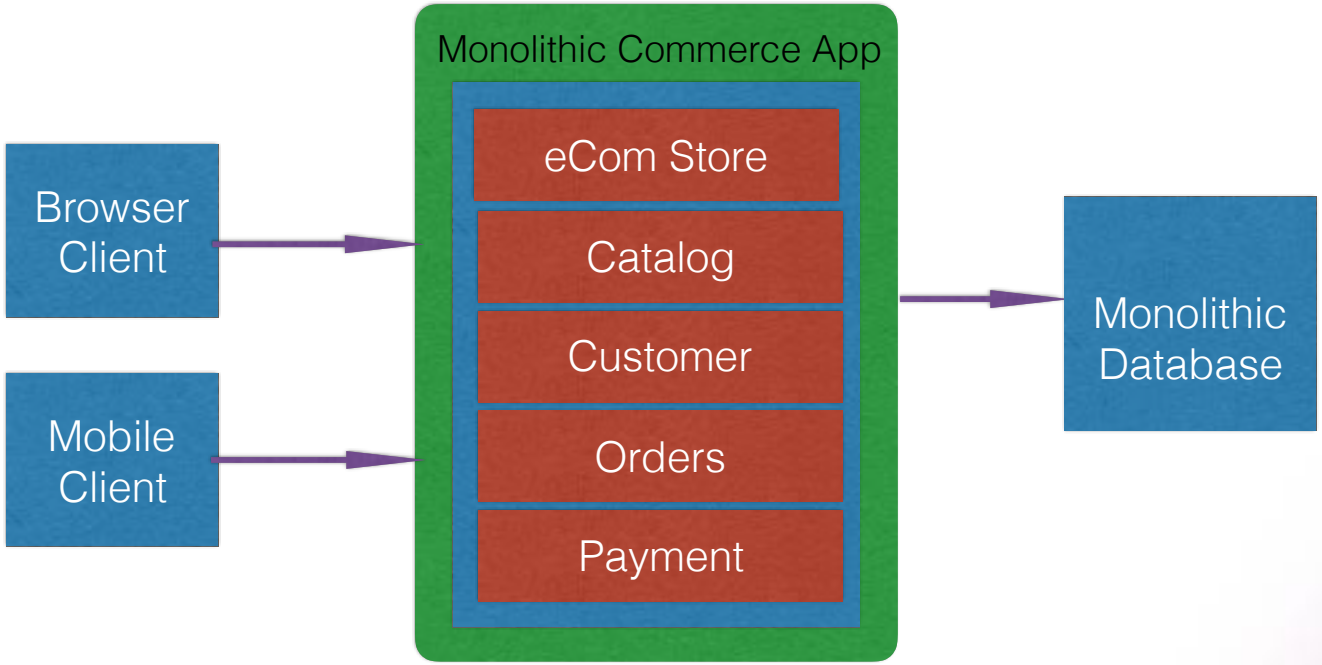

Go Recipes
A Problem-Solution Approach
Shiju Varghese
Apress

Agenda

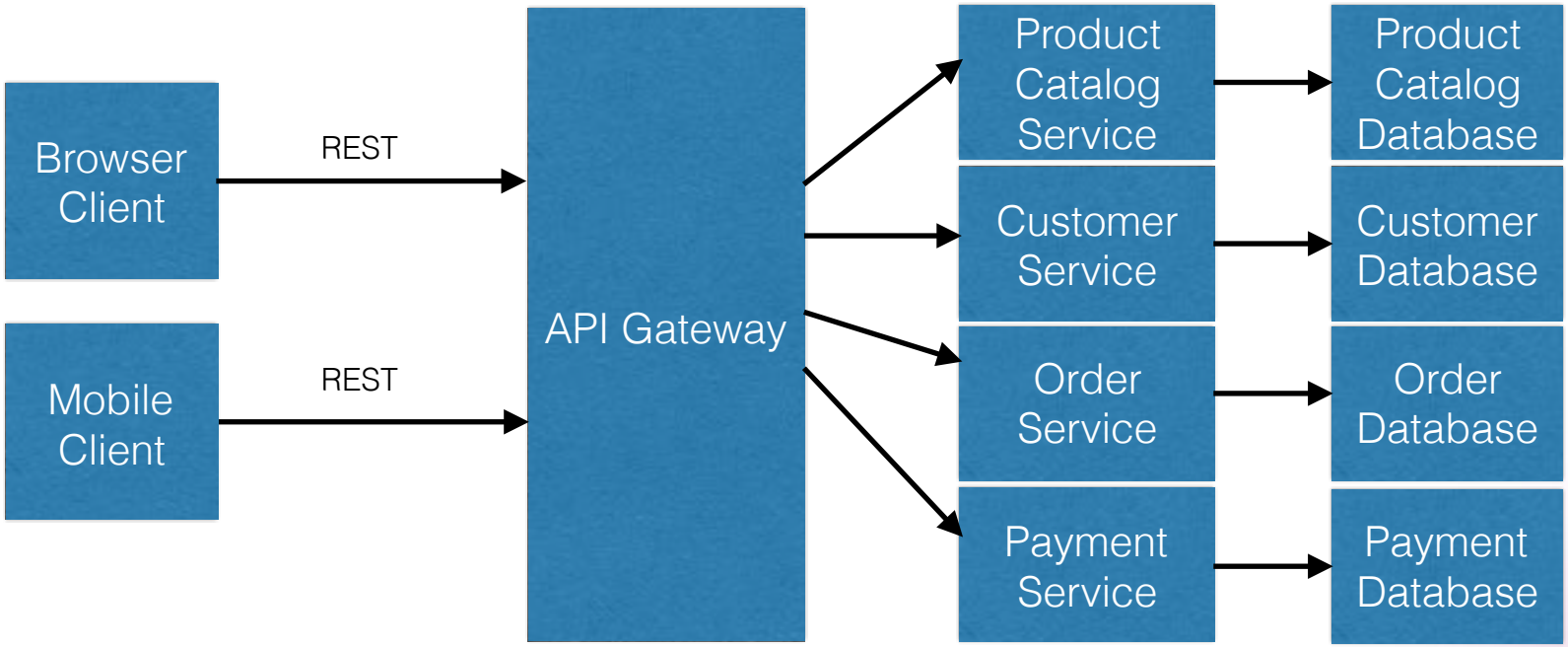
- Overview about the practical Challenges of Microservices Architecture
- Introduction to Event-Driven Architectures
- Introduction to NATS Streaming Server
- Using NATS Streaming Server to build Event-Driven Microservices


MICROSERVICES
ARCHITECTURE AND ITS
PRACTICAL CHALLENGES

Monolithic Architetture



Microservices Architecture



Autonomous Services Around Bounded Context

- Software broken up into functional components.
- Componentization via Services in which each service is packaged as one unit of execution.
- Independent, autonomous process with no dependency on other Microservices.
- Services are organized around business capability.
- Decentralization of Data Management.
- Independently replaceable and upgradeable.

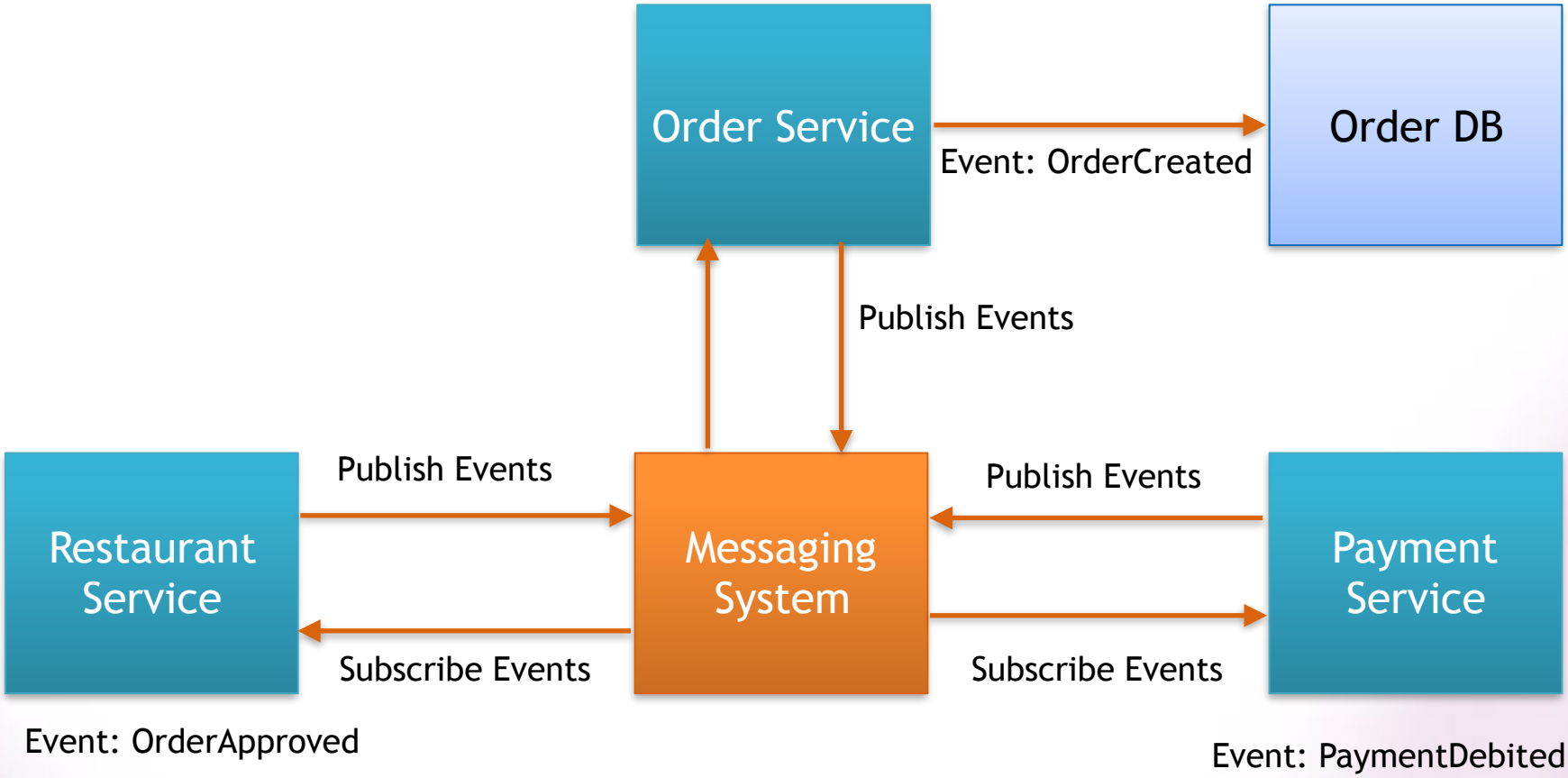
Practical Challenges

- Database per service, Isolated persistence
- A transaction may span into multiple microservices
- Distributed transactions with Two-Phase Commit (2PC) is not viable option
- How to manage failures while performing distributed transactions
- How to manage data consistency
- How to querying data from multiple data store

The greatest challenge of any Microservices based implementation is how to manage its data and transactions


EVENT-DRIVEN
ARCHITECTURES

Event-Driven Architecture



Events

- A fact represents something that has happened in the system (Eg: OrderCreated, PaymentDebited, OrderApproved)
- Events are immutable

Event-Driven Architectures

- Application Event
- Transaction Log Tailing
- Event Sourcing

Event Sourcing

- An event-centric pattern for implementing business logic and persisting aggregates
- Events represent state changes of aggregates; Publishes domain events on mutations of aggregates
- Event log for the aggregates
- Preserves the history of aggregates

Event Store of Immutable Log of Events

Aggregate ID	Aggregate Type	Event ID	Event Type	Event Date
301	Order	1001	OrderCreated	…
301	Order	1002	OrderApproved	…
301	Order	1003	OrderShipped	…
301	Order	1004	OrderDelivered	…

Command Query Responsibility Segregation (CQRS)

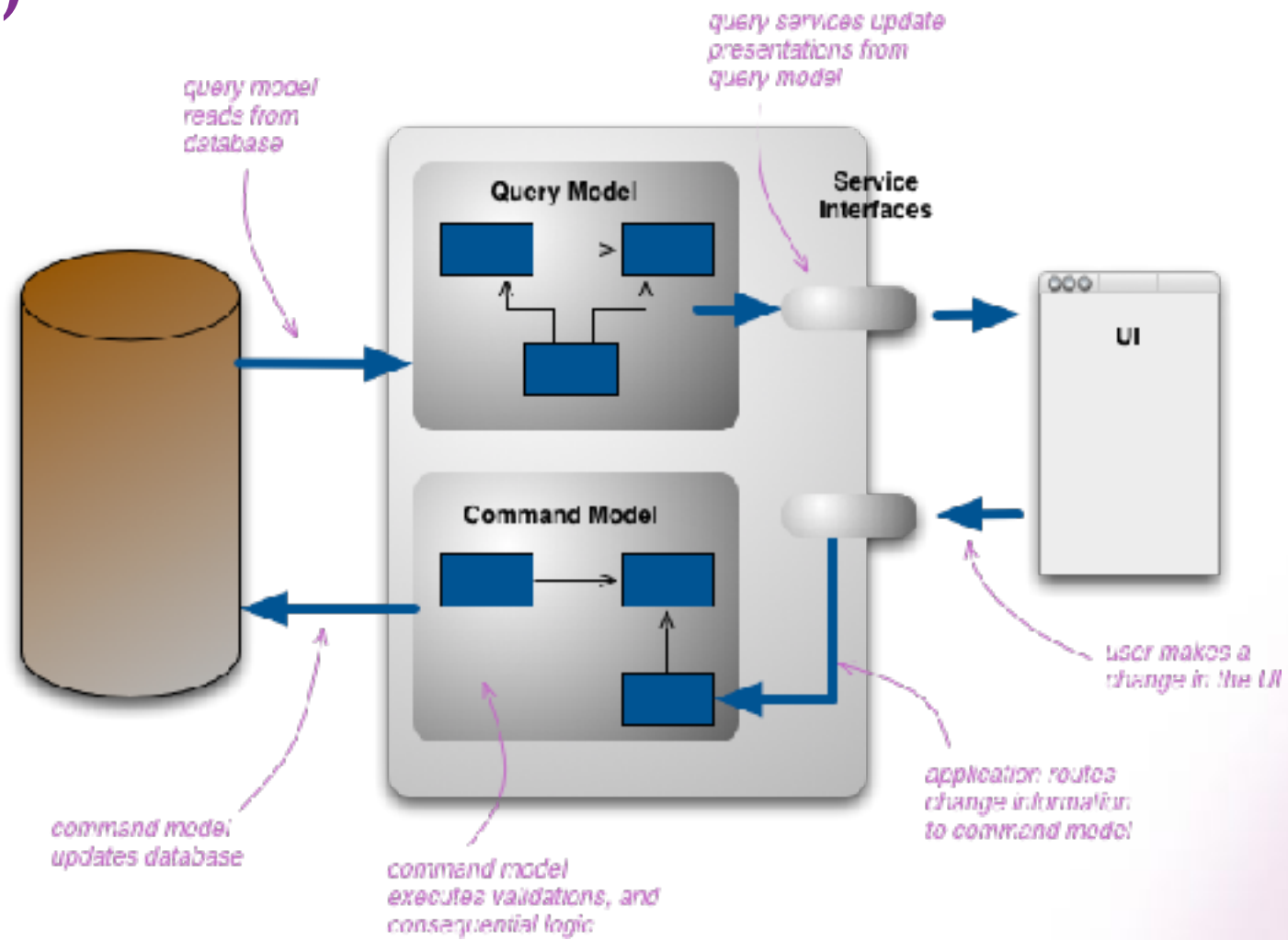


Image Courtesy: http://martinfowler.com


NATS
Open Source. Simple. Secure. Scalable.
NATS is now a hosted CNCF Project

NATS

- A Cloud Native Computing Foundation (CNCF) hosted incubation project
- NATS is an open-source, high performant cloud-native messaging system
- NATS Streaming is an extremely performant, lightweight reliable streaming platform built on the top of core NATS platform that provides persistent logs.
- Both NATS and NSTS Streaming Server are written in Go
- NATS was created by Derek Collison, and supported by his team at Synadia
- NATS is deployed in some of the largest cloud platforms, including: VMware, CloudFoundry, Baidu, Siemens, and GE

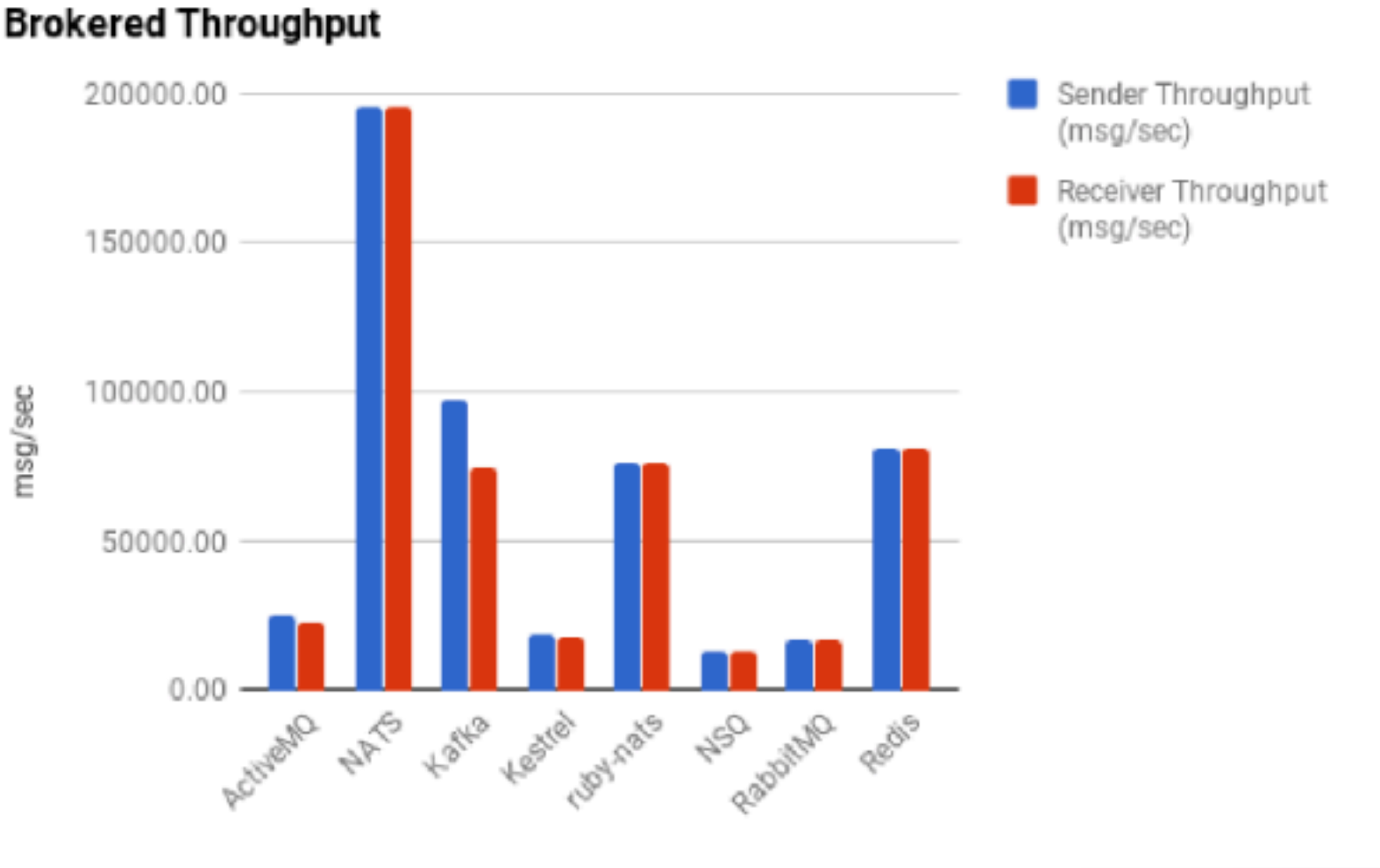
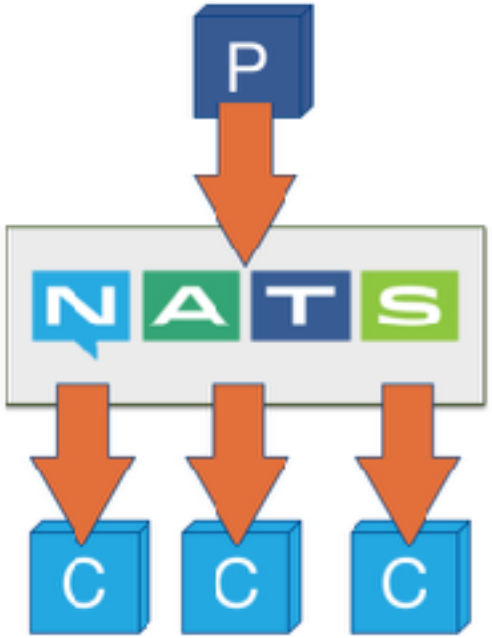


Chart source: http://bravenewgeek.com/dissecting-message-queues/

Publish - Subscribe

- Basic messaging pattern
- A producer publishes a message with a specific subject/channel
- All active consumers with subscriptions matching the subject/channel receive it



NATS

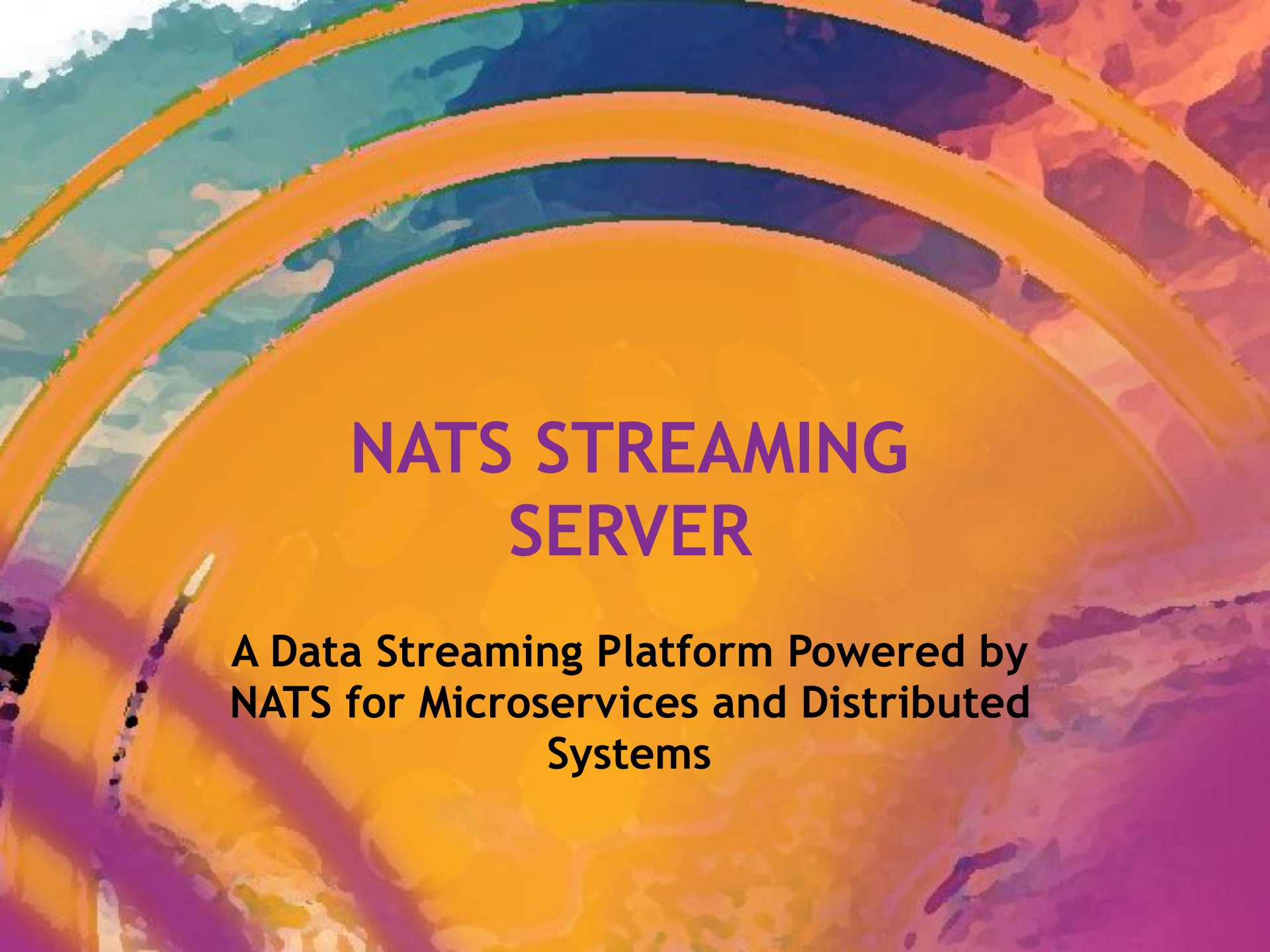
NATS Server

Synadia Supported Clients

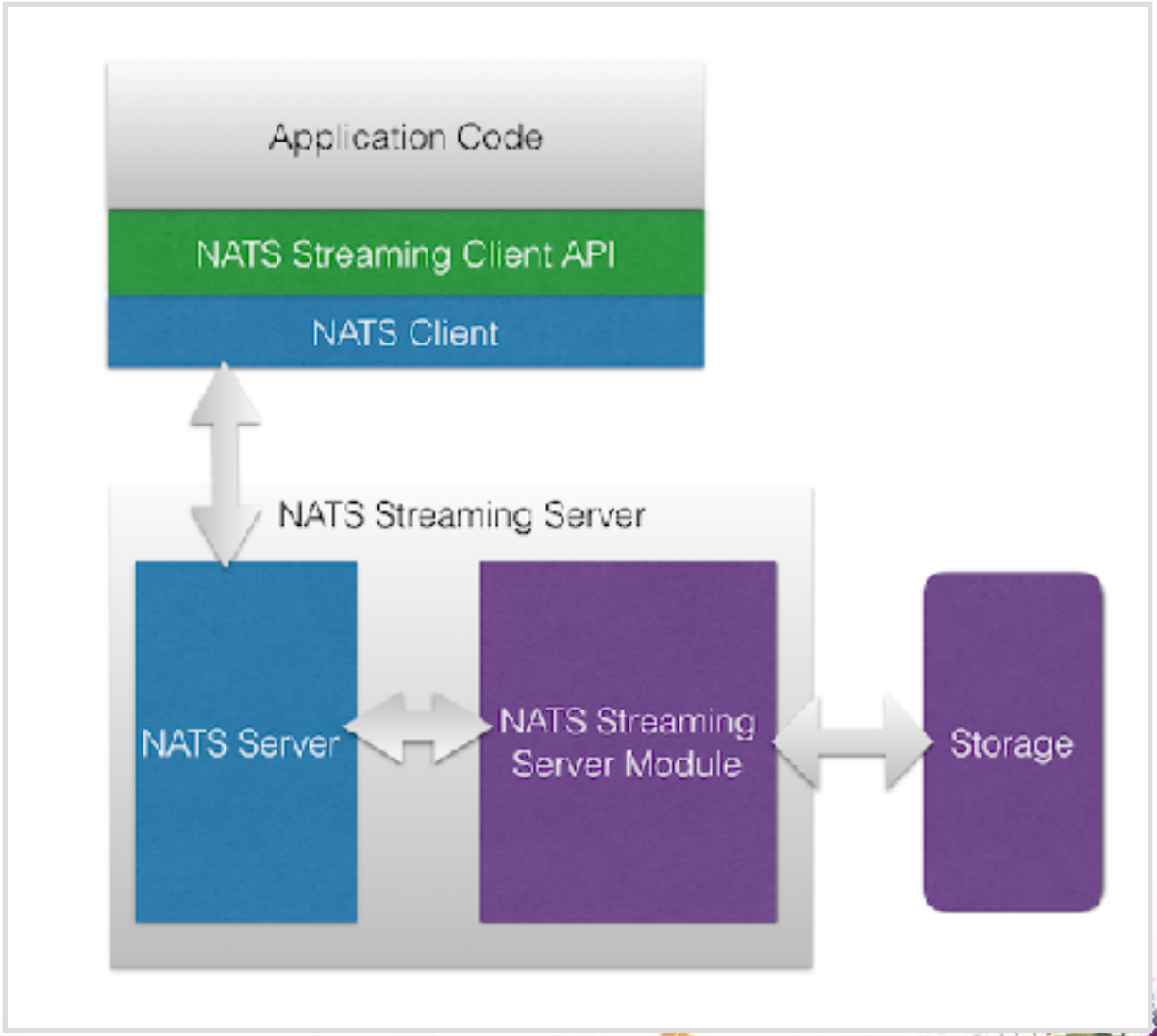
C /C# /Elixir /Go /Java /NGINX /Node.js /Pure Ruby /Python Asyncio /Python Tornado /Ruby

Community Clients

.NET /Arduino /Clojure /Elixir /Elm /Erlang /Haskell /Lua /MicroPython /PHP /Perl /Python /Python Twisted /Qt5 C++ / Rust /Scala /Spring API /Swift


NATS STREAMING
SERVER
A Data Streaming Platform Powered by
NATS for Microservices and Distributed
Systems

NATS Streaming Server



- Message/event persistence
- At-least-once-delivery
- Publisher rate limiting
- Rate matching/limiting per subscriber
- Historical message replay by subject
- Durable subscriptions

Publish Events

```
// publishEvent publishes an event via NATS Streaming server
func publishEvent(event *pb.Event) {
    // Connect to NATS Streaming server
    sc, err := stan.Connect(
        clusterID,
        clientID,
        stan.NatsURL(stan.DefaultNatsURL),
    )
    if err != nil {
        log.Print(err)
        return
    }
    defer sc.Close()
    channel := event.Channel
    eventMsg := []byte(event.EventData)
    // Publish message on subject (channel)
    sc.Publish(channel, eventMsg)
    log.Println("Published message on channel: " + channel)
}
```

Subscribe Events

```
// Subscribe with manual ack mode, and set AckWait to 60 seconds
aw, _ := time.ParseDuration( s: "60s")
sc.Subscribe(channel, func(msg *stan.Msg) {
    msg.Ack() // Manual ACK
    order := pb.Order{}
    // Unmarshal JSON that represents the Order data
    err := json.Unmarshal(msg.Data, &order)
    if err != nil {
        log.Print(err)
        return
    }
    // Handle the message
    log.Printf( format: "Subscribed message from clientID - %s for Order: %+v\n", clientID, order)

}, stan.DurableName(durableID),
    stan.MaxInflight( m: 25),
    stan.SetManualAckMode(),
    stan.AckWait(aw),
)
```

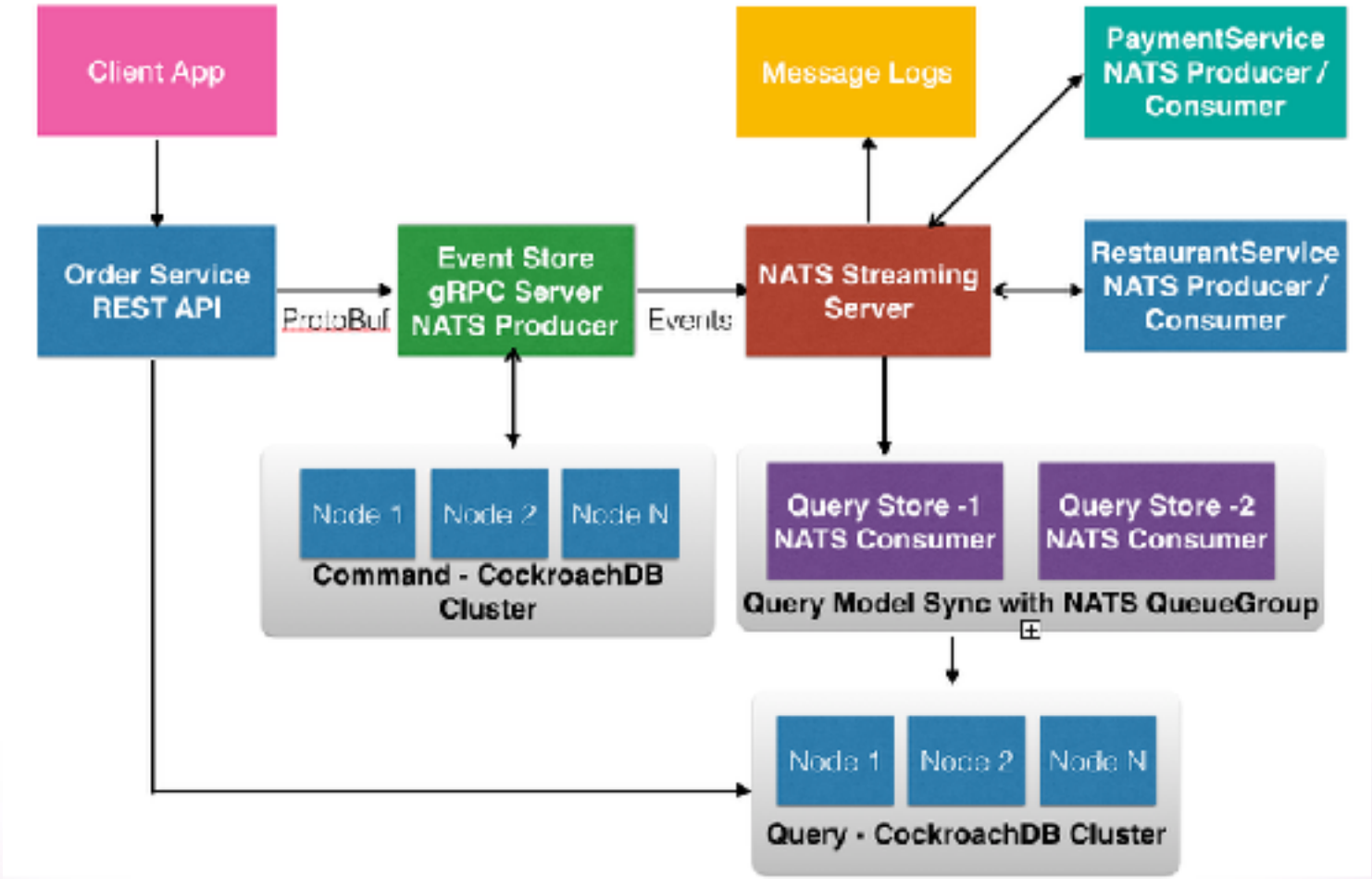
Subscribe with QueueGroup

```
const (
    clusterID  = "test-cluster"
    clientID   = "order-query-store1"
    channel    = "order-notification"
    durableID  = "store-durable"
    queueGroup = "order-query-store-group"
)
```

```
sc.QueueSubscribe(channel, queueGroup, func(msg *stan.Msg) {
    order := pb.Order{}
    err := json.Unmarshal(msg.Data, &order)
    if err == nil {
        // Handle the message
        log.Printf( format: "Subscribed message from clientID - %s: %+v\n", clientID, order)
        queryStore := store.QueryStore{}
        // Perform data replication for query model into CockroachDB
        err := queryStore.SyncOrderQueryModel(order)
        if err != nil {
            log.Printf( format: "Error while replicating the query model %+v", err)
        }
    }
}, stan.DurableName(durableID),
)
```


DEMO

1: Request for place an Order from client app to OrderService 2: OrderService calls to a gRPC API provided by EventStore to create events 3: Event Store persist event data into Event Store and publishes events via NATS streaming server 4: Microservices subscribe events published by Event Store 5: By subscribing events, application state is constructed by composing event data, query model creates read model for views. 6: Microservices reactive to events by subscribing events, and do its own actions and publish other set of events



https://github.com/shijuvar/gokit/tree/master/examples/nats-streaming


Thank you
saltmarch
GREAT INDIAN
DEVELOPER
SUMMIT
2018


GREAT INDIAN
DEVELOPER™
SUMMIT 2019
Conference : April 23-26, Bangalore
12th EDITION
#gids19
Register early and get the best discounts!
www.developersummit.com
@greatindiandev
bit.ly/gidslinkedin
facebook.com/gids19
bit.ly/saltmarchyoutube
flickr.com/photos/saltmarch/