Distributed, Event-Based Systems Using Java and NATS

rob signorelli
co-founder, chief technology officer, marketing intern, part-time accountant, assistant to the director of support, ...

bridgekit

what the heck is nats?

NATS is a suite of tools that enable communication in distributed systems. Your services have events/data that they need to share with other services - NATS provides easy to use tools and APIs to handle that communication.

It’s a message queue. It’s a cache. It’s an object store. It’s an event streaming platform. It’s a stock market predictor.*

Open Source member of CNCF

Written in Go

https://nats.io

* NATS is absolutely not a stock market predictor

why is nats awesome?

01

A distributed systems Swiss army knife

Why install, configure, manage, and write vastly different integrations to Redis, Kafka, ActiveMQ, and S3 when NATS does all of the above?

02

Lightweight and incredibly performant

Process millions of messages per second with minimal resources. Small size and sub-second startup makes it great for containerized apps and serverless as well.

03

Brain-dead simple to install and run

The entire suite of tools is a single 15MB static binary. It’s really as easy as download the binary and run “nats”. Configuration and APIs are equally as focused on simplicity.

time for some examples

- Demos written to highlight NATS interactions, that’s it.
- Error handling and validation is non-existent to make code a bit easier to read.
- No real business logic since it distracts from the actual topic we’re learning.
- No Spring Boot. I hate magic. I like main() functions and traceable code.

example pub/sub

A simple scenario where one service publishes “order.placed” events, and another remote service receives messages when that happens.

```
# In terminal A
make nats

# In terminal B
make demo-pubsub-publisher

# In terminal C
make demo-pubsub-subscriber
```

Similar Tools

ActiveMQ, RabbitMQ, Beanstalkd

example caching

A very crude rate-limiting scenario where one service stores request counts in the cache, and another periodically reads/reports on those totals. Data is ephemeral.

```
# In terminal A
make nats

# In terminal B
make demo-kv-cache-writer

# In terminal C
make demo-kv-cache-reader
```

Similar Tools

Redis/Valkey, Memcached

example remote config

You have settings/configs that need to be known at startup, but may change at runtime. The writer makes periodic changes to the state while the reader listens for and reacts to those changes.

Similar Tools

Consul, etcd, Zookeeper

```
# In terminal A
make nats

# In terminal B
make demo-kv-settings-writer

# In terminal C
make demo-kv-settings-reader
```

example object store

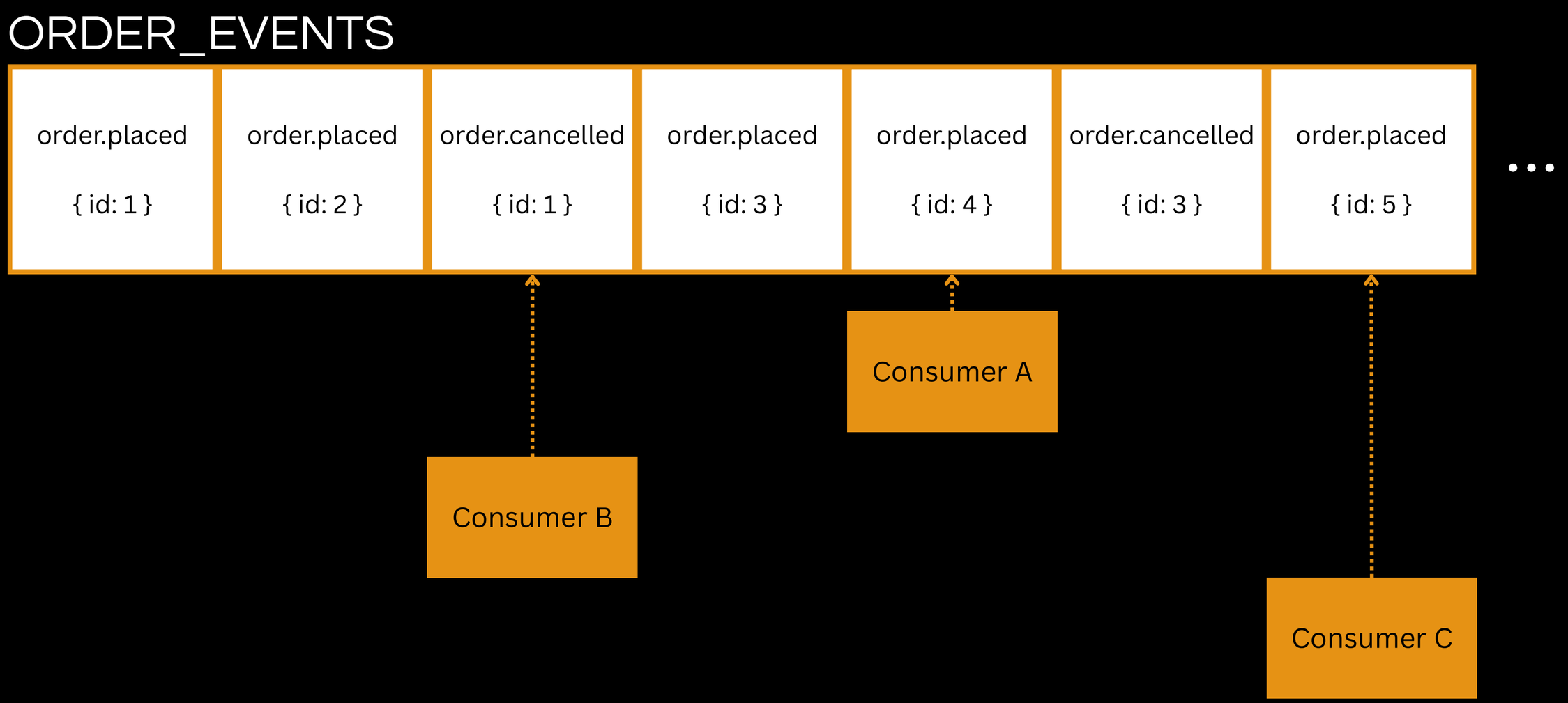
You’re saving database records that have some large file associated with them. Rather than storing those in the DB, keep the files in an object store with metadata that links back to the record. The reader app downloads the stored file to data/out.

Similar Tools

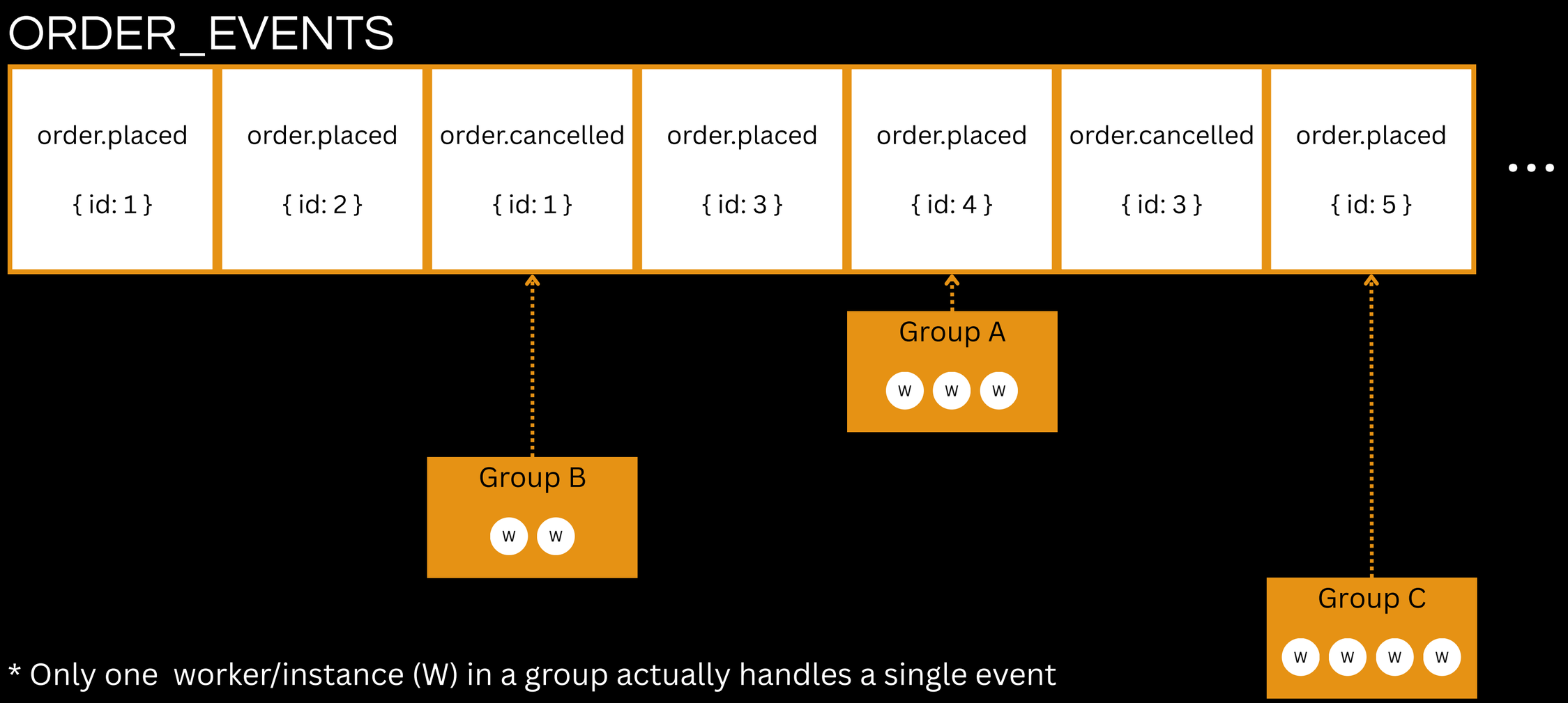
S3, Cloudflare R2, Any Database

```
# In terminal A
make nats

# In terminal B
make demo-objects-writer-a
make demo-objects-reader-a

# Or...
make demo-objects-writer-b
make demo-objects-reader-b

# Or...
make demo-objects-writer-c
make demo-objects-reader-c
```

overview event stream



overview consumer groups



* Only one worker/instance (W) in a group actually handles a single event

example event stream

An order processing/fulfillment program that creates/cancels orders. Those events trigger other workflow tasks in a durable, load-balanced fashion.

Similar Tools

Kafka, Redis Streams, Kinesis

```
# In terminal A
make nats

# In terminal B
make demo-stream-publisher

# In terminal C, D, and E
make demo-stream-consumer-confirm
make demo-stream-consumer-fulfill
make demo-stream-consumer-trash
```

```
Run multiple instances of the
consumers to see load balancing.

Shut down all consumers then restart
them to see catch-up behavior.
```

example complete app

A basic ordering platform with multiple services. Some rely on HTTP/API calls - others rely on events to invoke them.

Workflows like cancelling an order are loosely coupled, so the order service doesn't know about the confirmation email or the payment refund.

```
# In terminal A
make nats

# In terminal B
make demo-app

# ...or in separate terminals
make demo-app-api
make demo-app-events
make demo-app-events
make demo-app-events
```

helpful links

NATS Docs and Examples

https://docs.nats.io

https://natsbyexample.com

Examples Source Code

https://github.com/bridgekit-io/jug-nats-demo

Go Framework Based on NATS

https://github.com/bridgekit-io/frodo