# Hareflow: C++ Client for RabbitMQ Streams

November 2022

Yannis Rivard-Mulrooney

Software Developer, Coveo

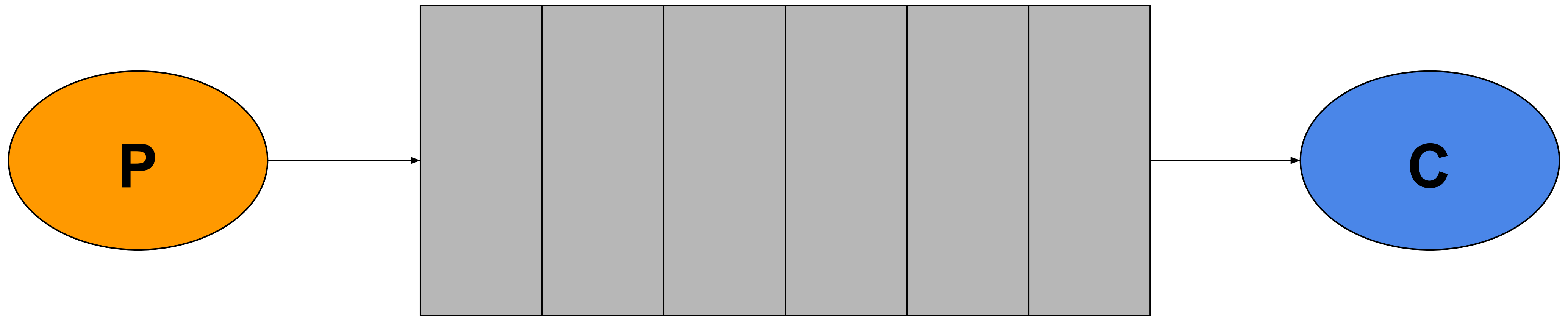
# Agenda

1. A brief description of RabbitMQ
2. What are RabbitMQ streams?
3. High level overview of the stream protocol
4. Hareflow: feature overview
5. Hareflow: backing tech and libraries
6. Simple usage examples
7. Questions?

# 1. A brief description of RabbitMQ

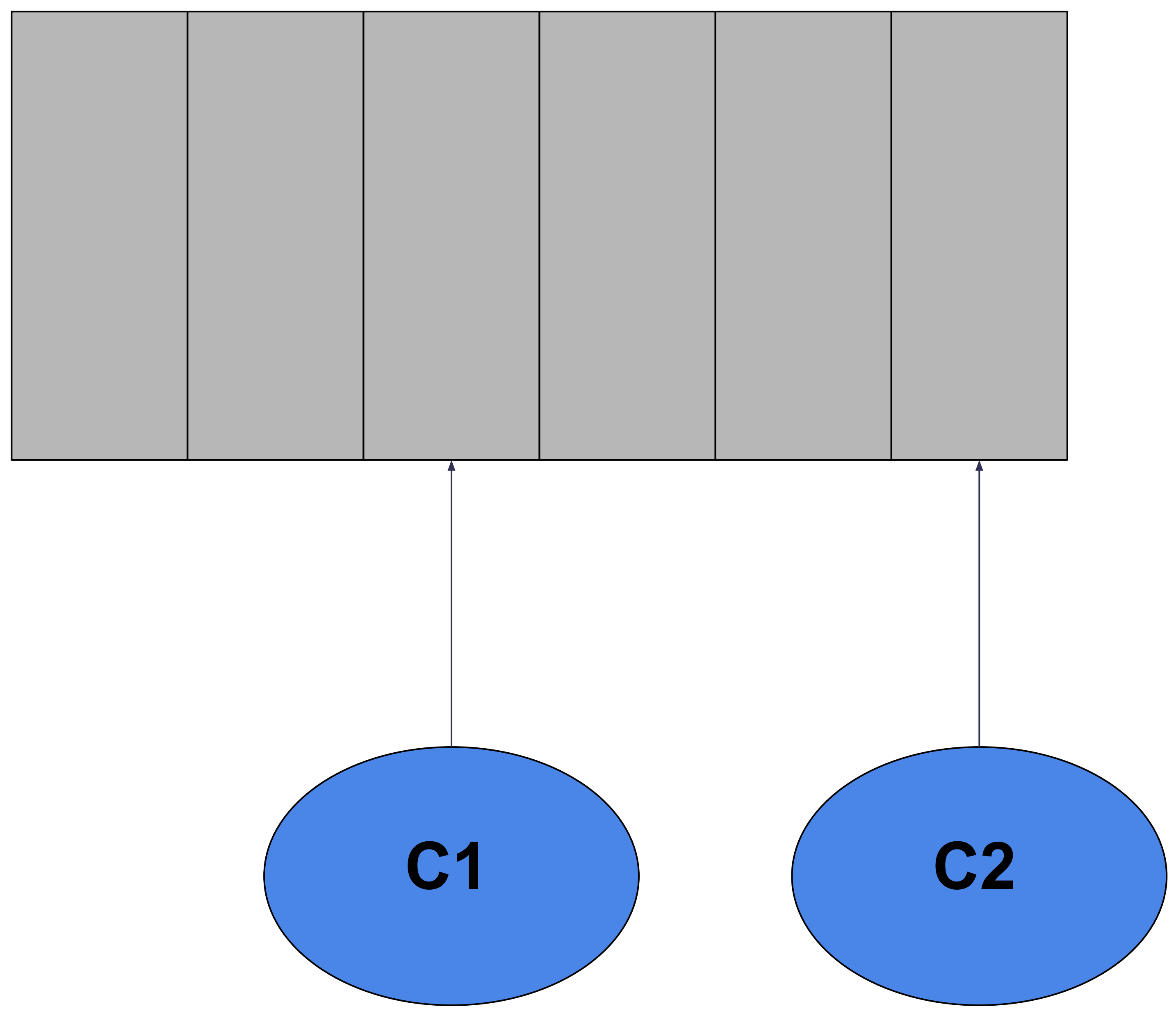
# RabbitMQ™

- Messaging broker
- Queuing semantics
- Can be run as a single server or cluster
- Flexible dispatching via exchanges
- Different types of backing queues
- Uses AMQP 0.9.1 protocol



# 2. What are RabbitMQ streams?

# RabbitMQ Streams



- Log-based data structure
- Non destructive consumption semantics
- Size limited by time or used space
- Consumers use offsets to track their position in the stream
- Enables replay/time travelling
- Suitable for large logs and/or fanouts

# RabbitMQ Streams

- Can be used by AMQP clients transparently*
- Very high throughput
- Very low broker memory usage
- Not really designed for consumer groups OOTB
  - Can use super streams (partitioned streams) to scale out

# 3. High level overview of stream protocol

# Protocol

- Binary, full duplex, frame based protocol
- Long lived tcp connections
- Mostly asynchronous
- Most commands contain an id to link request to response
- Most commands originate from client, but some are server initiated
- Can have multiple consumers or producers per connection

# Protocol

- Publisher:
  - Used to send messages
  - Messages are batched by default
  - Each message is individually confirmed by the server when persisted
- Subscription:
  - Attach to a stream to receive messages from an offset onwards
  - Offset can be managed by user or persisted server-side

# 4. Hareflow: feature overview

# Hareflow Features

- High level client, mostly turnkey
- Supports creation and deletion of streams
- Supports publishing and consuming of stream messages
- Respects protocol best practices
  - Targets leader node for publishing
  - Targets replica nodes for subscriptions

# Hareflow Features

- Transparent reconnection on broker or network failure
  - Will resend unconfirmed messages
  - Will resume consumption at offset where failure occurred
- Ability to automatically or manually manage subscription offsets
- Automatic message batching for publishers
- AMQP 1.0 codec for queue client interoperability
- Currently no support for super streams

# 5. Hareflow: backing tech and libraries

# Backing Tech

- Built on top of boost asio
  - Default `io_context` included, but can be set to a custom one
  - Default `io_context` uses its own threadpool
- Minimal logger included
  - Can log to stdout/stderr
  - Can dispatch to a custom log handling function
- Exposes a blocking API with callbacks for message handlers

# Backing Tech

- Requires compiler with C++17 support
  - Currently supported: GCC, Clang, MSVC
  - Uses standard threading primitives and concurrency control

⚠️ Not meant for resource-limited environments

⚠️ Aims for feature richness and ease of use, not minimalism

⚠️ Manages its own internal threads

# Backing Tech

- Integrated in the public vcpkg repository
  - Easy to use, just add it to your vcpkg.json
  - Recommended way of integrating hareflow
  - Takes care of transitive dependencies

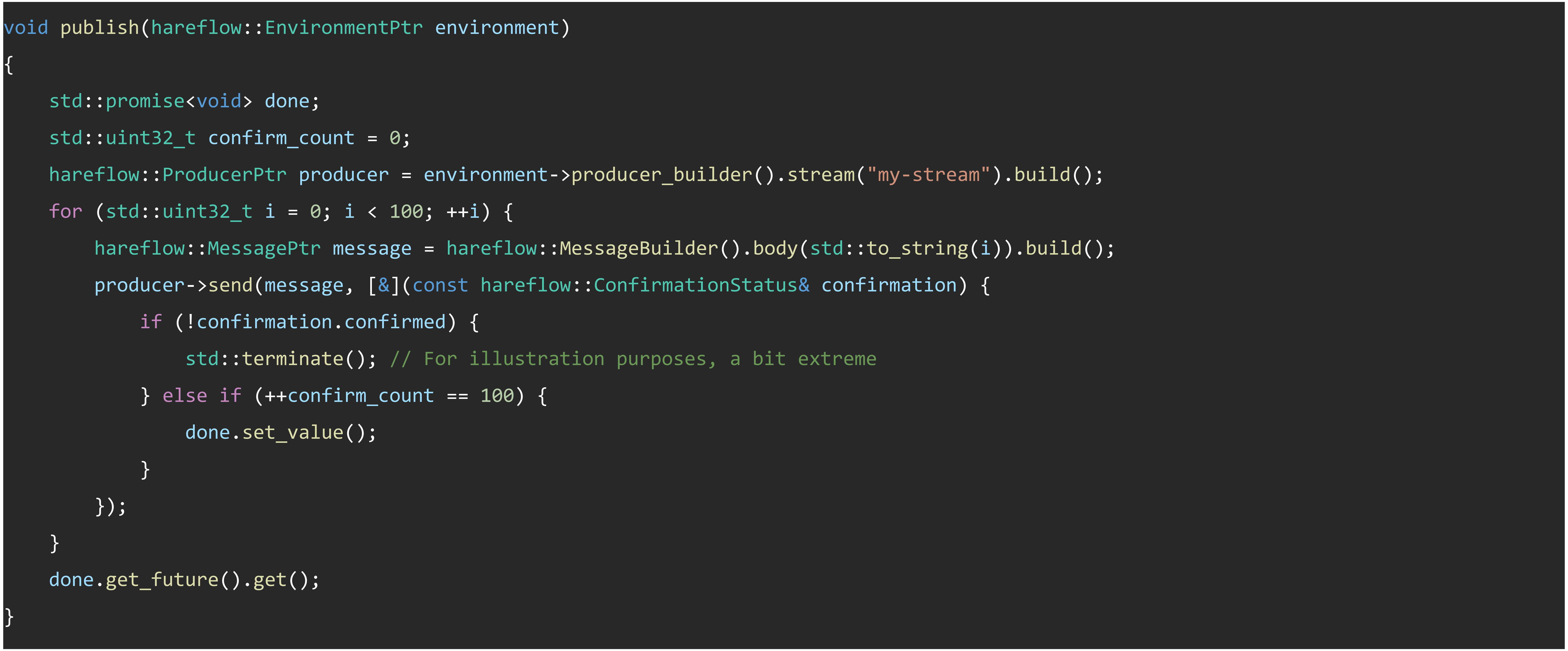
https://github.com/coveooss/hareflow

# 6. Simple usage examples

# Examples

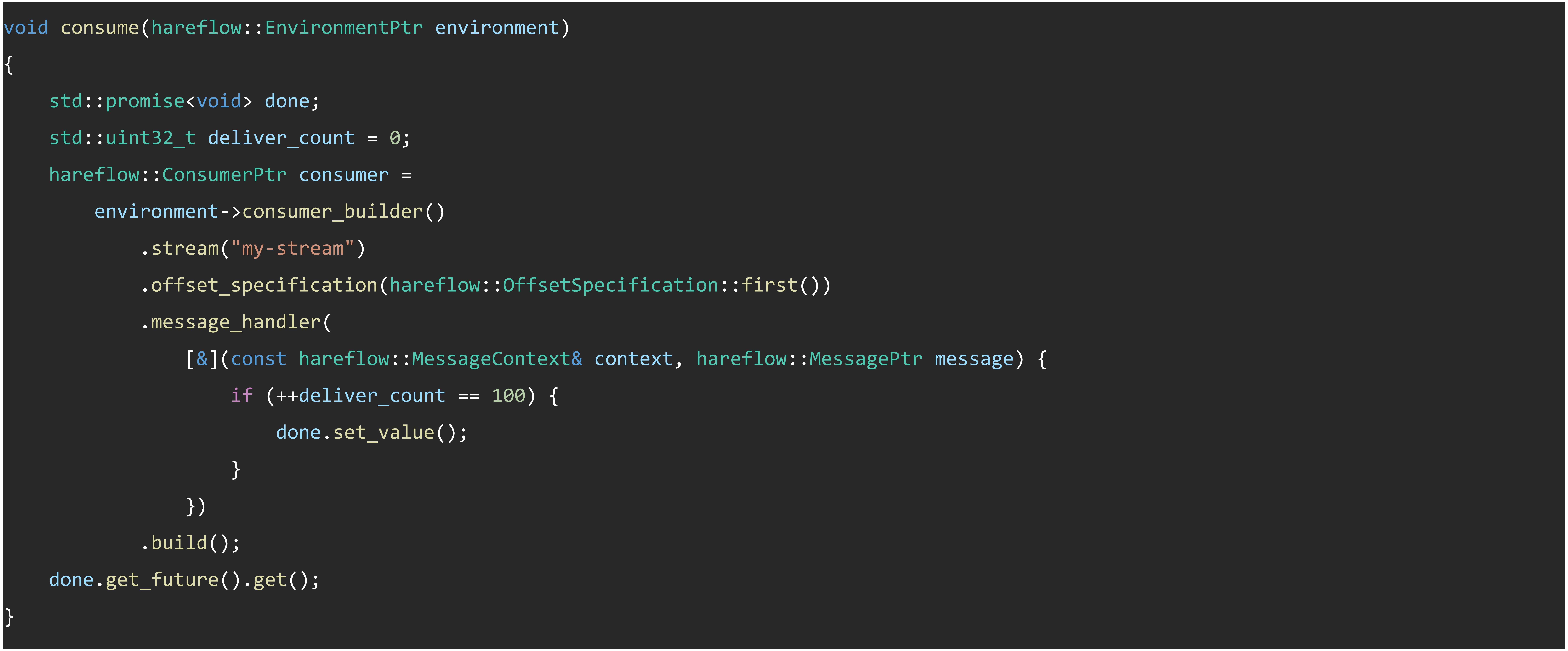
```cpp
int main()
{
    hareflow::EnvironmentPtr environment = hareflow::EnvironmentBuilder()
                                               .host("localhost")
                                               .username("guest")
                                               .password("guest")
                                               .use_ssl(true)
                                               .build();

    environment->stream_creator().stream("my-stream").max_age(std::chrono::hours(6)).create();
    environment->delete_stream("my-stream");

    return 0;
}
```

# Examples



```
void publish(hareflow::EnvironmentPtr environment)
{
    std::promise<void> done;
    std::uint32_t confirm_count = 0;
    hareflow::ProducerPtr producer = environment->producer_builder().stream("my-stream").build();
    for (std::uint32_t i = 0; i < 100; ++i) {
        hareflow::MessagePtr message = hareflow::MessageBuilder().body(std::to_string(i)).build();
        producer->send(message, [&](const hareflow::ConfirmationStatus& confirmation) {
            if (!confirmation.confirmed) {
                std::terminate(); // For illustration purposes, a bit extreme
            } else if (++confirm_count == 100) {
                done.set_value();
            }
        });
    }
    done.get_future().get();
}
```

# Examples



```
void consume(hareflow::EnvironmentPtr environment)
{
    std::promise<void> done;
    std::uint32_t deliver_count = 0;
    hareflow::ConsumerPtr consumer =
        environment->consumer_builder()
            .stream("my-stream")
            .offset_specification(hareflow::OffsetSpecification::first())
            .message_handler(
                [&](const hareflow::MessageContext& context, hareflow::MessagePtr message) {
                    if (++deliver_count == 100) {
                        done.set_value();
                    }
                })
            .build();
    done.get_future().get();
}
```

# 7. Questions?

# Thank you!