AMQP
For Flexible and Robust Messaging In Julia

•••

February 19, 2021

Tanmay Mohapatra, Julia Computing Inc.

TOC

AMQP - Advanced Message Queuing Protocol

AMQP

- Specifies
  - Wire Protocol
  - Logical Components of Message Broker
- Allows interoperable messaging between systems
- AMQP client can
  - connect with any AMQP compliant system
  - expect AMQP functionality to be available
- Versions: 0.9, 1.0
  - 0.9 (still quite prevalent) supported by AMQPClient.jl

AMQP Terminology (v0.9)

- **Broker**: Implements AMQP functionality
- **Virtual Hosts**: Broker side context
  - Logical brokers in a single broker instance
- **Connection**: Network connection to a broker (virtual host)
- **Channel**: Represents a client side context/communication thread
  - Single connection can multiplex multiple channels
- **Exchange**: Routes messages to queues depending on type of exchange
  - e.g. direct, fan out, or based on message attributes
- **Queue**: Buffers messages
  - Queue attributes determine durability of the queue
- **Message**: Blob of data, along with some attributes

AMQP Server (RabbitMQ)

- Managed
  - Amazon MQ - https://aws.amazon.com/amazon-mq/
  - CloudAMQP - https://www.cloudamqp.com
  - There are others as well
- Docker - https://hub.docker.com/_/rabbitmq
- Download and Other options: https://www.rabbitmq.com/download.html

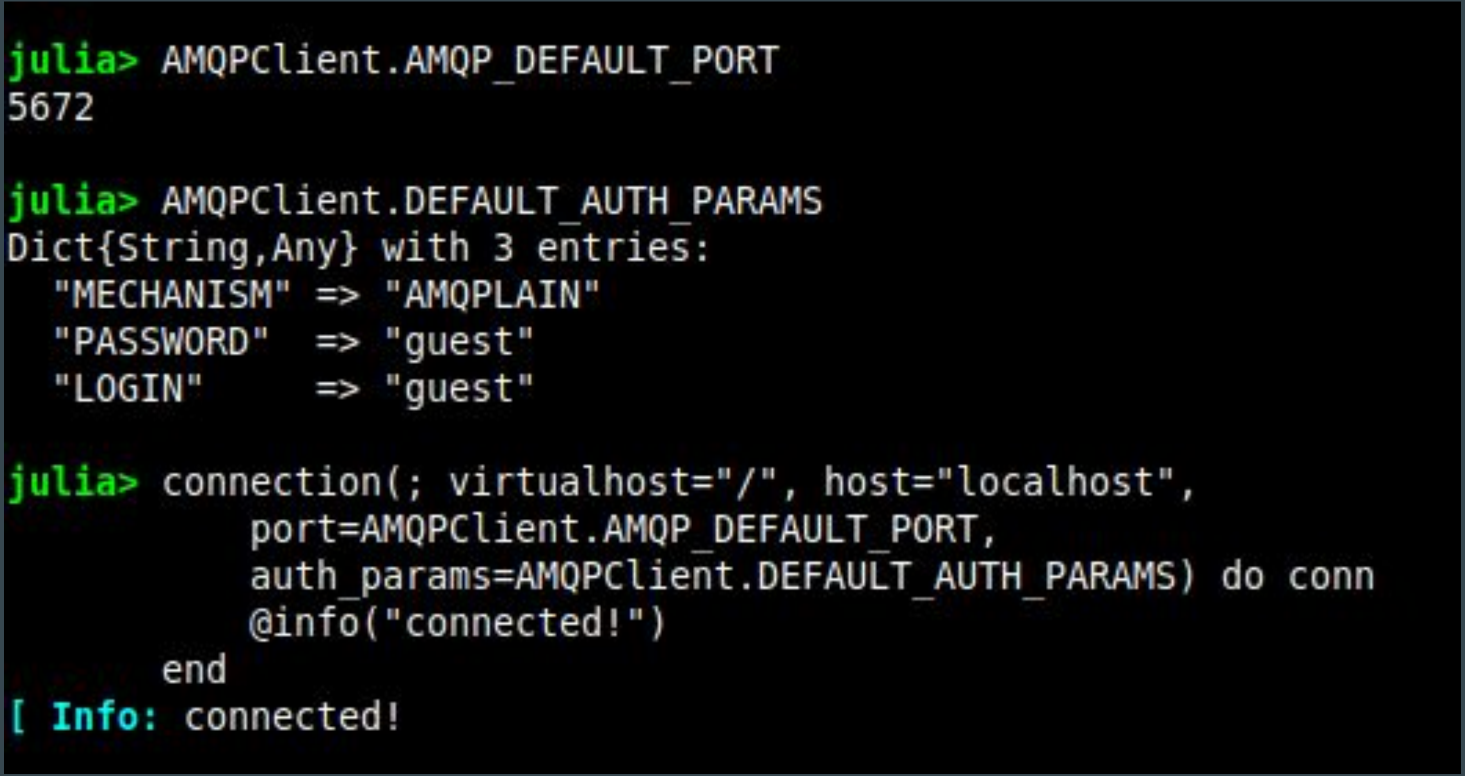
AMQPClient.jl

Using AMQPClient.jl

https://github.com/JuliaComputing/AMQPClient.jl

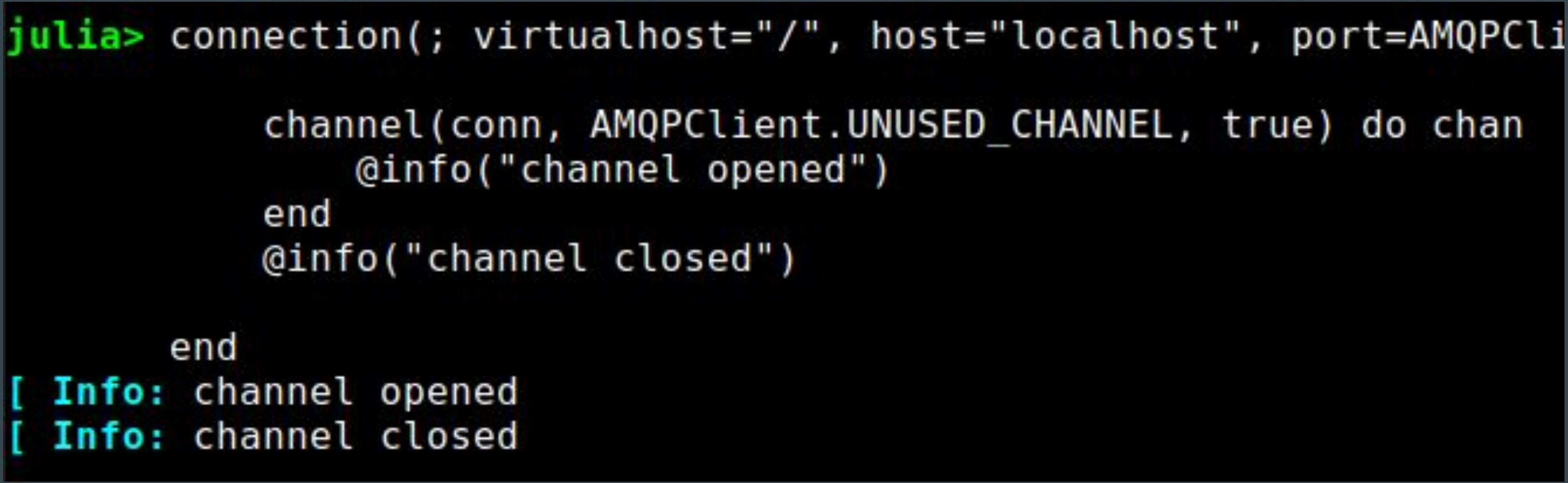
```
(@v1.5) pkg> up

(@v1.5) pkg> add AMQPClient

  Installed AMQPClient ── v0.4.1

julia> using AMQPClient
```

Connecting



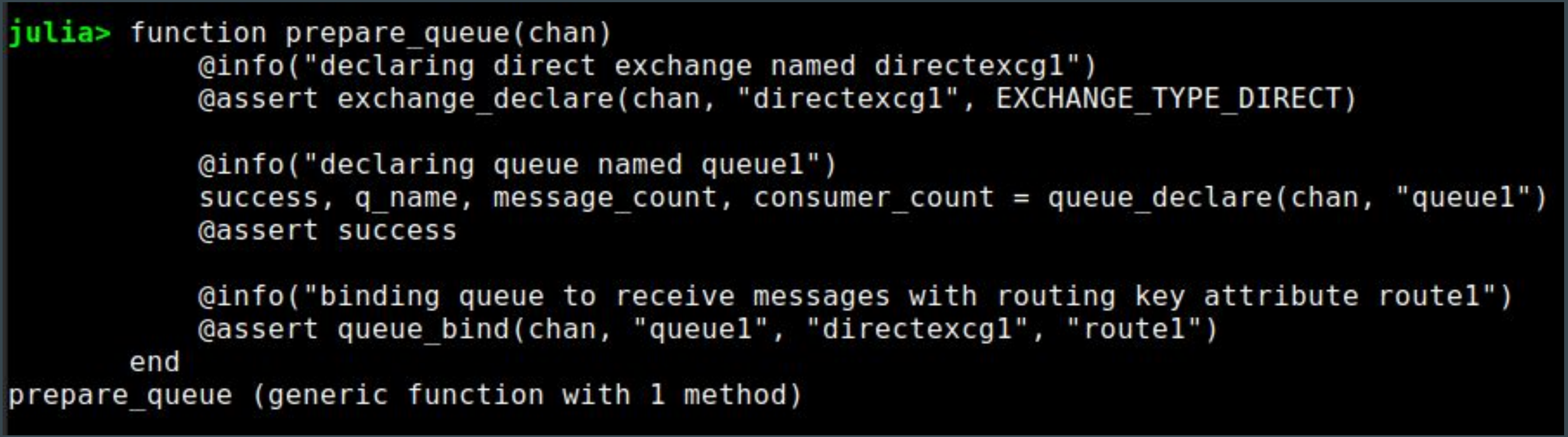
```
julia> AMQPClient.AMQP_DEFAULT_PORT
5672

julia> AMQPClient.DEFAULT_AUTH_PARAMS
Dict{String,Any} with 3 entries:
  "MECHANISM" => "AMQPLAIN"
  "PASSWORD"  => "guest"
  "LOGIN"     => "guest"

julia> connection(; virtualhost="/", host="localhost",
           port=AMQPClient.AMQP_DEFAULT_PORT,
           auth_params=AMQPClient.DEFAULT_AUTH_PARAMS) do conn
           @info("connected!")
       end
[ Info: connected!
```

Opening a Channel



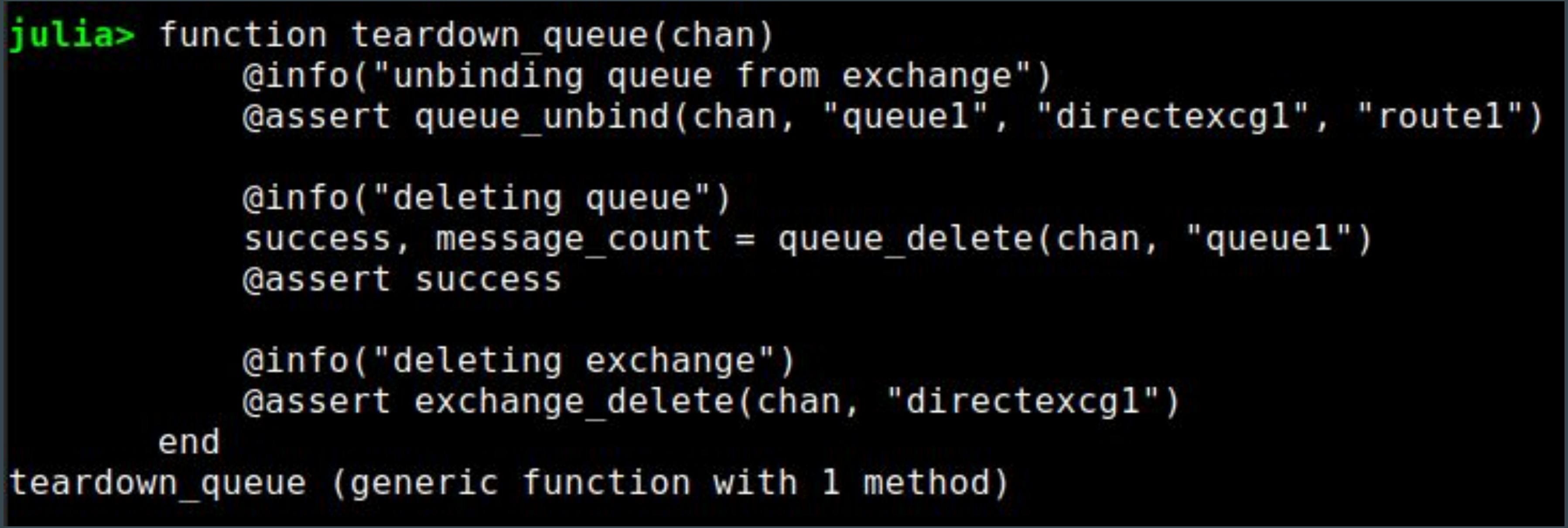
```
julia> connection(; virtualhost="/", host="localhost", port=AMQPCli

           channel(conn, AMQPClient.UNUSED_CHANNEL, true) do chan
               @info("channel opened")
           end
           @info("channel closed")

       end
[ Info: channel opened
[ Info: channel closed
```

Setting up Exchanges & Queues



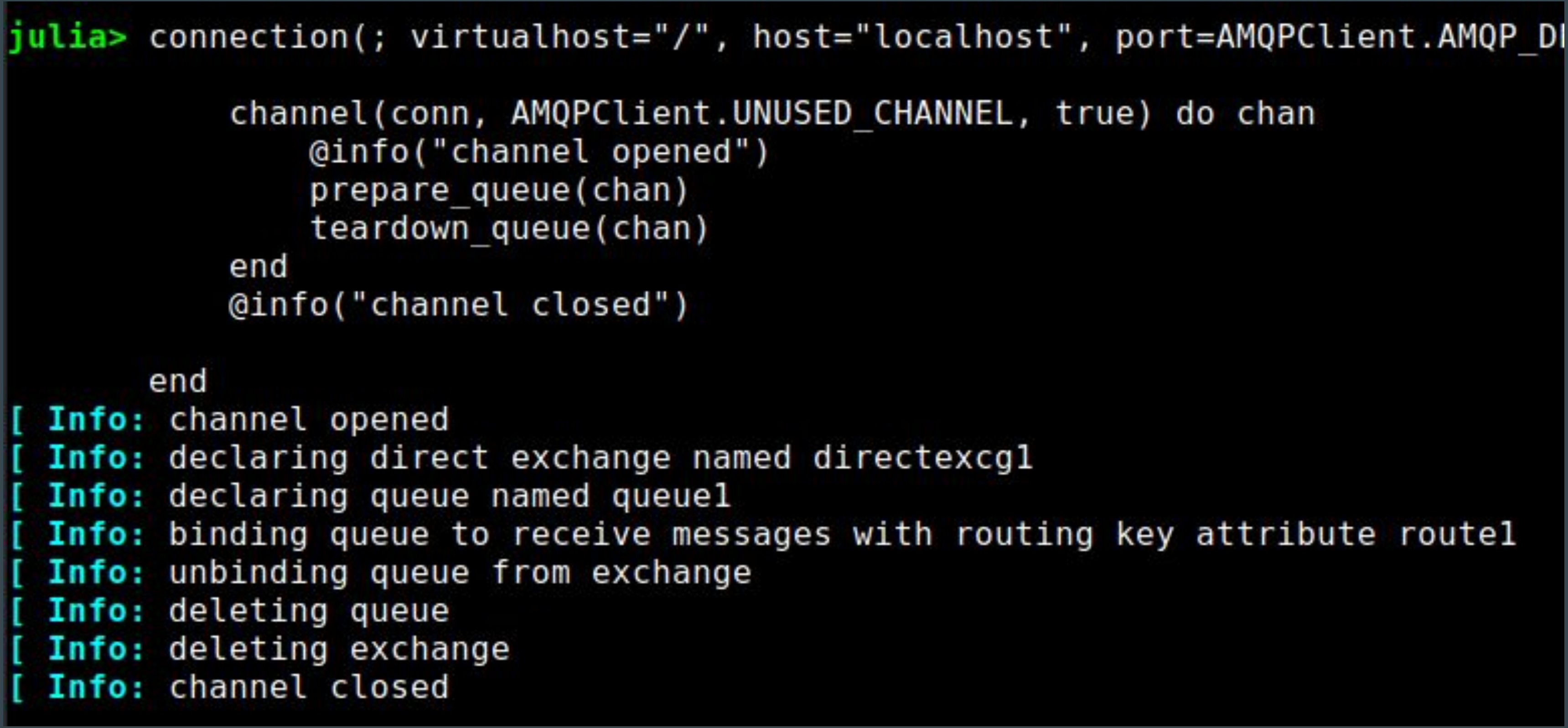
```
julia> function prepare_queue(chan)
           @info("declaring direct exchange named directexcg1")
           @assert exchange_declare(chan, "directexcg1", EXCHANGE_TYPE_DIRECT)

           @info("declaring queue named queue1")
           success, q_name, message_count, consumer_count = queue_declare(chan, "queue1")
           @assert success

           @info("binding queue to receive messages with routing key attribute route1")
           @assert queue_bind(chan, "queue1", "directexcg1", "route1")
       end
prepare_queue (generic function with 1 method)
```

Closing down Exchanges & Queues



```
julia> function teardown_queue(chan)
           @info("unbinding queue from exchange")
           @assert queue_unbind(chan, "queue1", "directexcg1", "route1")

           @info("deleting queue")
           success, message_count = queue_delete(chan, "queue1")
           @assert success

           @info("deleting exchange")
           @assert exchange_delete(chan, "directexcg1")
       end
teardown_queue (generic function with 1 method)
```

Exchanges & Queues



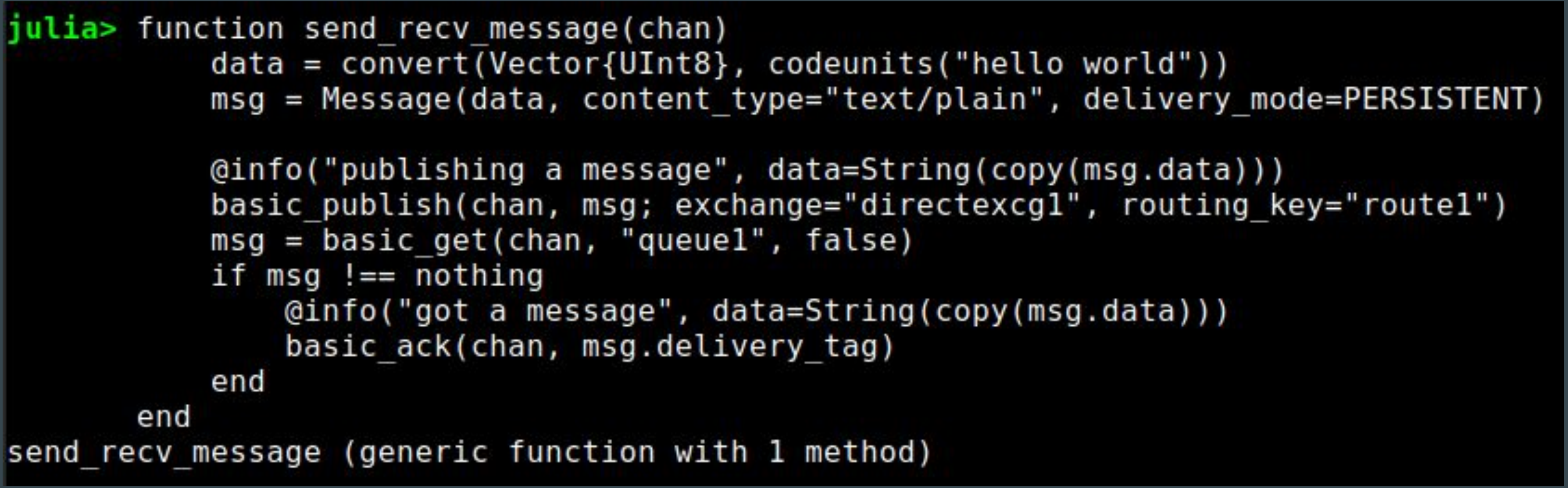
```
julia> connection(; virtualhost="/", host="localhost", port=AMQPClient.AMQP_DI

           channel(conn, AMQPClient.UNUSED_CHANNEL, true) do chan
               @info("channel opened")
               prepare_queue(chan)
               teardown_queue(chan)
           end
           @info("channel closed")

       end
[ Info: channel opened
[ Info: declaring direct exchange named directexcg1
[ Info: declaring queue named queue1
[ Info: binding queue to receive messages with routing key attribute route1
[ Info: unbinding queue from exchange
[ Info: deleting queue
[ Info: deleting exchange
[ Info: channel closed
```

Durability & Persistence

- Durable Exchanges & Queues survive broker restarts
- Messages can be marked Persistent

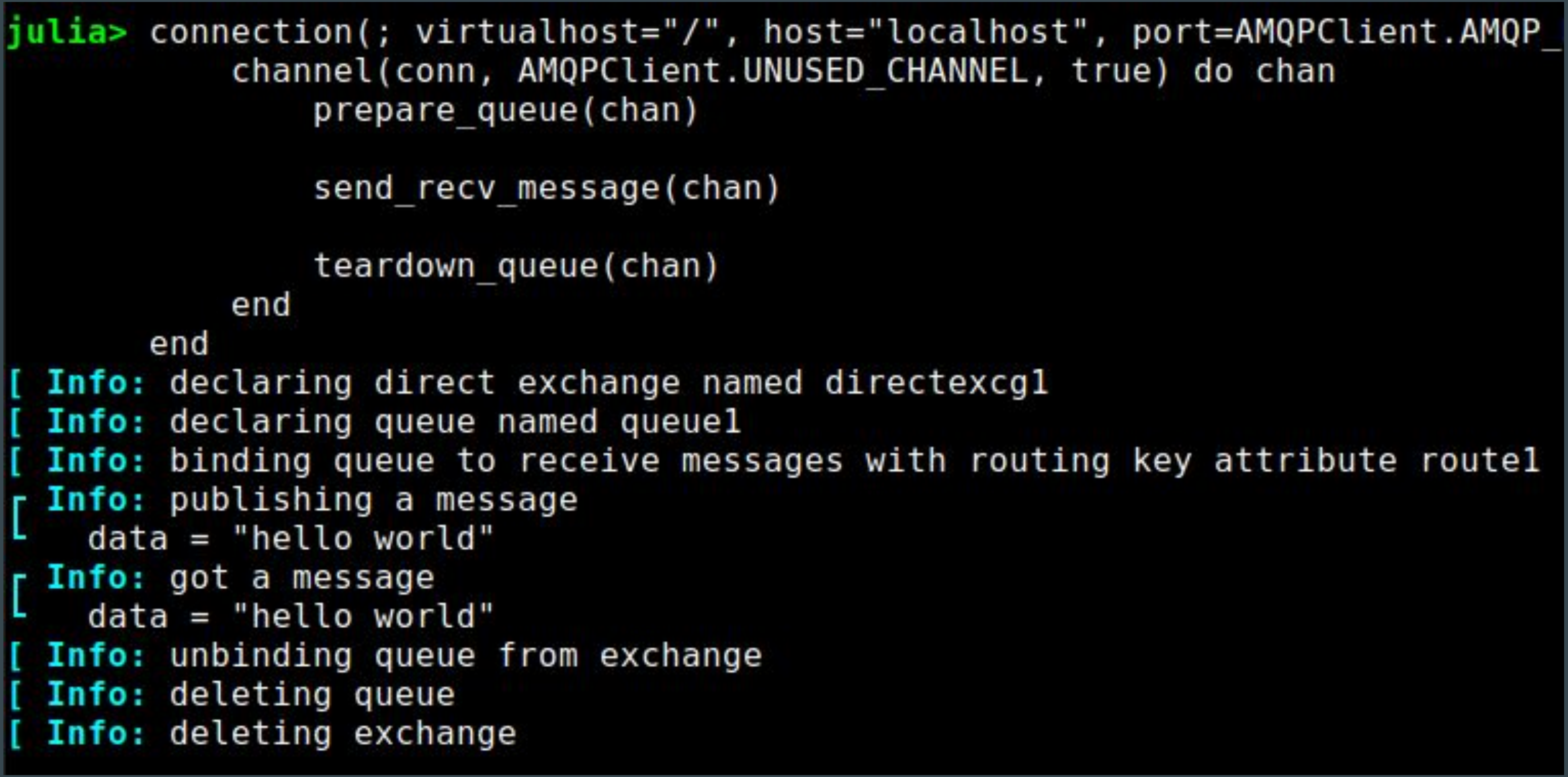
Persistent messages routed via durable exchanges & queues are reliable

Sending & Receiving Messages



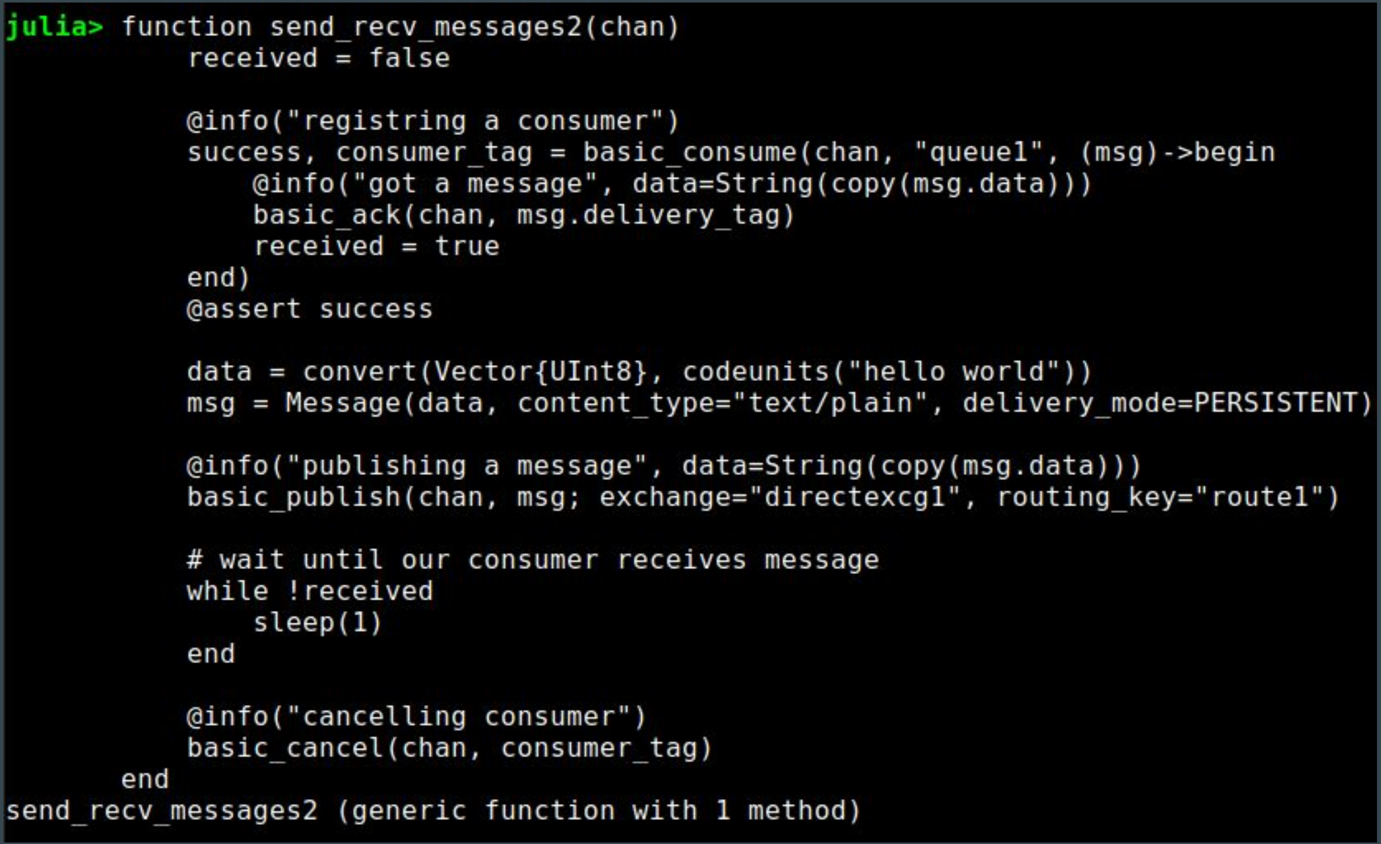
```
julia> function send_recv_message(chan)
           data = convert(Vector{UInt8}, codeunits("hello world"))
           msg = Message(data, content_type="text/plain", delivery_mode=PERSISTENT)

           @info("publishing a message", data=String(copy(msg.data)))
           basic_publish(chan, msg; exchange="directexcg1", routing_key="route1")
           msg = basic_get(chan, "queue1", false)
           if msg !== nothing
               @info("got a message", data=String(copy(msg.data)))
               basic_ack(chan, msg.delivery_tag)
           end
       end
send_recv_message (generic function with 1 method)
```

Sending & Receiving Messages



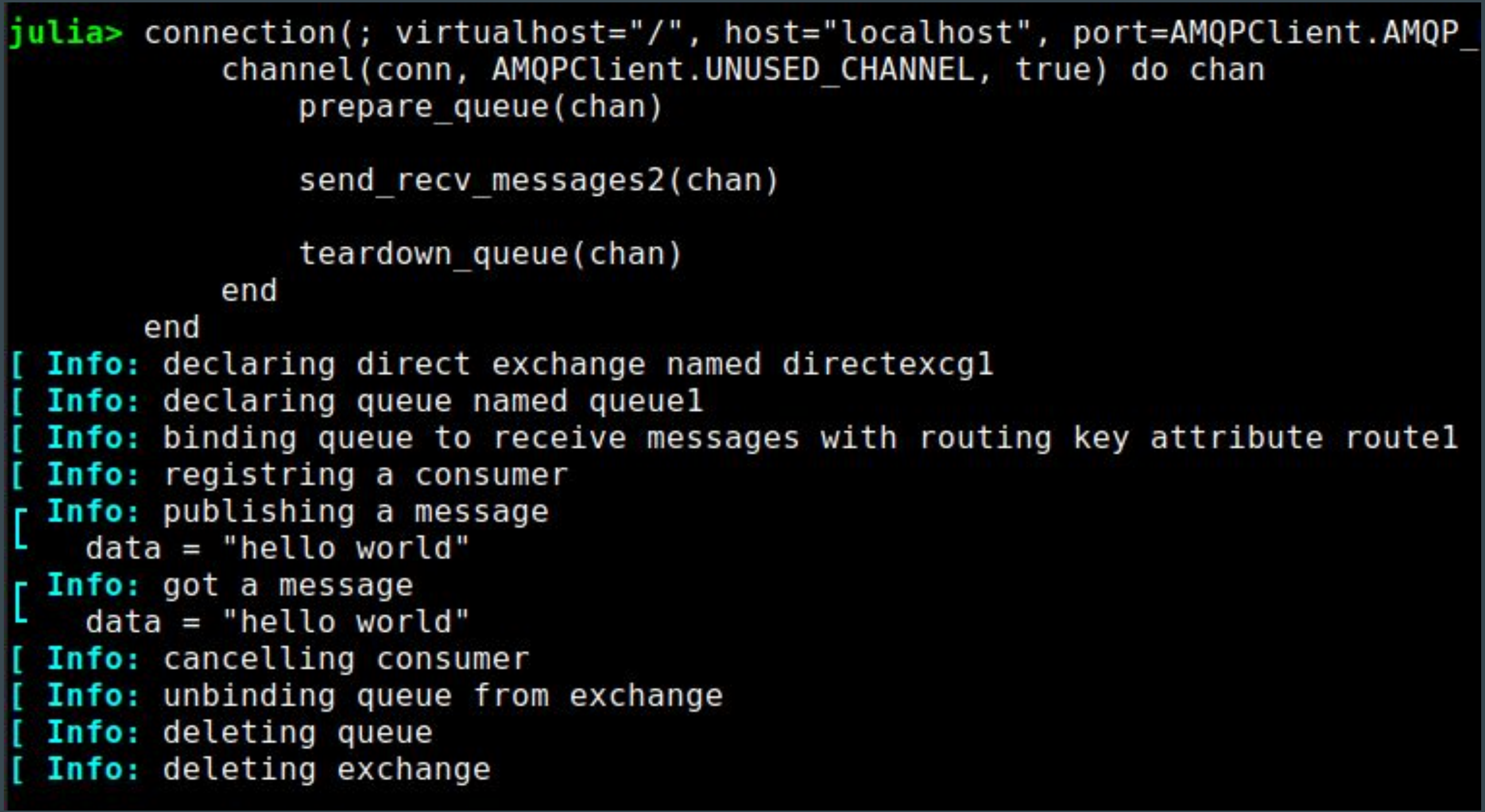
```
julia> connection(; virtualhost="/", host="localhost", port=AMQPClient.AMQP_
           channel(conn, AMQPClient.UNUSED_CHANNEL, true) do chan
               prepare_queue(chan)

               send_recv_message(chan)

               teardown_queue(chan)
           end
       end
[ Info: declaring direct exchange named directexcg1
[ Info: declaring queue named queue1
[ Info: binding queue to receive messages with routing key attribute route1
┌ Info: publishing a message
└   data = "hello world"
┌ Info: got a message
└   data = "hello world"
[ Info: unbinding queue from exchange
[ Info: deleting queue
[ Info: deleting exchange
```

Asynchronous Message Consumer



```
julia> function send_recv_messages2(chan)
           received = false

           @info("registring a consumer")
           success, consumer_tag = basic_consume(chan, "queue1", (msg)->begin
               @info("got a message", data=String(copy(msg.data)))
               basic_ack(chan, msg.delivery_tag)
               received = true
           end)
           @assert success

           data = convert(Vector{UInt8}, codeunits("hello world"))
           msg = Message(data, content_type="text/plain", delivery_mode=PERSISTENT)

           @info("publishing a message", data=String(copy(msg.data)))
           basic_publish(chan, msg; exchange="directexcg1", routing_key="route1")

           # wait until our consumer receives message
           while !received
               sleep(1)
           end

           @info("cancelling consumer")
           basic_cancel(chan, consumer_tag)
       end
send_recv_messages2 (generic function with 1 method)
```

Asynchronous Message Consumer

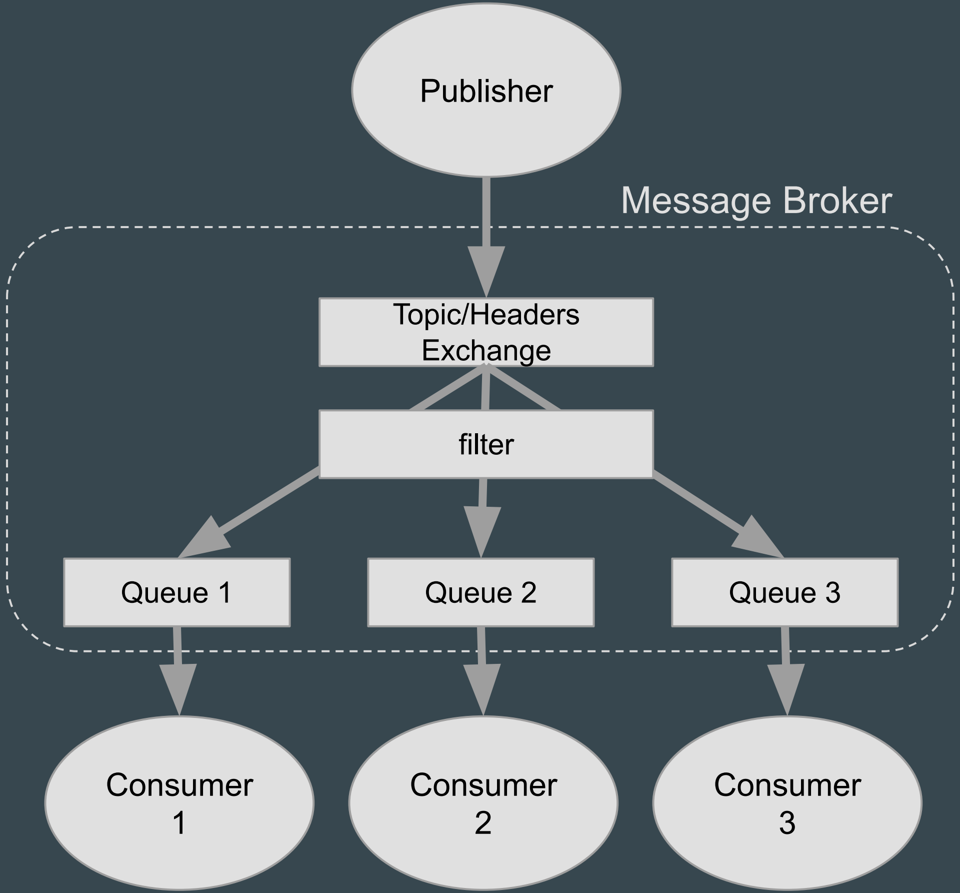


```
julia> connection(; virtualhost="/", host="localhost", port=AMQPClient.AMQP_
           channel(conn, AMQPClient.UNUSED_CHANNEL, true) do chan
               prepare_queue(chan)

               send_recv_messages2(chan)

               teardown_queue(chan)
           end
       end
[ Info: declaring direct exchange named directexcg1
[ Info: declaring queue named queue1
[ Info: binding queue to receive messages with routing key attribute route1
[ Info: registring a consumer
┌ Info: publishing a message
└   data = "hello world"
┌ Info: got a message
└   data = "hello world"
[ Info: cancelling consumer
[ Info: unbinding queue from exchange
[ Info: deleting queue
[ Info: deleting exchange
```

Some Useful Messaging Patterns

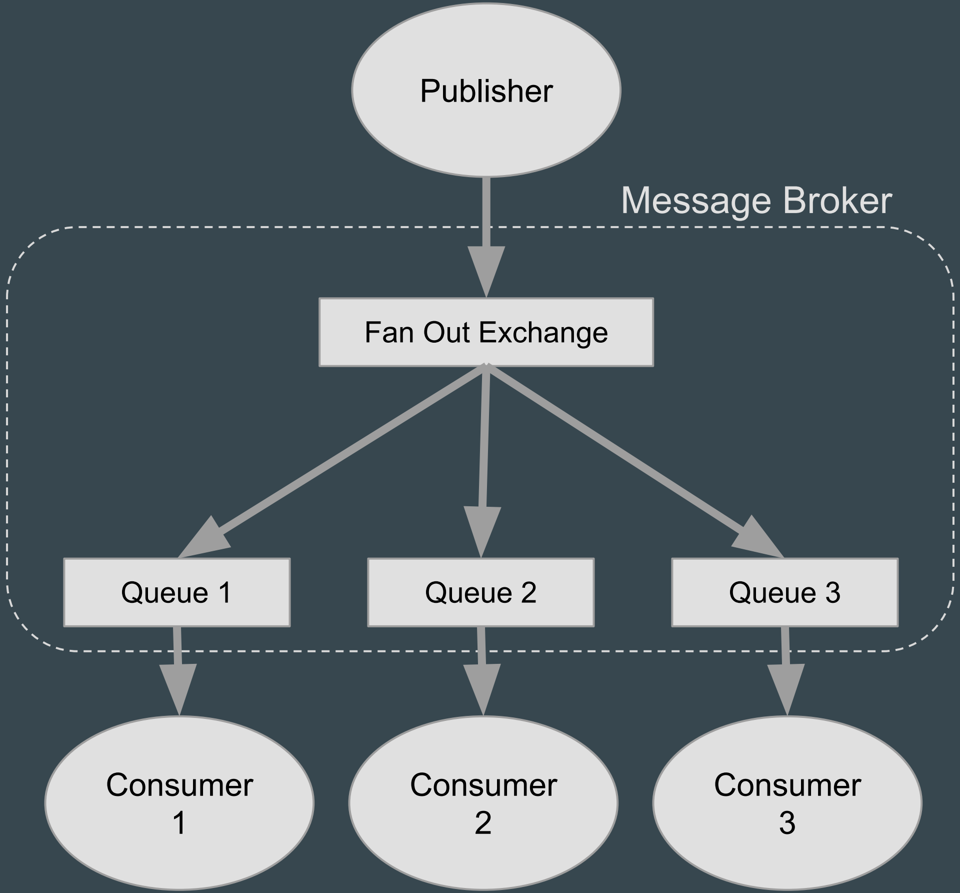
Publish Subscribe

- One to Many
- Publish tagged messages to exchange
- Receive messages matching tags subscribed to
- Filtered by:
    - routing key - Direct Exchange
    - routing key patterns - Topic Exchange
    - Header patterns - Headers Exchange



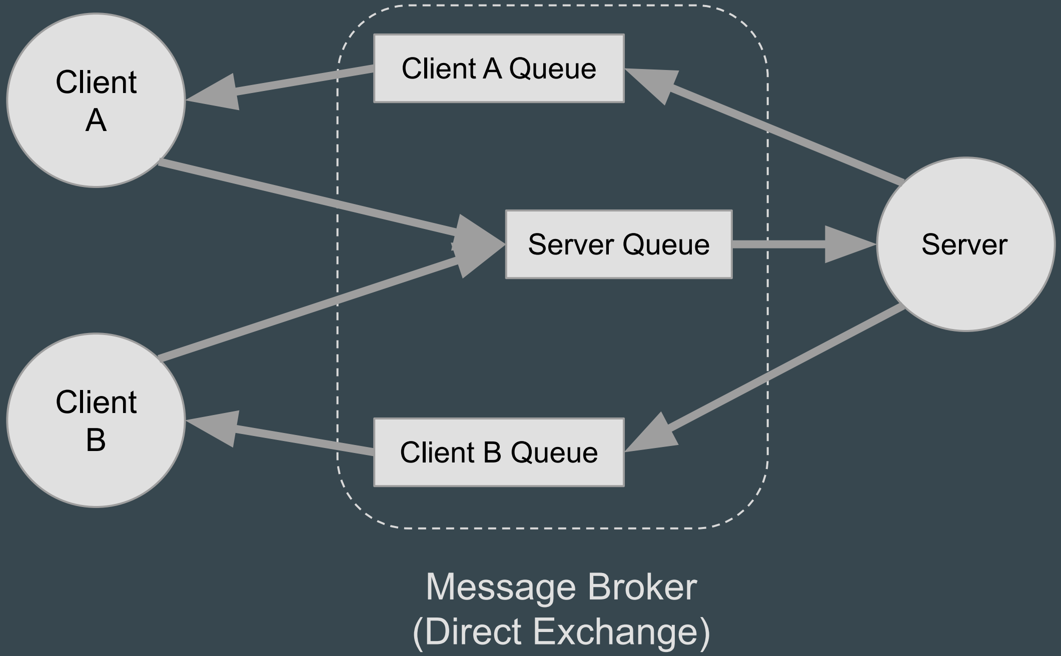
Fan Out

- Independent & Simultaneous
- Routing keys and patterns ignored



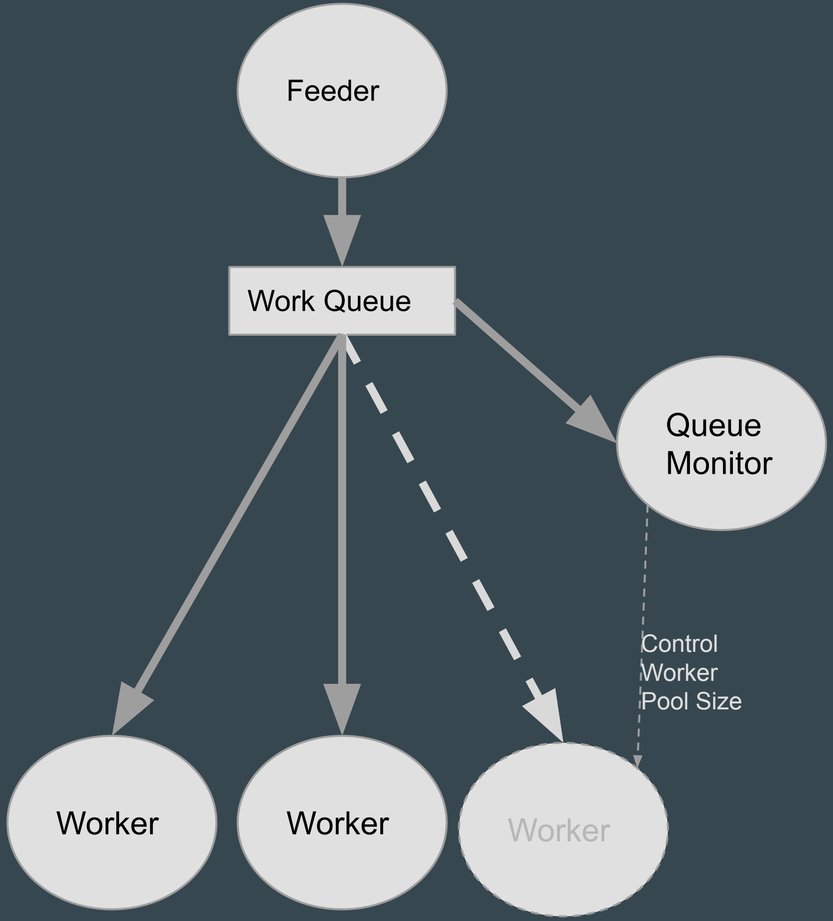
RPC

- Remote Procedure Call
- Request - Response
- Response must come back to requesting process/context
- Message attributes used:
  - correlation_id
  - reply_to



Work Queues

- Work queued
- One or more workers
  - Consume from queue
  - Acknowledge message after processing
- Failed work (unacknowledged) re-queued
- Control parallelism based on queue size

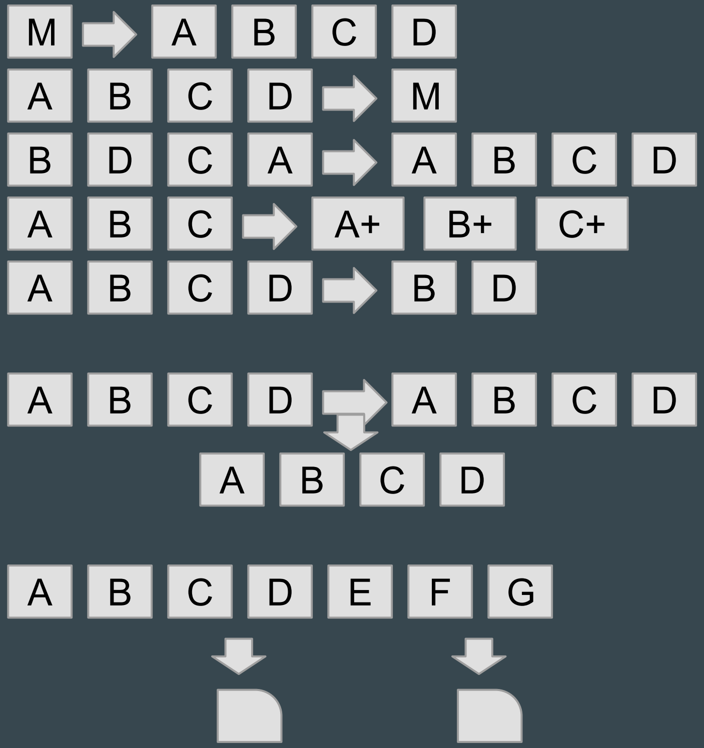


Other Patterns

- Splitter
- Aggregator
- Sequencer
- Enricher
- Filter

- Wiretap

- Service Activator



Q & A