Using RabbitMQ for Cross-language Interprocess Communication

Keith Bisset
keith@light.house

PETZL
SHERIFF
SEARCH & RESCUE

What is RabbitMQ?

High-Scalability, High-Availability Message Broker

Asynchronous Messaging

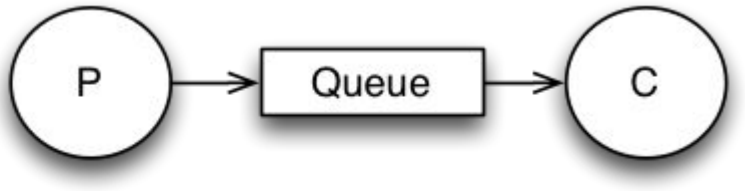
Hello World

```
int main()
{
  auto channel = AmqpClient::Channel::Create();
  const std::string queuename{"hello"};

  channel->DeclareQueue(queuename);
  auto msg = AmqpClient::BasicMessage::Create("Hello, World!");
  channel->BasicPublish("", queuename, msg);

  std::string consumer_tag = channel->BasicConsume("");

  AmqpClient::Envelope::ptr_t envelope;
  channel->BasicConsumeMessage(consumer_tag, envelope);

  std::cout << envelope->Message()->Body() << std::endl;
}
```

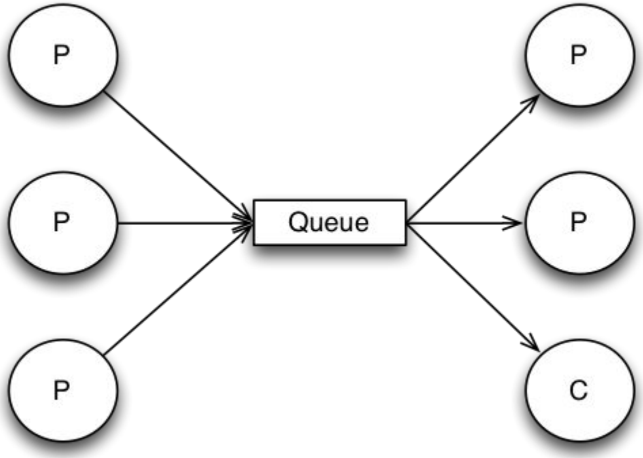
Hello World++

```
auto channel = AmqpClient::Channel::Create();
const std::string queuename{"hello"};
channel->DeclareQueue(queuename);

std::thread producer([channel, queuename](){
     for (int i=0; i < 5; i++) {
       auto msg = AmqpClient::BasicMessage::Create("Hello, World! " + std::to_string(i));
       channel->BasicPublish("", queuename, msg);
     }});

std::thread consumer([channel](){
     std::string consumer_tag = channel->BasicConsume("");
     while (true) {
       AmqpClient::Envelope::ptr_t envelope;
       channel->BasicConsumeMessage(consumer_tag, envelope);
     }});
```

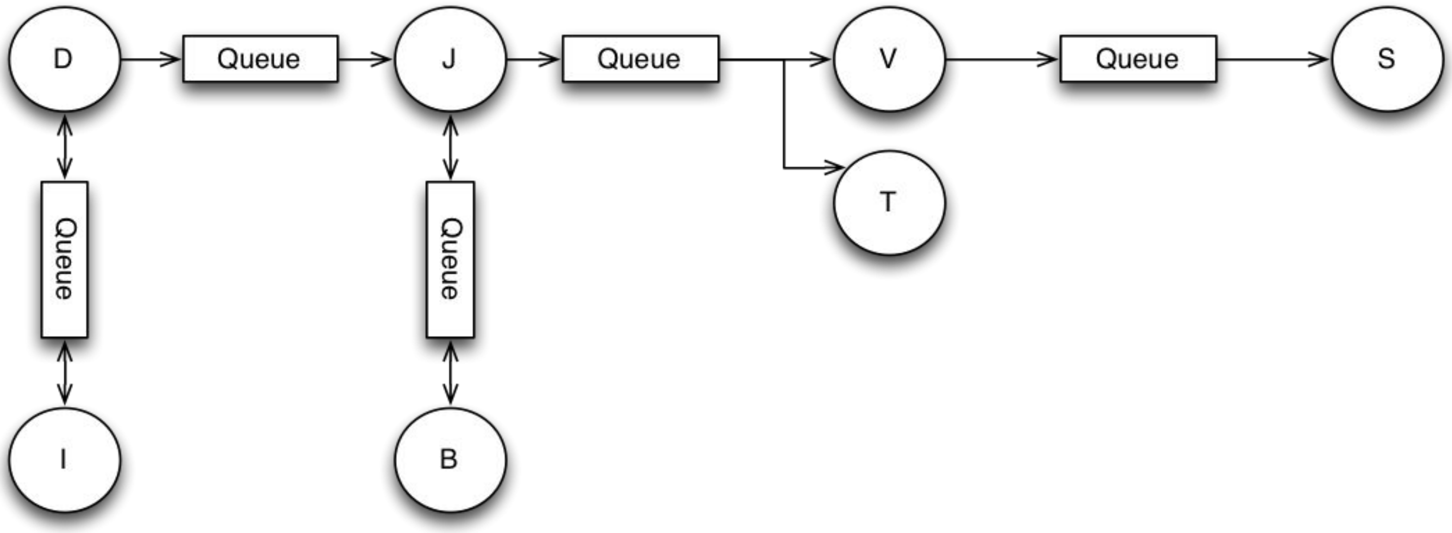
Architecture



Architecture



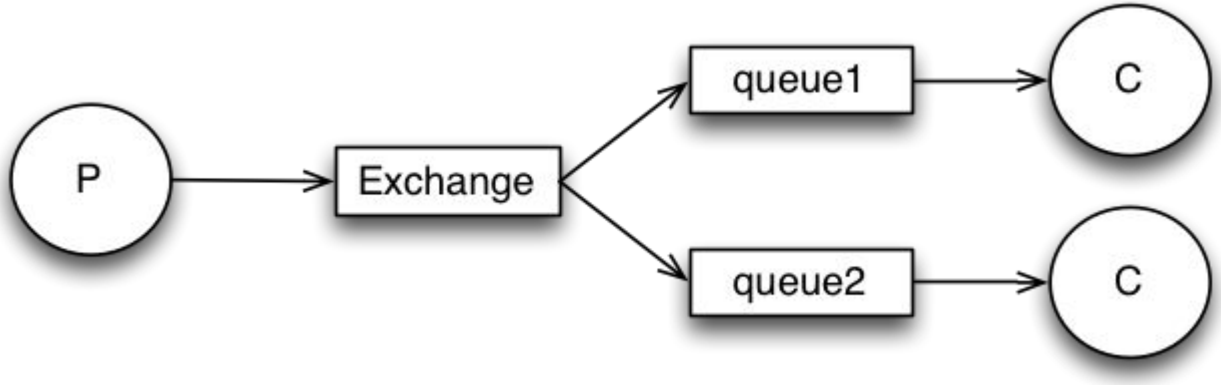
Architecture



Exchanges

All messages get Published to an Exchange which routes the message to one or more queues

Defaults to direct exchange (amq.direct) that routes message the queue with a specific name



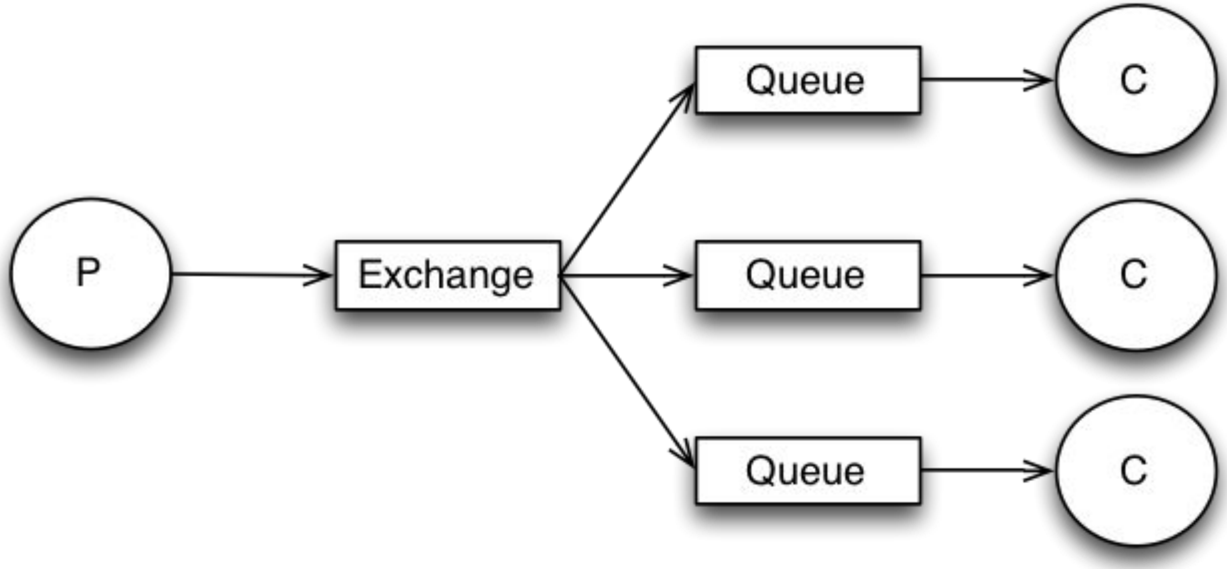
Direct Exchange

```
int main()
{
  auto channel = AmqpClient::Channel::Create();
  const std::string queueName{"hello"};
  const std::string exchangeName{"helloX"};
  channel->DeclareExchange(exchangeName);
  channel->DeclareQueue(queuename);
  auto msg = AmqpClient::BasicMessage::Create("Hello, World!");
  channel->BasicPublish(exchangeName, queuename, msg);

  std::string consumer_tag = channel->BasicConsume("");

  AmqpClient::Envelope::ptr_t envelope;
  channel->BasicConsumeMessage(consumer_tag, envelope);

  std::cout << envelope->Message()->Body() << std::endl;
}
```

Fanout Exchange

A fanout exchange routes the message to all attached queues

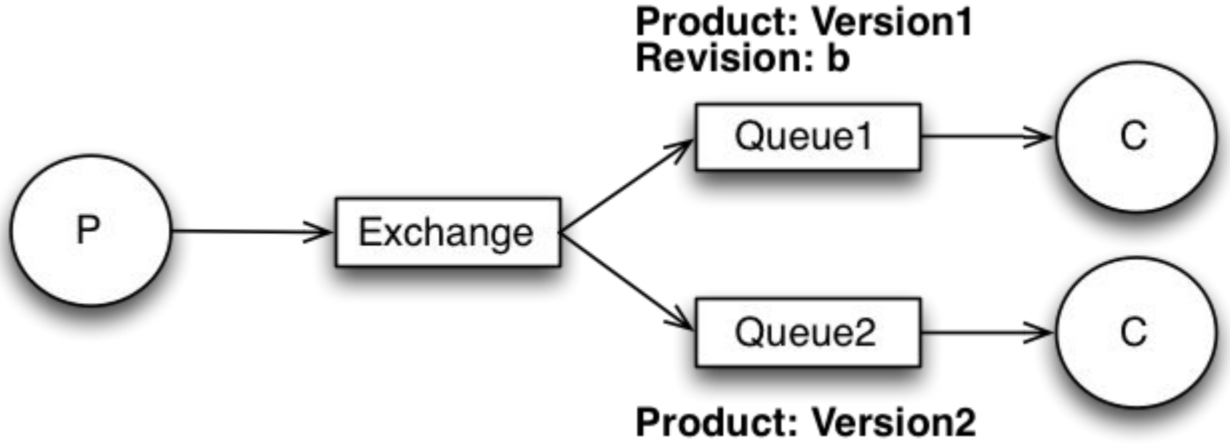
Queues can be added and removed as needed



Header Exchange

The producer can add headers (key, value pairs) to a message that are used to route the message to any matching queues

Special header *x-match* that can be *any* or *all* to specify how many headers must match



Header Exchange - Binding

```
auto channel = AmqpClient::Channel::Create();
const std::string queueName1{"v1"}, queueName2{"v2"};
const std::string exchangeName{"helloX"};
channel->DeclareExchange(exchangeName, HEADER);
channel->DeclareQueue(queueName1); channel->DeclareQueue(queueName2);

AC::Table headers1, headers2;
headers1.insert(AC::TableEntry("product", "version1"));
headers1.insert(AC::TableEntry("rev", "b"));
channel->BindQueue(queueName1, exchangeName, routing_key, headers1);

headers2.insert(AC::TableEntry("product", "version2"));
channel->BindQueue(queueName2, exchangeName, routing_key, headers2);
```

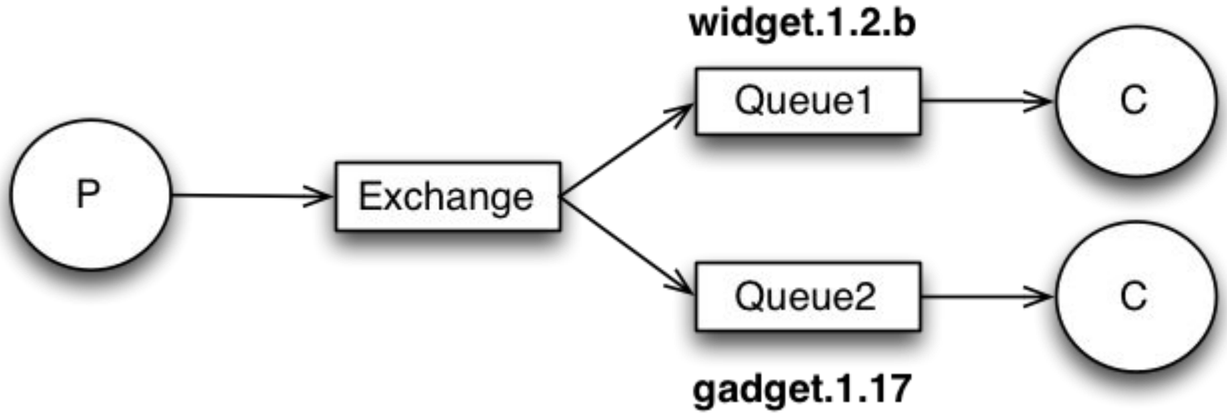
Header Exchange - Publishing

```
auto msg = AmqpClient::BasicMessage::Create("Hello, World!");
AC::Table headers1, headers2;
headers1.insert(AC::TableEntry("product", "version1"));
headers1.insert(AC::TableEntry("rev", "b"));

msg.HeaderTable(headers1);

channel->BasicPublish("", queuename, msg)
```

Topic Exchange

The producer specifies a routing key in dotted notation to match the routing pattern on one or more queues. '*' matches any single word, '#' matches zero or more words

widget.1.*.* -> Queue1
widget.1.# -> Queue1
*.1.# -> Queue1, Queue2
widget.1.* -> dropped



Dead Letter Exchange

Normally, if a message doesn't match any queues, it is silently dropped or returns an error to the client.

RabbitMQ provides an extension called the Dead Letter Exchange, that receives any undeliverable messages.

Channel::Create()

```
 std::string host{"localhost"};
 int port{5672}
 std::string username{"guest"};
 std::string password{"guest"};
 std::string vhost{"/"};

AmqpClient::Channel::Create(host, port, username, password, vhost);
```

Channel::DeclareExchange()

```
// How the broker should react if the exchange does not exist.  If passive and the exchange
// does not exist the broker will respond with an error and not create the exchange
bool passive{false};

// will the exchange survive a broker restart
bool durable{false};

// Are we the only client that can use the exchange.
// An exclusive exchange is deleted when the connection is closed
bool exclusive{false};

// auto_delete the exchange will be deleted after at least one exchange has been bound to
// it, then has been unbound
bool auto_delete{false};

// exchange_type the type of exchange to be declared
const std::string exchange_type = AmqpClient::Channel::EXCHANGE_TYPE_DIRECT;
channel->DeclareExchange(exchangeName, exchange_type, passive, durable, auto_delete);
```

Channel::DeclareQueue()

```
// How the broker should react if the queue does not exist.  If passive and the queue does
not exist the broker will respond with an error and not create the queue
bool passive{false};

// will the queue survive a broker restart
bool durable{false};

// Are we the only client that can use the queue.
// An exclusive queue is deleted when the connection is closed
bool exclusive{true};

// auto_delete the queue will be deleted after at least one exchange has been bound to it,
then has been unbound
bool auto_delete{true};

channel->DeclareQueue(queue, passive, durable, exclusive, auto_delete);
```

Channel::BasicPublish()

```
// Is the message required to be delivered to a queue. A MessageReturnedException is thrown
// if the message cannot be routed to a queue.
bool mandatory{false};

// Is the message required to be both routed to a queue, and immediately delivered via a
// consumer. If the message is not routed, or a consumer cannot immediately deliver the
// message a MessageReturnedException is thrown.
bool immediate{false};

BasicPublish(exchangeName, queueName, msg, mandatory, immediate);
```

Channel::BasicConsume()

```
// Enable context between producer and consumer
std::string consumerTag{""};

// Don't deliver messages sent on this channel
bool no_local{true};

// If true, ack'ing the message is automatically done when the message is delivered.
bool no_ack{true};

// Only this consumer can access the queue
bool exclusive{true};

// number of unacked messages the broker will deliver
int message_prefetch_count{1};

BasicConsume(queueName, consumerTag, no_local, no_ack, exclusive, message_prefech_count);
```

Channel::BasicConsumeMessage()

```
// Message consumed
AmqpClient::Envelope::ptr_t &envelope;

// timeout in milliseconds. 0 - nonblocking read, -1 - infinite
int timeout{-1};

bool success = BasicConsumeMessage(Envelope::ptr_t &envelope, int timeout = -1);
```

w

High Availability - client side

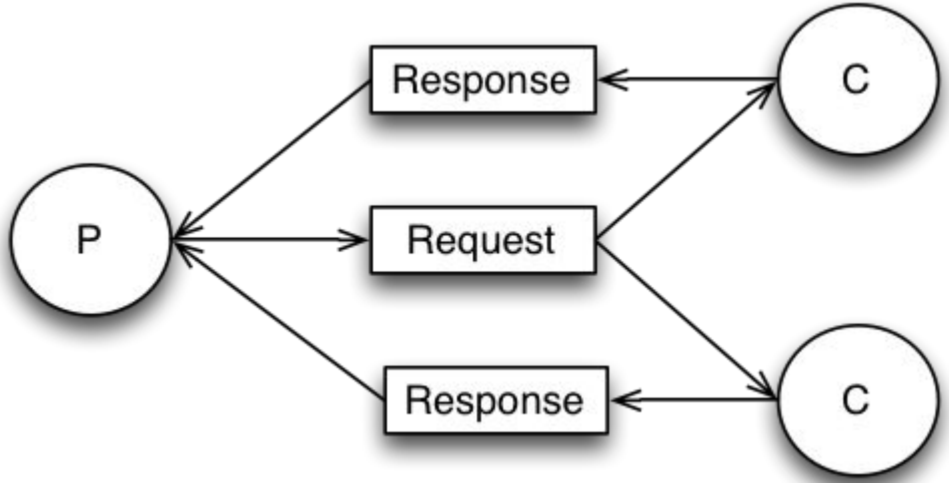
Durability - queues and messages

ack/nack/redeliver

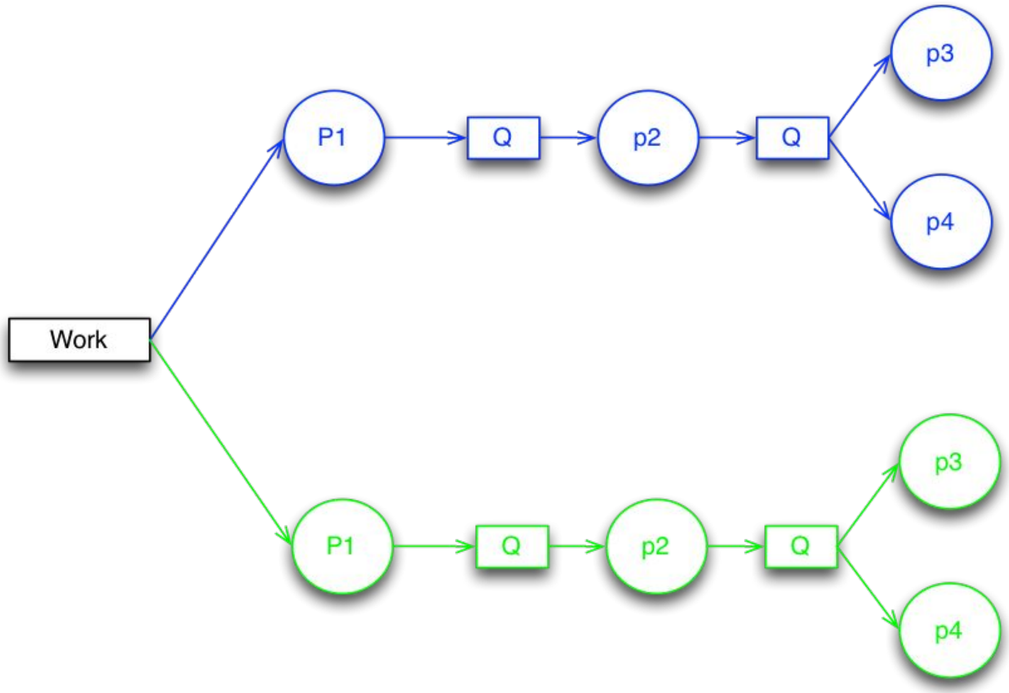
confirmation

Poison tasks

Using RabbitMQ as an RPC mechanism



Support for Blue/Green Deployment



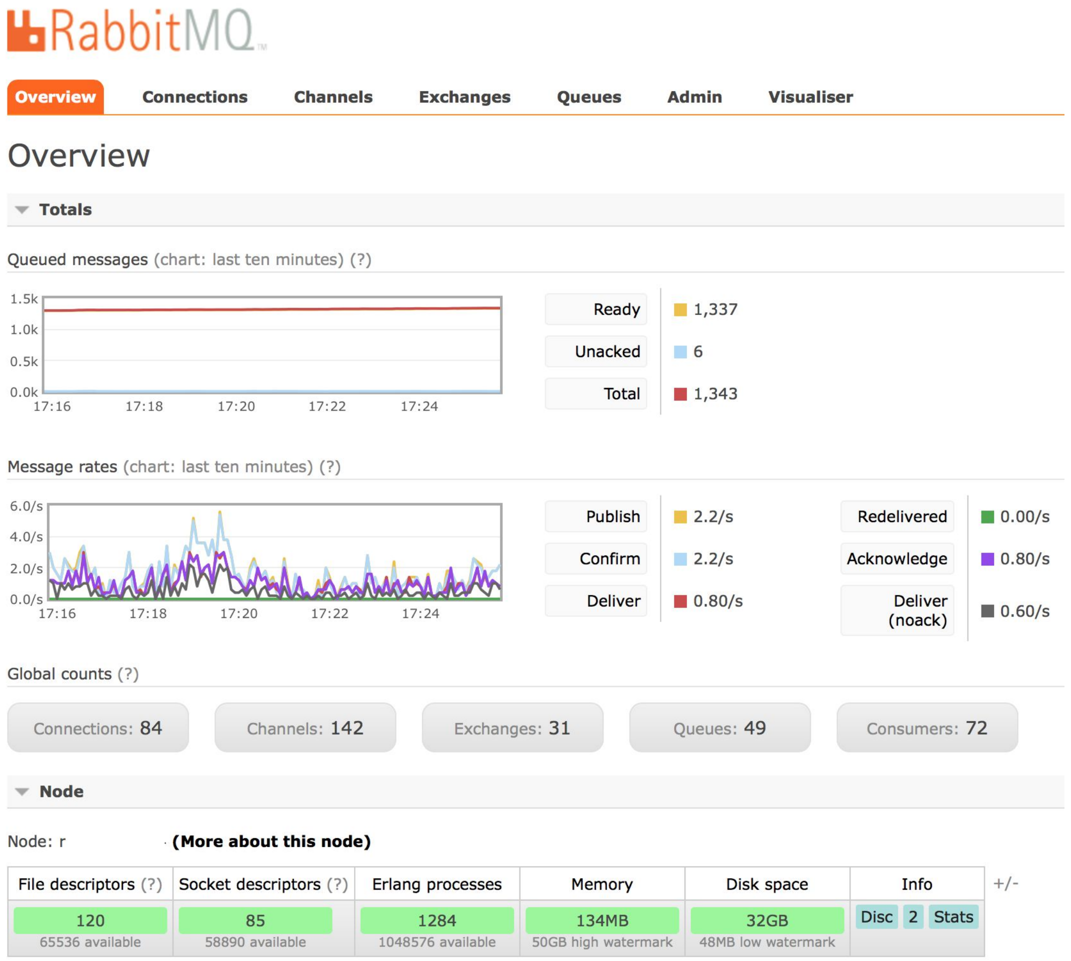
High Availability

Clustering

Mirroring

Federation

GUI Admin tool



Overview

Totals

Queued messages (chart: last ten minutes) (?)

Ready	🟨 1,337
Unacked	🟦 6
Total	🟥 1,343

Message rates (chart: last ten minutes) (?)

Publish	🟨 2.2/s	Redelivered	🟩 0.00/s
Confirm	🟦 2.2/s	Acknowledge	🟪 0.80/s
Deliver	🟥 0.80/s	Deliver (noack)	⬛ 0.60/s

Global counts (?)

Connections: 84 Channels: 142 Exchanges: 31 Queues: 49 Consumers: 72

Node

Node: r **(More about this node)**

File descriptors (?)	Socket descriptors (?)	Erlang processes	Memory	Disk space	Info
120 65536 available	85 58890 available	1284 1048576 available	134MB 50GB high watermark	32GB 48MB low watermark	Disc 2 Stats

+/-

Commandline admin tool

See queue lengths

add/remove queues/exchanges

add/remove messages