# Synchronizing Samvera Repositories

Samvera Connect 2018

James Griffin
Digital Infrastructure Developer
Princeton University Library

# Synchronizing Samvera Repositories

- Before discussing synchronization…
- What is a repository? Why does one build one?

# Digital Repositories: What Do They Do?

- Within an organization what do they provide?
  - Uploading and sharing things (Publishing)
  - Describing things (Curation)
  - Organizing things into collections (Asset management)
  - Finding things (Discovery)
  - Saving things (Preservation)
- Identifying these requirements, one develops a platform

# Building a Digital Repository

- How could one approach development?
  - ~~Fancy~~ Simple but expressive language?
  - Reliable and robust framework?
  - Community-driven?
- One can use Ruby on Rails and Samvera
- **Let’s build a repository!**

# Building a Digital Repository

*(7 months pass...)*

- **Samvera repository is released!**
- Curators can manage their collections
- Subject specialists can catalog items
- Reference librarians can use our repository as a new resource
- Generic users can browse each item and search for content
- **Successful deployment** 👏

# Building a Digital Repository

*(2 months follow...)*

- Subject specialists need new controlled vocabularies
- Curators need a custom user experience just for their collection
- Reference librarians need enhancements to the discovery interface
- Our response?
- **Extend the App!**

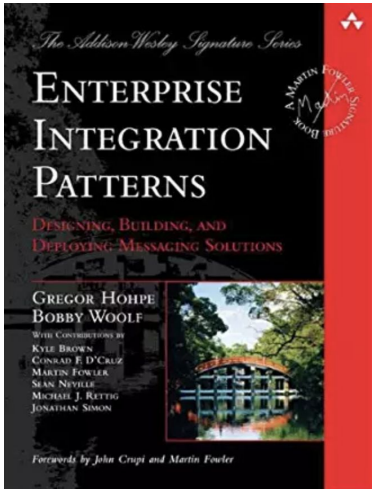
# Scaling a Digital Repository

*(1 year follows...)*

- Curators need another custom user experience for a new unique collection
- Reference librarians find the repository slow
- Subject specialists have new content to migrate into the repository
- We need to update the repository to the latest Samvera Gems
- ...but we can still only extend a single App!
- **Add a new server!**
- **Hire a new developer!**
- **We can use DevOps ⚡!**

# Motivation: Monolithic Repositories

- These problems are not new to web development
- Arise from an approach relying upon **monolithic system architecture**
- There are alternative architectural patterns
  - Service-Oriented Architecture
  - Distributed Object Architecture
  - Message-Oriented Middleware
  - Microservices
- These can be nebulous (and used as buzz-phrases)
- These often overlap



# Distributed Architecture

- How can system architecture be distributed?
- **This is a large topic**
- Computing can be distributed on many layers
  - Distributed object passing (CORBA, DCOM, dRuby*)
  - Web service standards (WSDL, SOAP, WADL)
  - Messaging protocols (XMPP, STOMP, AMPQ)
  - RESTful services
- We're going to focus upon distribution using messaging protocols

*Example of dRuby: https://github.com/youchan/drb-websocket

# Messages

- What is a message?
  - Data describing an event in our ecosystem
  - Payload also contains metadata about the event
  - e.g. `alice updated FileSet cca3c02 at 10/11/18 09:31:00UTC on repo1.institution.edu`
- Services publish messages
- Services listen for messages
- Messages are stored in queues
- Services only access messages using these shared queues

# Message Protocols

- Standard protocols determine how messages are sent
  - Over the TCP? Over the HTTP?
- Protocols also determine the message structure
- Streaming Text Oriented Messaging Protocol (STOMP)
  - Text-based messages
  - Key/value pairs contain the data
- Advanced Message Queuing Protocol (AMQP)
  - Messages are bitstreams (binary)
  - Defines how queues can be accessed by multiple clients

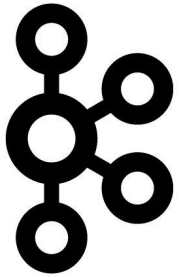

# Message Brokers

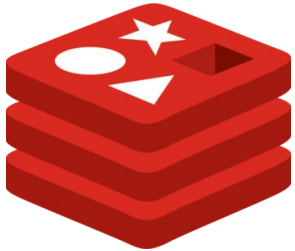
- Sharing queues between services is difficult
- What manages the queue?
  - **Message Broker**
- **Publish and Subscribe (Pub/Sub) pattern**
- Services publish messages to queue(s) through the broker
  - Services can publish to multiple queues
  - This is a *fanout*
- Services then access the queues by subscribing through the broker

# Message Brokers

## Open Source Message Brokers

- Apache Kafka
- Apache Camel
- ActiveMQ
- RabbitMQ
- HornetQ
- Redis
- Celery
- (There are more)




# Test Case: RabbitMQ

- We can only discuss one solution today :(
- Princeton University Library uses **RabbitMQ**
  - Implements the AMQP
- Other repositories are using alternatives
  - Apache Camel, Fedora 4, and Islandora (https://github.com/Islandora-CLAW/Alpaca)
  - Apache Kafka and Trellis (https://github.com/trellis-ldp-archive/trellis-kafka)

- Rails provides ActiveJob as an abstraction layer
  - ***Somewhat* comparable**
  - Support for Redis, MongoDB, PostgreSQL, ...
- Disclaimer:
  - **This talk is not an endorsement for RabbitMQ**

# Test Case: RabbitMQ and Ruby

- ruby-amqp/bunny
  - Gem for RabbitMQ in Ruby
- Configuration is simple
  - Client builds a connection
  - Client connects to a channel
  - Client interfaces with the queue

```ruby
require "bunny"

conn = Bunny.new
conn.start

ch = conn.create_channel

q = ch.queue("test1")

q.publish("Hello, everybody!")

delivery_info, metadata, payload = q.pop

puts "This is the message: #{payload}"
conn.stop
```

# Test Case: RabbitMQ and Rails


## Publishing to RabbitMQ

- ruby-amqp/bunny
  - Construct a Bunny client
  - Define an adapter for the client
  - Make a service object!

```
# frozen_string_literal: true

class MessagingClient

...

  def publish(status, model)

    message = generate_message(status, model)

    exchange.publish(message, persistent: true)

  rescue

    Rails.logger.warn "Unable to publish message to
#{amqp_url}"

  end

...
```

# Test Case: RabbitMQ and Rails


## Publishing to RabbitMQ

- ruby-amqp/bunny
  - Construct a Bunny client
  - Define an adapter for the client
  - Make a service object!
- Use transaction callbacks
  - `after_commit`

```ruby
class User < ApplicationRecord

  include Hyrax::User


  after_create_commit :publish_create_message

  after_update_commit :publish_update_message

  after_delete_commit :publish_delete_message

...
```

# Test Case: RabbitMQ and Rails


## Publishing to RabbitMQ

- ruby-amqp/bunny
  - Construct a Bunny client
  - Define an adapter for the client
  - Make a service object!
- Use transaction callbacks
  - `after_commit`
- **Publish messages to the queue**

```
class User < ApplicationRecord

...

  def publish_create_message

    messaging_client.publish_message(

            :create,

            self

    )

  end

...

  def messaging_client

    MessagingClient.new

...
```

# Test Case: RabbitMQ and Rails


## Subscribing to RabbitMQ

- Problem:
  - Rails can't serve requests and listen to the Rabbit queue
- **Use asynchronous workers**
- jondot/sneakers
  - Gem with support for RabbitMQ
  - Uses Redis for background processing

```
Sneakers.configure(

  amqp: "amqp://localhost:5672",

  exchange: "my_repository",

  exchange_type: :fanout,

  handler: Sneakers::Handlers::Maxretry

)

Sneakers.logger.level = Logger::INFO
```

# Test Case: RabbitMQ and Rails


## Subscribing using Sneakers

- Implement a Worker
- `Sneaker::Worker` Module
- Override the `#work` method

```ruby
class AmqpMessageWorker

  include Sneakers::Worker


  def work(payload)

    result = process(JSON.parse(payload))

    if result

      ack!

    else

      reject!

    end

  end
```

# Test Case: RabbitMQ and Rails


## Subscribing using Sneakers

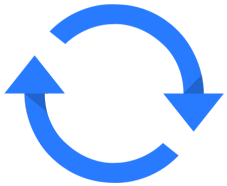
- Run the sneakers workers
- `bundle exec rake sneakers:run`

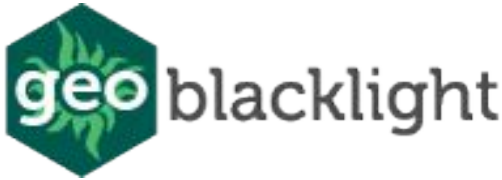
```
% bundle exec rake sneakers:run

WARN: Loading runner configuration...

...

INFO: New configuration:
#<Sneakers::Configuration:0x00007fb163c980d0 @hash=

INFO: Heartbeat interval used (in seconds): 30

INFO: Heartbeat interval used (in seconds): 30

...
```

# Test Case: RabbitMQ and Valkyrie

- How could one integrate RabbitMQ with Valkyrie?
  - Valkyrie uses `ChangeSets` to persist new or updated properties to repository objects
  - `ChangeSets` are persisted using a `ChangeSetPersister`
  - See the Valkyrie Wiki Documentation
- Extend the `ChangeSetPersister`
- Example: figgy

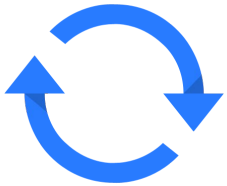
# Test Case: Valkyrie and GeoBlacklight

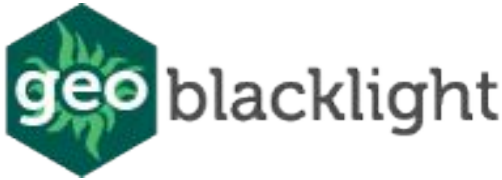





- How could one update a Blacklight catalog using repository messages?
- Implement a `Sneakers::Worker`
- Example: pulmap

# Test Case: Valkyrie and GeoBlacklight






Demonstration

# Test Case: RabbitMQ and Hyrax

- How could one integrate RabbitMQ with Hyrax?
  - Implement a new `Actor`
- Publishing
  - Implement a `Hyrax::Actors::MessagingActor`
  - Insert the `Hyrax::Actors::MessagingActor` into the stack
- Example Implementation
  - Example Hyrax on GitHub

# Test Case: RabbitMQ and a Newer Hyrax

- *But Actors will be deprecated*
  - https://github.com/samvera/hyrax/tree/destroy-all-actors
- Add a transaction step from `Dry::Transactions`
- Example Implementation
  - Example Hyrax on GitHub

# Questions? Comments?

# Synchronizing Samvera Repositories

Thank you to all involved in implementing this distributed architecture:

- Esmé Cowles
- Trey Pendragon
- Eliot Jordan
- Anna Headley
- Nikitas Tampakis
- Christina Chortaria
- Francis Kayiwa
- Kevin Reiss
- Shaun Ellis
- Jon Stroop
- Axa Liauw

**Thank you kindly for your attention**