Centuries Mutual Home App - Enterprise Integration Architecture

Dropbox Advanced + AMQP Messaging System Design

1. System Architecture Overview

High-Level Architecture Diagram

text Collapse Wrap Copy

```
Mobile/Web     API Gateway                Dropbox Advanced
 Frontend  <-> (Rate Limiting,   <->      (Storage Backend)   - File Storage
(Auth, Routing)

                              - Webhooks
                              - Audit Logs
                              - Shared Links

Client Devices  RabbitMQ
(Push/Email/SMS)<- (AMQP 1.0)

- Notifications   - Message Queue
- Document View   - Delivery Guar.
- Confirmations   - Daily Limits  -> Message Archive (Dropbox Folders)
```

Component Roles and Data Flow

Dropbox Advanced as Storage & Verification Backend:

- Unlimited API Calls: Leverages Dropbox Advanced's high-volume rate limits (no strict

caps for non-abusive use, supporting ~4TB/day uploads)

- Key Operations:
  - `/files/upload_session` for batch message archiving
  - `/sharing/create_shared_link_with_settings` for secure document sharing
  - `/files/list_folder` for client folder management
  - `/webhooks/process` for document view verification
- Storage Structure: Client-specific folders (`/clients/<client_id>/messages/` , `/clients/<client_id>/documents/` , `/clients/<client_id>/audit/`)

AMQP (RabbitMQ) as Messaging Infrastructure:

- Exchanges:
  - `insurance.direct` (direct exchange for client-specific messages)
  - `insurance.workflow` (topic exchange for enrollment -> claims -> payment workflows)
- Delivery Guarantees: Persistent queues with publisher confirms ensure at-least-once delivery
- Rate Limiting: Queue policies and app-level counters prevent message spam

Authentication & Security:

- Dropbox: OAuth 2.0 with short-lived access tokens (2-hour expiry)
- RabbitMQ: SASL/PLAIN over TLS 1.3
- End-to-End Encryption: Application-level encryption for message payloads using libsodium

Data Flow Example

1. Client Document Request: App sends AMQP message to `client123.documents` queue

2. Archive to Dropbox: Message payload archived to `/clients/123/messages/timestamp.json`
3. Document Generation: System creates shared link via Dropbox API
4. Client Notification: AMQP routes notification to client's device queue
5. Verification: Dropbox webhook confirms document view, stores audit log

2. Step-by-Step Implementation Guide

Phase 1: Environment Setup

Prerequisites:

- Dropbox Advanced/Enterprise account with API access
- RabbitMQ 3.13+ with AMQP 1.0 plugin
- Python 3.11+ with required SDKs:

bash Collapse Wrap Run Copy

```
pip install dropbox pika cryptography python-jose redis celery
```

Implementation Steps:

Step 1: Dropbox API Configuration

python Collapse Wrap Run Copy

```python
import dropbox
from dropbox.files import WriteMode

class DropboxManager:
    def __init__(self, access_token):
        self.dbx = dropbox.Dropbox(access_token)

    def setup_client_folder(self, client_id):
        """Create folder structure for new client"""
        folders = [
            f"/clients/{client_id}/messages",
            f"/clients/{client_id}/documents",
            f"/clients/{client_id}/audit"
        ]
        for folder in folders:
            try:
                self.dbx.files_create_folder_v2(folder)
            except dropbox.exceptions.ApiError as e:
                if "path/already_exists" not in str(e):
                    raise
```

Step 2: RabbitMQ Exchange and Queue Setup

python Collapse Wrap Run Copy

```
import pika
import json
```

```
import json

class AMQPManager:
    def __init__(self, connection_params):
        self.connection = pika.BlockingConnection(connection_params)
        self.channel = self.connection.channel()
        self.setup_infrastructure()

    def setup_infrastructure(self):
        # Declare exchanges
        self.channel.exchange_declare(
            exchange='insurance.direct',
            exchange_type='direct',
            durable=True
        )
        self.channel.exchange_declare(
            exchange='insurance.workflow',
            exchange_type='topic',
            durable=True
        )

    def create_client_queue(self, client_id, message_limit=10):
        """Create client-specific queue with daily message limit"""
        queue_name = f"client.{client_id}"
        # Apply daily message limit policy
        arguments = {
            'x-max-length': message_limit,
            'x-overflow': 'reject-publish'
        }
        self.channel.queue_declare(
            queue=queue_name,
            durable=True,
            arguments=arguments
        )
        # Bind to direct exchange
        self.channel.queue_bind(
            exchange='insurance.direct',
            queue=queue_name,
            routing_key=client_id
        )
```

Phase 2: Client Onboarding Workflow

Step 3: Client Registration Process

python Collapse Wrap Run Copy

```
class ClientOnboarding:
    def __init__(self, dropbox_mgr, amqp_mgr):
        self.dropbox = dropbox_mgr
        self.amqp = amqp_mgr

    def register_client(self, client_data):
        client_id = client_data['id']
        # Create Dropbox folder structure
        self.dropbox.setup_client_folder(client_id)
        # Create AMQP queues
        self.amqp.create_client_queue(client_id)
        # Store client metadata
        metadata = {
            'client_id': client_id,
            'registered_date': datetime.utcnow().isoformat(),
            'message_count_today': 0,
            'last_reset': datetime.utcnow().date().isoformat()
        }
        self.dropbox.upload_json(
            f"/clients/{client_id}/metadata.json",
            metadata
        )
```

Step 4: Document Template Management

python Collapse Wrap Run Copy

```
def setup_document_templates(self):
    """Create shared templates for common insurance documents"""
    templates = {
        'enrollment_form': '/templates/enrollment_form.pdf',
        'claims_form': '/templates/claims_form.pdf',
        'beneficiary_form': '/templates/beneficiary_form.pdf'
    }
    for template_name, path in templates.items():
        shared_link = self.dropbox.dbx.sharing_create_shared_link_with_settin
            path=path,
            settings=dropbox.sharing.SharedLinkSettings(
                requested_visibility=dropbox.sharing.RequestedVisibility.team
                link_password="secure_template_access"
            )
        )
        # Store template metadata
        template_data = {
            'name': template_name,
            'shared_link': shared_link.url,
            'created_date': datetime.utcnow().isoformat()
        }
        self.dropbox.upload_json(
            f"/templates/{template_name}_metadata.json",
            template_data
        )
```

Phase 3: Messaging Workflow Implementation

Component	Dropbox Role	RabbitMQ Role	Benefit
Messaging	Archive Messages	Route & deliver	Reliable, auditable
Documents	Store & share files	Notify on updates	Secure access control
Verification	Webhook confirmations	Delivery receipts	Compliance tracking
Rate Limiting	Store counters	Queue policies	Abuse prevention

Step 5: Message Publishing System

python ^ Collapse Wrap Run Copy

```python
class MessagePublisher:
    def __init__(self, dropbox_mgr, amqp_mgr):
        self.dropbox = dropbox_mgr
        self.amqp = amqp_mgr

    def send_client_message(self, client_id, message_data):
        # Check daily limit first
        if not self.check_message_limit(client_id):
            raise MessageLimitExceededException(f"Daily limit exceeded for cl

        # Prepare message payload
        message = {
            'id': str(uuid.uuid4()),
            'client_id': client_id,
            'type': message_data['type'],  # Document_Required, claim_update,
            'content': message_data['content'],
            'timestamp': datetime.utcnow().isoformat(),
            'attachments': message_data.get('attachments', [])
        }

        # Archive to Dropbox first
        archive_path = f"/clients/{client_id}/messages/{message['id']}.json"
        self.dropbox.upload_json(archive_path, message)

        # Publish to AMQP
        self.amqp.channel.basic_publish(
            exchange='insurance.direct',
            routing_key=client_id,
            body=json.dumps(message),
            properties=pika.BasicProperties(
                delivery_mode=2,  # Make message persistent
                message_id=message['id'],
                timestamp=int(datetime.utcnow().timestamp())
            )
        )

        # Update message counter
        self.increment_message_counter(client_id)

    def check_message_limit(self, client_id):
        """Check if client has exceeded daily message limit"""
```

```
            Check if client has exceeded daily message limit
        metadata = self.dropbox.download_json(
            f"/clients/{client_id}/metadata.json"
        )
        today = datetime.utcnow().date().isoformat()
        if metadata['last_reset'] != today:
            # Reset counter for new day
            metadata['message_count_today'] = 0
            metadata['last_reset'] = today
            self.dropbox.upload_json(
                f"/clients/{client_id}/metadata.json",
                metadata
            )
        return metadata['message_count_today'] < 10  # Daily limit

        except Exception as e:
            # Default to allowing if can't check
            logging.error(f"Error checking message limit: {e}")
            return True
```

Step 6: Message Consumer Implementation

python Collapse Wrap Run Copy

```
class MessageConsumer:
    def __init__(self, dropbox_mgr):
```

```
    def __init__(self, dropbox_mgr):
        self.dropbox = dropbox_mgr

    def process_message(self, channel, method, properties, body):
        """Process incoming AMQP messages"""
        try:
            message = json.loads(body)
            client_id = message['client_id']

            # Process based on message type
            if message['type'] == 'Document_Required':
                self.handle_document_request(message)
            elif message['type'] == 'claim_update':
                self.handle_claim_update(message)
            elif message['type'] == 'payment_reminder':
                self.handle_payment_reminder(message)

            # Store delivery confirmation
            confirmation = {
                'message_id': message['id'],
                'processed_at': datetime.utcnow().isoformat(),
                'status': 'delivered'
            }
            self.dropbox.upload_json(
                f"/clients/{client_id}/audit/delivery_{message['id']}.json",
                confirmation
            )

            # Acknowledge message
            channel.basic_ack(delivery_tag=method.delivery_tag)

        except Exception as e:
            logging.error(f"Error processing message: {e}")
            # Reject and requeue for retry
            channel.basic_nack(
                delivery_tag=method.delivery_tag,
                requeue=True
            )
```

Phase 4: Document Handling System

Step 7: Secure Document Sharing

python Collapse Wrap Run Copy

```python
class DocumentManager:
    def __init__(self, dropbox_mgr):
        self.dropbox = dropbox_mgr

    def create_secure_document_link(self, client_id, document_path, expires_h
        """Create time-limited secure link for document access"""
        # Set expiry time
        expires_at = datetime.utcnow() + timedelta(hours=expires_hours)

        # Create shared link with security settings
        shared_link = self.dropbox.dbx.sharing_create_shared_link_with_settin
            path=document_path,
            settings=dropbox.sharing.SharedLinkSettings(
                requested_visibility=dropbox.sharing.RequestedVisibility.pass
                link_password=self.generate_secure_password(),
                expires=expires_at
            )
        )

        # Log document access creation
        access_log = {
            'client_id': client_id,
            'document_path': document_path,
            'shared_link': shared_link.url,
            'created_at': datetime.utcnow().isoformat(),
            'expires_at': expires_at.isoformat(),
            'access_attempts': 0
        }
        self.dropbox.upload_json(
            f"/clients/{client_id}/audit/document_access_{uuid.uuid4().hex}.j
            access_log
        )

        return shared_link.url, shared_link.password
```

Step 8: Document Upload Handling

python Collapse Wrap Run Copy

```
def handle_document_upload(self, client_id, file_data, document_type):
    """Handle client document uploads with encryption"""
    # Encrypt file data
    encrypted_data = self.encrypt_document(file_data)

    # Generate unique filename
    filename = f"{document_type}_{datetime.utcnow().strftime('%Y%m%d_%H%M%S')
    upload_path = f"/clients/{client_id}/documents/{filename}"

    # Upload to Dropbox using upload session for large files
    file_size = len(encrypted_data)
    if file_size > 4 * 1024 * 1024:  # >4MB, use upload session
        session_start_result = self.dropbox.dbx.files_upload_session_start(
            encrypted_data[:4*1024*1024]
        )
        cursor = dropbox.files.UploadSessionCursor(
            session_id=session_start_result.session_id,
            offset=4*1024*1024
        )
        # Upload remaining chunks
        remaining_data = encrypted_data[4*1024*1024:]
        while len(remaining_data) > 0:
            chunk = remaining_data[:4*1024*1024]
            remaining_data = remaining_data[4*1024*1024:]
            if len(remaining_data) == 0:
                # Final chunk
                self.dropbox.dbx.files_upload_session_finish(
                    chunk,
                    cursor,
                    dropbox.files.CommitInfo(path=upload_path)
                )
            else:
                self.dropbox.dbx.files_upload_session_append_v2(
                    chunk, cursor
                )
                cursor.offset += len(chunk)
    else:
        # Simple upload for small files
        self.dropbox.dbx.files_upload(
            encrypted_data,
            upload_path
```

```
            upload_path,
            mode=WriteMode('overwrite')
        )

    # Create metadata record
    metadata = {
        'client_id': client_id,
        'document_type': document_type,
        'original_filename': file_data.get('filename', 'unknown'),
        'file_size': file_size,
        'uploaded_path': upload_path,
        'uploaded_at': datetime.utcnow().isoformat(),
        'encryption_method': 'AES-256-GCM'
    }
    self.dropbox.upload_json(
        f"/clients/{client_id}/documents/metadata/{filename}.json",
        metadata
    )
```

Phase 5: Verification & Audit System

Step 9: Webhook Configuration for Document Verification

python Collapse Wrap Run Copy

```
class WebhookManager:
    def __init__(self, dropbox_mgr):
```

```
        self.dropbox = dropbox_mgr

    def setup_document_webhooks(self):
        """Configure Dropbox webhooks for document access verification"""
        webhook_url = "https://your-api-gateway.com/webhooks/dropbox"

        # Create webhook for file access events
        webhook = self.dropbox.dbx.webhooks_create(
            uri=webhook_url,
            events=[
                'file_requests.deadline.reminder',
                'sharing.shared_link_accessed'
            ]
        )
        return webhook.webhook_id

    def process_webhook(self, webhook_data):
        """Process incoming Dropbox webhook for verification"""
        for account in webhook_data.get('list_folder', {}).get('accounts', []
            account_id = account['account_id']
            # Get file changes
            changes = self.dropbox.dbx.files_list_folder_continue(
                account['cursor']
            )
            for entry in changes.entries:
                if isinstance(entry, dropbox.files.FileMetadata):
                    # Document was accessed/modified
                    self.log_document_verification(account_id, entry)

    def log_document_verification(self, account_id, file_entry):
        """Log document access for audit purposes"""
        verification_log = {
            'account_id': account_id,
            'file_path': file_entry.path_lower,
            'file_id': file_entry.id,
            'client_session_id': file_entry.client_modified,
        }
```