TODO list for this document

- highlight the parts of the workflow that we need to create activities for to replace HW (70%)
- write up a proposed doc showing what our new workflows might look like (60%)
- complete tl;dr overview of the doc (10%)

This documentation aims to refer to the exp branch of the eLife-bot code.

Existing POA workflow

- every hour EJP sends csv files with metadata to the `elife-ejp-ftp` S3 bucket. They have been provided access to this via the https://cloudgates.net service, and to change the location of this we need to modify the cloudgates settings and resullpy FTP credentials to the vendor.
- when an article has been accepted for publication in EJP the production team hit a button in EJP that will cause EJP to FTP a file to the `elife-ejp-poa-delivery` S3 bucket
- `cron.py` checks at 11am for new content in a bucket defined by the setting `poa_bucket` which needs to be set to be teh same bucket that EJP are sending their content to (done in settings.py for the elife-bot code)
- on discovering a new file in that bucket (via the S3Monitor activity) the PackagePOA activity is started
- this activity looks for content in directories on the local Ec2 machine that are set in the settings file of the elife-poa-xml-generation code. It then sends the output to an s3 bucket that is defined by the settings.poa_packaging_bucket which is set to `elife-poa-packaging`
- the PublishPOA is invoked if a new file is found in `elife-poa-packaging`.
- this will create a folder in an outbox of the folloiwng format
  - folder name YYYYMMDD
  - folder contains, for each article to be published, a file of the following kind
  - elife_poa_eNNNNN.xml
  - elife_poa_eNNNNN_ds.zip
  - decap_elife_poa_eNNNNN.pdf
- this sends files to HW and to crossref and prepares files for downstream delivery.
- Files appears in Highwire express (HWX)
- HWX shows all POA articles for the day on one "batch"
- HW creates a record in HWX
- HW creates nodes in Drupal
- supp files are loaded to the appropriate location for download
- the PAP batch shows all the POA papers in HWX
- HWX has a link to the paper in Drupal where the content is in a "not published state"
- production manually checks the PDF on HWX, usually to check against
  - special characters
  - that decapitation has happened on the PDF
  - that the abstract appears OK
  - some other checks (listed in the POA protocals document)
- production push an "Approve" button
- another page is displayed with another "Approve button"
- a "Success Page" is displayed
- Content is added to the search index
- usually within 1/2 an hour the paper on the Drupal Site is in a published state
- it's not clear to me which of the "publish" activities operate on the contents that we have here.

Existing VOR workflow

- content processor sends a zip file to an S3 bucket and to a HW FTP endpoint, I *think* these go into the s3 bucket `elife-articles-hw`.
- content processor sends Crossref an XML file to update the crossref record
- These files appear in a directory named as NNNNN - where this is the `f-id`. There is one folder for every article. The folder contains files of the format
- elife_YYYY_NNNNN.img.zip
- elife_YYYY_NNNNN.pdf.zip
- elife_YYYY_NNNNN.xml.zip
- Files appears in Highwire express (HWX)
- each file takes it's own batch through the system
- HW creates a record in HWX
- if production can get to HWX then they see the following stages (information can been seen at each of these stages)
  - pre-intake
  - intake
  - processing
  - assembly
  - production
  - downstream
- before assembly HWX does the following
- images are converted
- HW creates nodes in Drupal
- assetts are loaded to the appropriate location, e.g. for download or to the CDN
- XML is transformed to HTML and provided via a markup service
- Some magic related article fun happens
- when the workflow gets to assembly assembly will turn orange and production can make a decision
- at assembly there is a QA report that links to the Drupal site
- production checks everything in the article on the Drupal site
- videos
- images
- tables
- decision letters
- production push an "Approve" button
- another page is displayed with another "Approve button"
- HWX shows that state is changing in the production process
- if the system works correctly then in about 20 minutes
- Content is added to the search index
- HW starts to collect PDF download and pageview metrics
- RSS feed is updated
- if it goes "red" all bets are off
- elife-bot via a setting in the db.provier script, is monitoring for new files of spcific types ("xml", "pdf", "img", "suppl", "video", "svg", "jpg", "figures").
- the bucket that is polled is defined in settings.py.
- Where new files have been identified as appropriate, then the following workflows are triggered, which mostly send content to the CDN, and also prepare a Lens landing page.
- cron_NewS3XML more info
- cron_NewS3PDF more info
- cron_NewS3SVG more info
- cron_NewS3Suppl more info
- cron_NewS3JPG more info
- cron_NewS3FiguresPDF more info

Existing Deposition workflows

- Pub router deposits once per day 23:45 UTC
- Cengage deposits once per day 22:45 UTC
- pubmed depostits happen every hour
- pubmed deposits inlude some code to determine version number of the article and to set the published date appropriatly.

TODO: determine where we have code that pings the Drupal site for publication dates

---

Proposed new VOR workflow (all to be discussed)

- content processor sends a zip file to an S3 bucket that follows our new naming convention.
- The arrival of a file triggers an SQS event which triggers a workflow
- an activity in the workflow unpacks the zip file into an s3 bucket
- an activity in the workflow sends a signal to the Processing Event Store (and appropriate other activities in this workflow also send signals to this event store)
- an activity in the workflow hits an API to determine the current working version number
  - this api will deterine what the previous published date was if we are looking at resupplying an article through this workflow.
- on receipt of a usable version number for the file a series of activities are invoked
  - zip content is placed into some working state or working directory (To be confiremed?)
  - article XML is rewritten to point to the new updated figure and supp file versions
  - images are renamed to match the new version number
  - images are resized based on an input YAML file
  - XML and generates the EIF JSON, which includes the version number, but indicates that the article needs to be in an unpublished state
  - XML is placed in a location where the markup service can access it
  - the Drupal site hits the markup service and generates the full article page (to be confimed)
  - a reciept JSON is generated by the Drupal site
- the publishing team receive a link to enable them to preview the content
- the publishing team approve content, and publish it
- Drupal is instructed to update it's search index
- the Drupal layer confirms that a specific version has been published
- content in the CDN is unmasked (to be confirmed)
- the data store that keeps track of version numbers is updated
- the files in the working directory are archived with the appropriate version number
- the publication date of that version is recorded in a data store
- RSS feed is updated on the Drupal site
- crossref is updated with our article info
- content is placed in the appropriate locations for downstream article deposition to occur

TODO: evaluate whether we need to retain the existing publish to CDN workflows

---

Bot overview.

The eLife bot uses Amazon Simple Workflow to manage coordinated tasks in our publishing workflow.

We start a workflow from a `cron.py` script, which usually invokes a starter, invoking a workflow which in turn invokes a set of activities. The usual strucutre for a given workflow looks like this:

- cron_starter: starter invoked from cron.py (usually invokes another starter, can sometimes go straight to triggering a workflow)
- starter: starter_name
  - workflow: Workflow name
  - activity: activity1
  - activity: activity2
  - activity: activity3

The workflows that we have developed so far are the following ones:

- S3Monitor - looks for changes in an s3 bucket and updated SDB with info (more details)
- PackagePOA - generates POA XML, decapitates PDF,prepares emails for authors and packages content in readyness for deliery to HW (more details)
- PublishPOA - generate go.xml, ftp content to HW, and place content ready for further downstream delivery to pubmed and crossref, send content to crossref. (more details)
- PublishArticle - puts an XML file on the CDN, creates a lens template HTML for that article, send the XML to an outbox for downstream processing (more details)
- PublishPDF - puts a PDF in the CDN (more details)
- PublishSuppl - puts supplemental data in the CDN (more details)
- PublishJPG - put JPGs into the CDN (more details)
- PublishFiguresPDF - moves figuresPDF to the CDN (more details)
- PublicationEmail - emails authors on publication of their POA article (more details)
- PubRouterDeposit - picks a destination to FTP content to, calls FTPArticle (more details)
- FTPArticle - FTPs content to an endpoint (more details)
- PubmedArticleDeposit - Ftps content to pubmed, does not use FTPArticle workfkow (more details)
- AdminEmail - sends emails about workflow state to admins (more details)
- PublishSVG - no longer used (more details)

Deployment and configuration of the elife-bot

Deployment

The main bot code is deployed via salt. The `exp` branch is currently deployed locally.

During salt deployment elife-poa-xml-generation is cloned is brought into the same directory structure as the bot-code. The elife-poa-xml-generation repo has many functions that are used throughout, e.g. here.

We may decide to integrate with the elife api

Configuration

As a result of bringing in the elife-poa-xml-generation repo repo there are a number of locations where bot settings can be found. Settings can be used to configure ftp credentials, the names of s3 buckets, times for cron jobs, and a number of other configurables. A summary of these are:

- elife-bot settings
  the most natural home for settings for the project, for the master branch this is configured via salt.
- cronfile for the bot
  this starts the main python processes that are responsible for interacting with amazon SWF. For the master branch this is set via salt
- cron.py & cron starters
  some specific workflows are tied to starting at specific times of the hour, and those times and workflows are laid out in cron.py and in the associated starter files.
- elife-poa-xml-generation
  some functions are called from the elif -poa-xml-generation repo, and that repo contains it's own settings file. For example ftp credentials for pubmed are set here, rather than in the main bot settings file.
- inline
  some functions have settings hardwired in them, I've attempted to call out where that happens in the documentation that follows.

Detailed description of workflows

Cron.py and the control flow

Cron.py is set to run by a cron script every *TBD* minutes. It runs it's main function `run_cron` which will invoke a starter process, if certain conditions are met at the given moment in time that cron.py is run. These starter processes in turn invoke workflows that themselves invoke activities. When each workflow and activity is begun a set of equivlant named workflows and activities are created in Amazon Simple Work Flow (SWF), and SWF is responsible for tracking ongoing workflows and activities, and tracking completion. History of activities tends to be written to an Amazon Simple DB instance.

---

starter_S3Monitor look in an S3 bucket and store changes to that bucket as metadata in a simple DB database.

- cron_starter: starter_S3Monitor
- workflow: S3Monitor
  - activity: S3Monitor (S3Monitor activity: poll S3 bucket and save object metadata into SimpleDB)

Config for simpledb is done in settings.py

`simpledb_region = us-east-1`

- cron_starter: cron_NewS3POA
- starter: starter_PackagePOA
  - workflow: PackagePOA
  - activity: PackagePOA (build new POA xml based on content from EJP, and place in a location for another workflow to pick up for delivery to HW)

The PackagePOA workflow is quite intricate. Crucially it calls on a number of functions in the elife-poa-xml-generation repo for in order to complete the workflow.

- looks for content in `elife-ejp-poa-delivery` (delivered by EJP)
- A set of directories are created on the ec2 instance that is running the activity.
- Zipfile gets downloaded to a temporary directory on the Ec2 instance running the activity
- extract a DOI from the Zipfile
- abort if no DOI can be generated
- otherwise create a new directory for output to Highwire (https://github.com/elifesciences/elife-bot/blob/master/activity/activity_PackagePOA.py#L192)
- in the copy_pdf_to_hw_staging_dir function we attempt to decapitate the PDF
- The new processed files are placed in `self.elife_poa_lib.settings.STAGING_TO_HW_DIR`.
- the processing of the zip file should also have decapitated the PDF from EJP, so we check whether that PDF has been decapitated. We look in `elife_poa_lib.settings.STAGING_DECAPITATE_PDF_DIR` to see if that decapitated PDF is present.
- a new manifest XML file is generated for Highwire
- download a set of CSV files from EJP
- create a new XML files for submission to HW that has the article XML in it, insofar as we can generate it from the csv files downloaded from EJP (this will not cotain the body text of the XML file).
- copy all of these files to an S3 Outbox
- create an email of the format " email_type = "PackagePOA" and add this to a mail queue. (Mail templates are stored on s3). This is a system email and is sent to emails that are configured in `settings.ses_poa_recipient_email`.

- cron_starter: starter_PublishPOA
- workflow: PublishPOA
  - activity: PublishPOA (files are sent to HW, and to local folders for further processing, see below)
  - activity: DepositCrossref (generate crossref XML and put a copy is a specific named S3 bucket)

The `PublishPOA` activity is quite intricate, and does the following:

- `PublishPOA` creates the following directories
  - self.publish_bucket = settings.poa_packaging_bucket
  - self.outbox_folder = "outbox/"
  - self.published_folder = "published/"
- files are downloaded from the s3 bucket that `NewS3POA` populated
- some checks are made to confirm that supp files and zip files have complimentary data and files in them
- supplement files and zip files are sent to the HW ftp site
- a go.xml file is created and sent the HW FTP endpoint
- If these files make it to HW then xml is sent to
  - xml_to_crossref_outbox_s3
  - xml_to_pubmed_outbox_s3
  - xml_to_publication_email_outbox_s3
- an email is sent to `settings.ses_poa_recipient_email` (the code here is duplicated from the packagePOA script)

The base S3 bucket for packaging POA content is set in the settings file, and sub-folders are set in the PublishPOA activity, see for example the setting of crossref/outbox.

TODO: de-duplicate the email code

TODO: determine how this workflow knows which files to start working on

---

cron_NewS3XML - check in SimpleDB for info on any new or modified XML files that are in an S3 bucket since a given date, then publish that specific numbered XML file via a series of activities.

- cron_starter: cron_NewS3XML
- starter: starter_PublishArticle
  - workflow: PublishArticle
  - activity: UnzipArticleXML (download XML from S3 and save to the CDN)
  - activity: LensArticle (Create a lens article index.html)
  - activity: ArticleToOutbox (copy article from s3 to a set of folders on the local ec2 instance).
  - activity: LensXMLFilesList (Create the eLife Lens xml list file for cache warming, and then save those to the S3 CDN bucket)

filesystem.py provides access to the file system.

Cdn location setting (in settings.py).

The HTML template for the lens landing page is stored in S3 and is accessed via templates.py.

Templates dir in s3 is set in settings.py.

Lens bucket location is set in a call to settings.py.

ArticleToOutbox creates the following local directories in the Ec2 instance

- `self.pubmed_outbox_folder = "pubmed/outbox/`
- `self.publication_email_outbox_folder = "publication_email/outbox/"`
- `self.pub_router_outbox_folder = "pub_router/outbox/"`
- `self.cengage_outbox_folder = "cengage/outbox/"`

Much of the logic of `ArticleToOutbox` seems quite similar to `UnzipArticleXML`, however this activity includes a function is_resupply that seems to hard-code eLife article numbers into volumes, and it's not clear why.

TODO:How do we know where to download a file from?

TODO: find out why we have a hard coded file list in ArticleToOutbox.

---

cron_NewS3PDF, cron_NewS3Suppl, cron_NewS3JPG, cron_NewS3FiguresPDF, these workflows are almost identical and they take a file from S3 and place it into the CDN. They have a significant amount of code duplication.

- cron_starter: cron_NewS3PDF
- starter: starter_PublishPDF
  - workflow: PublishPDF
  - activity: UnzipArticlePDF (takes a PDF file from S3 and puts it in the cdn)

TODO:look at how to tidy up the starter code to remove dependency on SDB

TODO: refactor UnzipArticlePDF along with other activities, that push content to the CDN, and provide one generalised activity for this.

- cron_starter: cron_NewS3Suppl
- starter: starter_PublishSuppl
  - workflow: PublishSuppl
  - activity: UnzipArticleSuppl (Downloads a S3 object from the elife-articles bucket, unzip if necessary, and save to the elife-cdn bucket.)
- cron_starter: cron_NewS3JPG
- starter: starter_PublishJPG
  - workflow: PublishJPG
  - activity: UnzipArticleJPG puts a file into the CDN.
- cron_starter: cron_NewS3FiguresPDF
- starter: starter_PublishFiguresPDF
  - workflow: PublishFiguresPDF
  - activity: [UnzipArticleFiguresPDF](https://github.com/elifesciences/elife-bot/blob/exp/activity/activity_UnzipArticleFiguresPDF.py)

---

starter_PublicationEmail - prepares an email to send to authors on publication of their paper via POA.

- cron_starter: starter_PublicationEmail
- workflow: PublicationEmail
  - activity: PublicationEmail (prepares an email to send to authors)

The PublicationEmail activity is moderatly involved, it does the following:

- sets a POA publication bucket from settings.py
- internally defines some settings that specify which article types not to email about, and which kinds of emails to send.
- downloads templates from s3
- gets xml files from s3 outbox
- do some author extraction and article checking
- send an email
- emails are sent by adding the email to a queue
- clean up the outbox

TODO: find out how this queue is then processed for actually sending mail

---

starter_PubRouterDeposit - deposits articles either with HEFCE or with CENGAGE depending on how the workflow is created.

- cron_starter: starter_PubRouterDeposit (pick between HEFCE or CENGAGE workflow)
- workflow: PubRouterDeposit
  - activity: PubRouterDeposit (deposit an article via FTP)
  - workflow: FTPArticle
    - activity: FTPArticle

The PubRouterDeposit activity is moderatly invovled, and it will inovke a new workflow - the FTP article workflow, if it needs to FTP an artticle.

- pick between HEFCE or CENGAGE
- looks again at poa packaging outboux
- if we have an approved article ftp the article
- starts the FTPArticle workflow. This seems to potentially be the first location where an activity starts a new workflow.
- the FTPArticle workflow simply starts the FTPArticle activity
- creates internal activity directories
- download files to be sent
- choose between HEFCE and CENGAGE for ftping the article
- extract ftp credentials from the settings.py file
- the main differences between HEFCE and CENGAGE can be seen here and seem to be based on differences in files to be sent.

TODO: get clarity on how we get aproval to FTP an article

TODO: refactor to rationalise download from s3

TODO: refactor activity_FTPArticle.py to be a generic FTP script with no knowledge of the endpoint

---

- cron_starter: starter_PubmedArticleDeposit
- workflow: PubmedArticleDeposit
  - activity: PubmedArticleDeposit
- `PubmedArticleDeposit` starts workflow `PubmedArticleDeposit` which starts activity `PubmedArticleDeposit`
- PubmedArticleDeposit activity Download article XML from pubmed outbox, generate pubmed article XML, and deposit with pubmed.
- poa packaging bucket set by settings
- download files
- ftp file files
- this calls on a function in the elife-poa-xml-generation repo to ftp to highwire
- the actual ftp to highwire code seems to get it's FTP settings from a settings file local to elife-poa-xml-generation
- **TODO: bring the ftp to pubmed function into a common ftp function**

TODO: bring more clarity to which kinds of new files (VOR vs POA, asset vs XML) are the files that get actioned by the many cdn deposit workflows.

---

starter_AdminEmail - Email administrators a workflow history status message.

- cron_starter: starter_AdminEmail
- workflow: AdminEmail
  - activity: AdminEmailHistory

Admin email list is set by `settings.ses_admin_email`.

---

cron_NewS3SVG: converts some SVG files to JPGs, this workflow is no longer needed and should be terminated.

- cron_starter: cron_NewS3SVG (this job is no longer needed)
- starter: starter_PublishSVG
  - workflow: PublishSVG
  - activity: UnzipArticleSVG (Download a S3 object from the elife-articles bucket, unzip if necessary, and save to the elife-cdn bucket.)
  - activity: ConverterSVGtoJPG (Extract base64 image data from SVG and save as JPG: Download a S3 object from the elife-articles bucket, unzip if necessary, convert each, and save to the elife-cdn bucket.)

TODO: remove this workflow from cron.py

Reccomendations for refactoring

- remove SVG workflow
- unify FTP workflows
- unify code used for sending files to CDN
- understand, and potentially remove, the hard coded article number list in ArticleToOutbox
- deduplicate code that is used to send emails
- remove starter dependcy on SDB
- move hard coded config variable out of the code into a settings or YAML file