

Growing a Sustainable Publishing Technology Service for Libraries

Library Publishing Forum 2021

Bart Kawula & Kaitlin Newson ([@kaitlinnewson](#))
Scholars Portal

[Kaitlin start]

Hello everyone. My name is Kaitlin Newson, and I'm joined by my coworker Bart Kawula from Scholars Portal.

As many of you will know, there's more to hosting technology than just putting it on a server. Today we're going to talk about our experiences with growing a sustainable publishing technology service for libraries.

Our hope with today's talk is to reflect on what goes into technology hosting for publishing services, the workflows we're using, how we make certain decisions, and ultimately the best practices we can work towards with the resources that we have.

Publishing services at Scholars Portal

- Scholars Portal & OCUL
- Public Knowledge Project (**PKP**)
 - Open Journal Systems (**OJS**)
 - Open Monograph Press (**OMP**)



First, a bit of background on our publishing services.

Scholars Portal is the technology service arm for the Ontario Council of University Libraries, an academic library consortium representing 21 universities across Ontario.

The Public Knowledge Project, or PKP, develops open-source publishing software.

2 of the applications that PKP develops are Open Journal Systems, or OJS, which is a journal management and publishing software, and Open Monograph Press, or OMP, which is a monograph publishing software.

At Scholars Portal, we provide a subscription-based hosting service for OJS and OMP to Ontario universities.

Publishing services at Scholars Portal

- 12 universities in Ontario, 150+ journals
- 2 librarians & 1 systems admin
- Includes technical support, sponsored CrossRef memberships, analytics, preservation, theming & customizations

Our hosting service began in 2012 and has grown to 12 universities across Ontario which collectively host over 150 journals. We host and manage the software on behalf of university libraries, who then run their own individual library publishing programs. Each institution has its own instance of the software.

The service is supported by myself, Bart, and a systems support specialist.

Along with hosting software, the service includes troubleshooting and technical support, sponsored CrossRef memberships for DOI minting, hosted Matomo analytics (an alternative to Google Analytics), preservation of journals in the Scholars Portal Trusted Digital Repository, as well as options for theming and customizations.

Service scaling

- 1 -> 12 libraries
- Balancing sustainability, service & workload
- Supporting scale with technology & automation

Over the last 9 years of the service, we've gradually grown from 1 to 12 institutions that we host publishing software for.

As the service has grown, we've had to find ways to ensure that we find a balance between doing things sustainably, providing good service to libraries and end-users with varying needs and resources of their own, and managing the workload of our team alongside other projects. One way we do this is by leveraging different technologies to support scaling our service, which is critical when our service needs grow but the number of people working on the service doesn't.

We've made use of a number of tools to help us as we scale our service, including tools that help us in the most time-consuming part of the service: client support.

Client support

[Bart start]

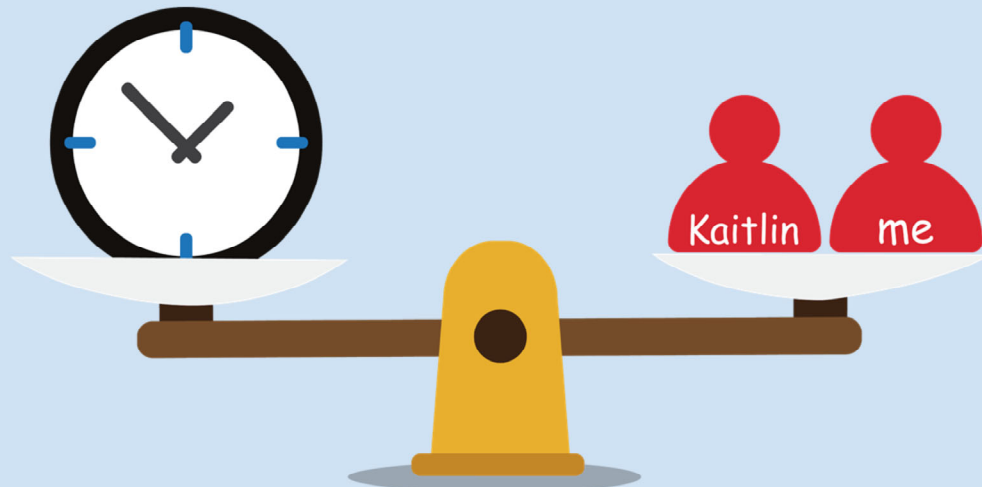
Client support is what takes up most of our time when it comes to our publishing services and its become the most critical component of what we do.

Client support

- Workload management
- Time tracking
- Defining roles & responsibilities
- External communities

When we started hosting we were expecting to only take care of software and maintenance, but as we acquired new members we found that there's a whole layer of technical support around making the software do what the user needs it to do, rather than just using it the way it's intended to be used. Over the years it's become apparent that for technology to be fully utilized there needs to be a strong support component because the more journals that are added to each instance the more the need for client support grows.

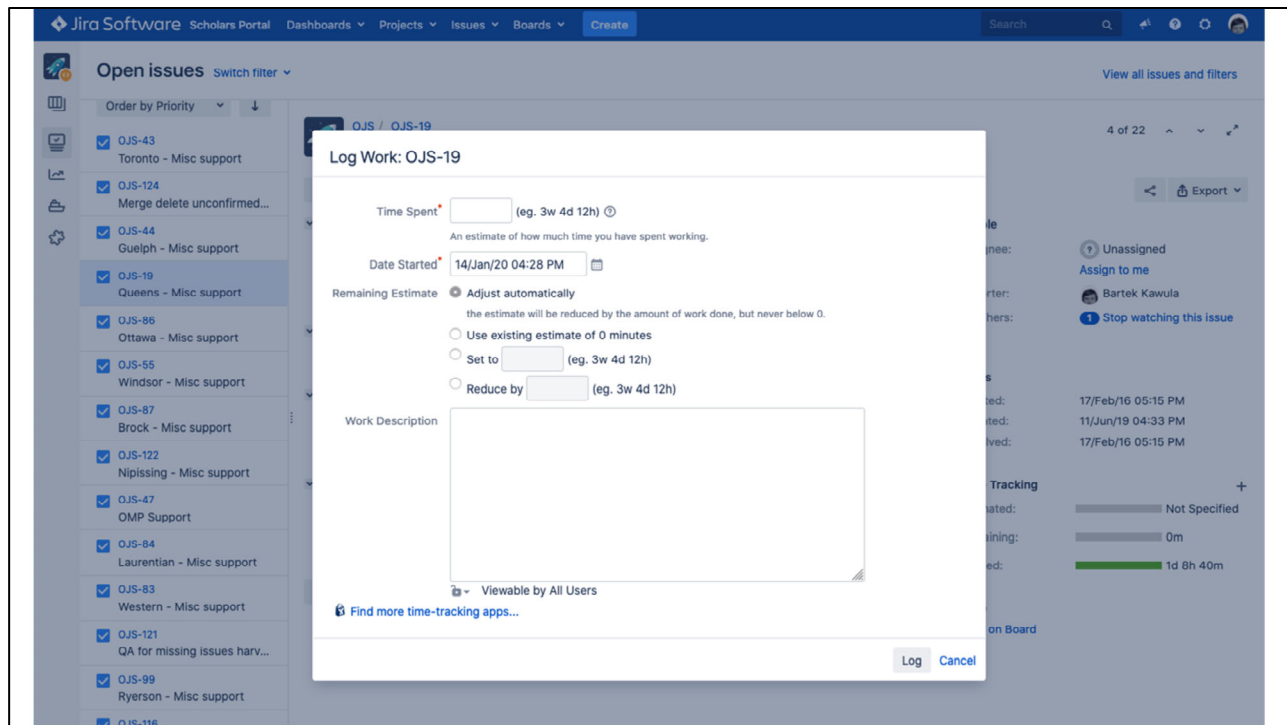
Workload management



Like in most library IT departments no one person does one thing. People get spread across projects in order to create redundancy, which is critical for providing scalable client support.

Since Kaitlin joined our team we've been better able to balance our publishing efforts with our other work responsibilities. From a practical point of view this means that we can answer most support queries within the same business day.

Typically when an email comes in we use our internal chat system to decide who can do what based on current workloads. Support requests can range widely from simple interface or workflow question that require a few minutes to answer to more involved systems level issues that can take days to resolve.



As you can see, time and its constraints are the main thing for us when dealing with client support, which is why we try to track it. We use the built in time logging features for JIRA. For those of you who aren't familiar with JIRA, it's a ticket management system that also works as a project management tool.

So everytime we answer an email we log the work we do associated with that request, with both a description and the time spent. It's very tedious, but it's critical for determining the amount of resources you devote to something and when you have solid numbers you can critically evaluate the sustainability of what it is that you do.

Time tracking

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
		ALGOMA	BROCK	GUELPH	LAURENTIAN	OTTAWA	QUEENS	RYERSON	TORONTO	UOIT	WESTERN	WINDSOR	MAINTENANCE	BILLABLE	TOTAL
1															
2	# of journals	0	7	20	2	11	16	4	41	1	9	8			
3															
4	support hours*	0	2	15.75	0.75	4.5	20	0	20.5	0.25	28.5	13.25	64.25	5	
5	OJS 3 upgrade*	0	72	5.75	0	13.5	0	0	15	0	0	39.5			
6	total hours	0	74	21.5	0.75	18	20	0	35.5	0.25	28.5	52.75	64.25	5	320.5
7															
8	* from August 2017 to August 2018														
9															
10	Current Fee	0	2500	2500	2500	2500	2500	2500	2500	2500	2500	2500		375	25375
11															
12	Proposed Fee	0	2500	4000	2500	3500	4000	2500	4000	2500	3000	3000			31500
13	(support hours - 3) x \$75	0	0	956.25	0	112.5	1275	0	1312.5	0	1912.5	768.75			
14	total	0	3000	4956.25	2500	3612.5	5275	2500	5312.5	2500	4912.5	3768.75			38337.5
15															

Using a full years worth of work logs we were able to see that the amount of time we were putting into our publishing services work wasn't being recouped by the \$2,500 dollar annual fee, which was a bit of an arbitrary figure to begin with because we weren't sure of the real costs of running the service when we started. Given our data we saw that we needed at to cover at least one half time position. We debated various pricing models including charging hourly in addition to our base fee, but managing billable hours can get messy and we noticed that there was a strong correlation between the number of journals a library supported and the amount of support hours dedicated to that instance.

Time tracking

	A	B	C	D	E	F	G	H	I	J	K	L
1		ALGOMA	BROCK	GUELPH	LAURENTIAN	OTTAWA	QUEENS	RYERSON	TORONTO	UOIT	WESTERN	WINDSOR
2	# of active journals included in base price	0	5	5	2	5	5	5	5	2	5	5
3	# of active journals not in base price	0	1	3	0	7	5	0	29	0	5	2
4	# of inactive journals	0	2	11	0	3	9	0	10	0	2	2
5	TOTAL JOURNALS	0	8	19	2	15	19	5	44	2	12	9
6												
7	annual base price	2500										
8	extra active journal price	200										
9	extra inactive journal price	50										
10												
11	base price	0	2500	2500	2500	2500	2500	2500	2500	2500	2500	2500
12	extra journals @ \$200 / title	0	200	600	0	1400	1000	0	5800	0	1000	400
13	extra journals @ \$50 / title	0	100	550	0	150	450	0	500	0	100	100
14	total	0	2800	3650	2500	4050	3950	2500	8800	2500	3600	3000
15												
16	TOTAL	37350										
17												

In the end we went with a base price plus number of journals approach, which we proposed to our members and nobody seemed bothered by it because we were able to show exactly how much work goes into sustaining the service for each member.

Defining roles, responsibilities & boundaries



Working with our members is our main focus and we enjoy developing our relationships with them. Each of our relationships is unique and we like to think that we offer a bespoke level service geared to the specific needs of each library. But, as membership grows, as well as the number of journals each member hosts, it's become important to have clearly defined roles and responsibilities for everyone involved.


It's important to establish the roles early on and clarify who is responsible for what because defining roles is easy, but redefining roles is hard. For example, we've had cases where editors will start to contact us directly for support and it can be hard to go back. Sometimes it's easier to just work with the editors directly, but it sets a bad precedent because we simply wouldn't be able to handle that amount of support work. This is why we insist on only working with scholarly communications librarians who do the remarkable job handling all of the training, journal setup and management aspects of publishing while we support them on the systems side. Nevertheless situations arise where the librarian we work with goes on leave or moves to a new position and there's a gap in service. In those situations we do offer to work directly with the journals with the understanding that it's only temporary.

External communities

PKP|COMMUNITY FORUM

Q

≡



all categories ▾

all tags ▾

Categories





Latest

New (12)

Unread (8)

Top

+ New Topic

Category	Topics	Latest
Announcements This category is for PKP staff to post announcements of upcoming events, new software releases, etc.	72 2 new	 Missing Plugin Gallery? • ■ Questions 1 17m
FAQs Go here to find answers to frequently asked questions.	21	 Justify alignment for How to Cite, Copyright and reference section • ■ Questions ojs3 1 21m
Questions Add your questions here about how to use PKP software, about editorial workflow, how to install, upgrades, etc. English is PKP's primary language, but feel free to post in other languages	8113 8 unread 6 new	 Article viewers without journal header/footer • ■ Questions 0 28m
OMP Topics This category is for questions and topics about Open Monograph Press	372	 "ojs2: 404 not found" appears in php_error_log thousands of times per day 7 44m

One of the best things about working with OJS and open source software in general is the community. We encourage our members to get involved in the broader journal hosting community with things like mailing lists, slack channels, and forums. We don't always have the answer to questions that come up, especially because we don't typically work with editors or within the editorial process, so involvement in these communities is important. Nothing makes me happier than getting an email identifying a bug that includes the forum thread discussing the issue AND the github pull request for fixing it.

And now since we're on the topic of fixing code, let's segue into code management.

Code management

Code management

Deciding when to upgrade

- Security issues?
- What are the new features?
- Minor or major release?
- Known bugs?

When thinking about code updates, we work closely with institutions and monitor the latest developments from PKP so we can make judgment calls about when to upgrade.

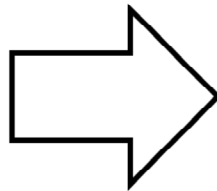
Some things that we think about include: are there security issues? What are the new features? is it a minor release? Or are there known bugs in the current release?

If it's a security issue we obviously prioritize that immediately, but minor bugs we tend to fix on an as per need basis. Often times if something comes up and there's a patch, we'll just patch the reporting instance because those fixes will most likely be in the next code update. When it comes to major release updates each member is different. Some might be eagerly waiting on new features while others like wait a few months because they might not want to potentially disrupt a busy editorial cycle. From experience we like to wait too because there's always a few kinks that quickly get fixed after each major release, but again if a member is eager then we work with their needs.

Code updates

OJS 3.1.2

1 Git repo
per instance
(12 repos!)



OJS 3.3

1 Git repo
for all

At the moment each of our members has their own code repository because of the inevitable code customizations that have accumulated over the years. This means that updating code has been a manual process where we pull from PKP and merge local changes into our repos because it preserves all of the customizations. With 12 instances this takes a bit of time and so with our latest upgrade from version 3.1.2 to 3.3 we're finally moving to a single code base shared by all of the instances. This is no simple feat and it's taken us years to get to a point where everyone can use the same codebase.

Code customizations

- Institutions request individual customizations
- Every code customization has a cost associated with it when you upgrade
- Consider:
 - Could this be useful to the larger project community?
 - How large of a change is this?
 - Does it create technical debt?

Sometimes institutions will request code customizations for their instances. In the past we allowed for most customizations requested, but as the service has grown managing these has been challenging. It's important to remember that every code customization has a cost associated with it when we're upgrading because we need to re-apply or merge these changes with each upgrade.

With all of that in mind, we're trying to rethink how we handle customizations. Some of the things that we think about include:

Could this be useful to the larger project community? Can we submit this back to the main project?

How large of a change is this? Can it be contained within a custom theme plugin?

And finally, does it create technical debt? Technical debt is essentially kicking the can down the road where easy fixes today become a burden in the future.

And now Kaitlin will be discussing the nitty gritty details of how we deploy our code in our next section on Environments and Deployment.

Environments and deployment

Environments and deployment

[kaitlin start]

Environments

- Local & development environments
- Environment configuration
- Multi-server -> shared server

In the past we used to do our test upgrades locally, but as the service has grown we can't always do this on our own machine -- some of the reasons that this can be challenging include the sizes of files on each instance which are too large for a small laptop hard drive, and the length of time that an upgrade can take to run, which requires a more reliable environment than a 2015 macbook.

A development environment allows us to test upgrades more easily, and to open testing to users before updating production - this is especially valuable when a new version of the software is released that introduces major changes to the user experience. This is also a space for each institution to work on custom developments in an environment that mimics their production space.

One thing to note is that you have to make sure to set up the development environment properly; this means things like turning off system emails, making sure no data is being sent out, and ideally limiting the environment to certain IP addresses so users don't find it accidentally and it isn't crawled by other websites.

When we migrated to OJS 3 we made our production environment more efficient by moving from individual servers for each instance to a shared server, which works for OJS because it's not that resource intensive from a technical perspective. By moving to a single server, we removed the need to manage 12 servers for the service and reduced the maintenance workload for our systems team. It also ensures that we don't have differences between servers that can cause issues, like a package that's

been installed in one server but not another.

Upgrading OJS 3 instances on OJS server

Once the code is updated in GitLab and you've verified the submodule commits:

1. Set permissions of the ojs directory to the ojs user:

```
sudo chown -R ojs:users the-ojs-directory/
```

2. `sudo su - ojs` to login as the OJS user which has access to pull from GitLab

3. cd to the OJS directory

4. If existing branch:

```
git pull
```

If new branch:

```
git fetch
git checkout -b stable-3_1_X origin/stable
```

5. Reset the submodule directories so we don't have issues updating them:

```
git submodule foreach git reset --hard
```

6. Update the submodules:

```
git submodule update --init --recursive
git submodule <- check to see everything l
```

7. Run composer updates:

```
composer --working-dir=lib/pkp update
composer --working-dir=plugins/paymethod/p
composer --working-dir=plugins/generic/cit
```

8. Get NPM dependencies (run in the main OJS directory):

```
npm install
npm run build
```

This is a snapshot of our deployment documentation for OJS. In the past we've relied on lengthy documents that outline different steps in the upgrading process - this process may be familiar to you if you've been responsible for managing hosted software. This document outlines all of the different commands that we ran to upgrade OJS on a production server, and has many more steps beyond what's in this screenshot.

The problem with this process is that it's prone to human error, as it can be easy to enter a command incorrectly or miss a step. While we have backups in place, it adds a lot of time to the process if we need to restore something and is tedious and time-consuming to do multiple times. To improve on this process, we've started using a tool called Jenkins.

Jenkins

- automation server for deploying and automating tasks
- self-hosted
- <https://jenkins.io>



Jenkins

Jenkins is an open source automation server for deploying code and automating tasks. At Scholars Portal we host Jenkins and use it to across many of our applications, and have started to use it with OJS.

People

Build History

Project Relationship

Check File Fingerprint

My Views

Open Blue Ocean

Build Queue

No builds in the queue.

Build Executor Status

master

1 Idle

2 Idle

Mojito

1 Idle

OJS-DEV

1 Idle

AdminTool NodeJS

All

Books Backend

Books FrontEnd

Journals

ML-Resources

Odesi NodeJS

SwiftBrowser

S	W	Name	Last Success	Last Failure	Last Duration	Fav
		AdminTool NodeJS DEV	4 days 6 hr - #119	1 yr 3 mo - #92	3 min 20 sec	
		AdminTool NodeJS PROD	4 days 6 hr - #32	4 days 6 hr - #31	3 min 7 sec	
		Books Backend DEV	4 hr 19 min - #291	1 mo 5 days - #283	7.6 sec	
		Books Backend PROD1	1 hr 13 min - #39	N/A	6.9 sec	
		Books Backend PROD2	1 hr 28 min - #39	N/A	7.2 sec	
		Books Backend STAGING	1 hr 32 min - #50	1 yr 1 mo - #36	6.6 sec	
		Books Frontend DEV	4 hr 25 min - #208	1 yr 8 mo - #142	2 min 34 sec	
		Books Frontend PROD1	1 hr 13 min - #32	N/A	2 min 29 sec	
		Books Frontend PROD2	1 hr 28 min - #33	1 yr 6 mo - #11	2 min 30 sec	
		Books Frontend STAGING	1 hr 32 min - #45	4 mo 11 days - #40	2 min 38 sec	
		Ebrary Deploy - Loader	35 min - #77	2 mo 22 days - #62	21 sec	
		Journals Staging	1 day 1 hr - #132	4 days 1 hr - #130	2.3 sec	
		ML Resources - TEST	1 mo 23 days - #44	1 mo 23 days - #42	5.8 sec	
		Odesi NodeJS TEST	2 mo 18 days - #215	11 mo - #204	8.9 sec	
		OJSDEV- Create OJS Dev Instance	1 mo 12 days - #85	3 mo 16 days - #81	55 min	
		Swiftbrowser TEST	4 mo 12 days - #171	8 mo 10 days - #166	7 sec	
		Update Ebook Scripts	8 days 3 hr - #273	2 yr 8 mo - #5	3.5 sec	
		Update IP-Updater	2 mo 10 days - #15	N/A	2.1 sec	
		Update LIMA Scripts	1 mo 24 days - #108	N/A	2.1 sec	
		Update OJS Harvest scripts	1 min 31 sec - #2	N/A	3.8 sec	

This is a screenshot of the Jenkins interface, with a list of different jobs we have available in the system. Each job represents a script that strings together different commands. Jobs perform different functions, like deploying code to a server or running a script to download data. Without a tool like this, someone would have to run various commands individually, and the process would be more time-consuming and prone to human error. Jenkins provides a number of other valuable features, like scheduling jobs to run at certain times and rolling back to a previous version when there's an error.

Project OJSDEV- Create OJS Dev Instance

This build requires parameters:

INSTANCE_NAME

Choose Instance to Clone from Prod and Deploy to Test

BRANCH_TAG

To build a particular Branch use : refs/heads/(branchName)
Eg. refs/heads/master

To build from a Particular Tag use : refs/tags/(tagName)
Eg. refs/tags/git-2.3.0

To build from a Particular Commit use : (commitId)
Eg. 5062ac84

☐ GET_LATEST_DB

Get Latest Database from Production and Restore to DEV

Note: It will take upto 40minutes to Run this step depending on DB Size!

☐ RUN_UPGRADE

Check this if you want to Run Upgrade step

OUTPUTDIR

DO NOT CHANGE.

BUILD

DO NOT CHANGE.

Build

In this screenshot of our Jenkins workflow for our development environment in OJS, we have a drop-down menu that allows us to select which school's instance we're upgrading. We can then enter the code branch, in this case for version 3.1.2. We have a checkbox for pulling in the latest database from our production environment, and another for if we want to run the upgrade script.

We're currently only using this in our development environment, but we hope to roll this out to our production environment after we test it out further with our next upgrade. While this change might seem minor at first glance, it will save us a significant amount of time when we have to upgrade our OJS instances. Instead of running a series of commands 12 different times, we can do the same series of tasks with a few clicks on a form.

Community contribution

The last topic we'd like to touch on is getting involved in the larger project or community of your service. In order to grow services that are built on an open source code, it's vital to give back to that project when and if you can. Giving back to the open source software that we build our services on both helps our own users and builds on the sustainability of the software for the long-term. Many open source projects are facing their own challenges with limited resources.

Community contribution

- PKP Forums
- Code and documentation
- Working groups & communities
- Financial support
- Supporting broader initiatives around open infrastructure & publishing, e.g. Library Publishing Coalition, Invest in Open

In the case of our publishing services at Scholars Portal, this means finding ways to contribute back to PKP and the user community.

This can include things like:

- participating in the PKP forums to help other community members
- contributing code or documentation back to the project, especially when our users find bugs that we can resolve
- being involved in working groups and communities; for instance, we're involved in PKP's documentation and metadata groups
- contributing financially back to PKP - thankfully many institutions in Ontario have been willing and able to contribute to PKP financially; this can also give us more opportunities to advocate for our community's needs
- Finally, another way we can support scaling scholarly infrastructure is by supporting broader initiatives in this space, like the Library Publishing Coalition
 - Another example is the Invest in Open project, is "an initiative dedicated to improving funding and resourcing for open technologies and systems supporting research and scholarship." The Invest in Open Infrastructure is one example of how we can broadly support efforts towards more sustainable and scalable scholarly infrastructure.

Questions?

ojs@scholarsportal.info

That brings us to the end of our talk. If you'd like to contact us, you can email us at ojs@scholarsportal.info.