2019-06-17 - F2F Session 1 (Production Deployment)

Date

17 Jun 2019

Attendees

- Wayne Schneider (Index Data)
- Jakub Skoczen (Index Data)
- Patty Wanninger (EBSCO)
- Jason Root (TAMU)
- Richard Redweik (Leipzig)
- Steffen Köhler (Leipzig)
- Philip Robinson (Cornell)
- Michelle Suranofsky (Lehigh)

Remote Attendees (by Zoom):

- Catherine Smith
- Ingolf Kuss
- Jack Hill

Goals

- FOLIO Production Deployment / Technical Discussion

Discussion Topics for Production Deployment Sessions 1 & 2

Time	Item	Who	Notes
	System Architecture for Production		
	Dependency Resolution		
	Security		
	Upgrades/Updates		

Notes (from Patty Wanninger)

Overall architecture

FOLIO is made of micro services that communicate through an API gateway. It relies on HTTP messaging between modules to get things done.

4 main components/layers

1. OKAPI API gateway
2. Back end modules that process and store business data
3. Database layer – not required to be monolithic but is right now. Each module can have own database. Storage persistence layer is logically separated by module and tenant. Can run it on one database – most are doing that now. Not preferable
4. Ui/UX javascript front end – would be nice to have dynamic ability to select modules. FOLIO has collection of APPS that gets released all at once. Effectively a static collection. Deployment will become more complicated if doesn't deploy the bundle.

Discussion: Production deployment and system architecture

Architecture is coalescing around plans to deploy in cluster of containers. Not a lot done to support other kinds of deployment. Could run each module in its own part of a big server.

Containers talk to a web service. Dependency management shift over to developers. Containers are portable. Value to the portability.

Community edition and enterprise edition of Docker. Docker gives you an ecosystem of ready to deploy containers.

Kubernetes is an abstraction layer over docker so you can orchestrate your deployment.

Stripes: In production environment, near to medium term, will be dealing with static bundles. Can be prepackaged in a hierarchy by FOLIO – platform core circ + inventory and platform core+ everything else. Can build own platform but there are a lot of inter-dependencies in the core of the UI with password validator and inventory storage. Some apps make sense to work on their own.

Okapi: need to run in high availability mode. Need multiple instances of the thing running. There's no "state" for many modules. But Okapi has STATE and needs to share information among all its instances. Okapi uses Hazelcast to do this.

In production need to run Okapi in clustered mode with shared storage – supports Postgres theoretically it still supports MongoDB but won't be recommended.

Not a smooth way to upgrade a Hazelcast Okapi cluster. It requires system downtime. Lots of Okapi upgrades. Hazelcast piece – they all search for a while and just can't join back together. Community edition of Hazelcast is the problem. Paid Hazelcast might solve the problem.

Is there a ticket for this? Jakub will make sure there is a ticket for this.

For go-live, downtime is not ideal. OKAPI should be stable at go-live. Voyager makes people used to downtime for upgrades.

Temporary fix is to pin Hazelcast version into Okapi. Only a problem if Hazelcast upgrades. ? Jason says he sees inconsistencies. No data loss, just loss of communication. Okapi persistence doesn't change very much.

Okapi production deployment is defining a workload with a number of instances and making them be a part of Hazelcast. (Jason at TAMU)

Do you use Okapi to deploy your system. Okapi uses docker API to deploy which is fine – but if you are running Okapi in a pseudo-stateless way in a container – that's where Kubernetes comes in.

Okapi can talk to docker API and deploy containers on a docker server. We could add a feature where O talks to Kubernetes API but still Okapi would control what software is running in your cluster. Would simplify bringing up a production environment. Kubernetes is open source. If project were to work on Okapi, would be to add API access to Kubernetes rather than adding things to Okapi.

Enabler for Okapi to orchestrate upgrades –some workflow that okapi can stand up a new version of the module, hook up the database. But then okapi is in charge.

How to come up with process that most institutions can adopt – actual scripts, high level description.

Now would just implement small workflow in Okapi now . But you might want to "do it yourself." Just want to use Okapi as API and dependency tool, want to have more control over deployment.

Don't want to use Okapi as orchestration tool (Jason). Does the project offer any tooling as part of the project? There will be multiple steps. Jason's ok with Okapi triggering data migration but if it hoses my database that is a problem.

Production for deploying modules: workloads of clustered modules that share a persistence layer. How do we do HA High Availability on the persistence layer? AWS has paid services avail but on premise – Postgres has HA ability. Chris has looked at postgres solution that looks at several copies of postgres at the same time. HA databases in some flavor that can run postgres is OK. Running multiple copies of the module that address the database is not a problem.

Marccat will need its own database per tenant.

Could run separate DB for each module. Limitation that can run each tenant with own database – architectural limitation of RMB. Separate backend is needed – there is issue for that in FOLIO.

Change to run multiple database – but need to scale postgres connections – can't pull connections from separate database – explode the need for connections. That decision needs to lie on devops side. Configuration limitation now. Can't run multiple databases using the same back end. What's the better way. Running different copies of same module is a computer cost.

Multi-tenancy is good for testing, etc. using feature in production and use it for PR previews – have facility where you issue a PR and spin up tenant with the feature.