EMCO In Production

There have been asks from partners and prospective customers regarding the feasibility of deploying EMCO in production. There are some areas where EMCO needs enhancements to get it closer to production. Hopefully, the community can come together and contribute to an initiative that identifies the gaps, determines the enhancements needed to fill those gaps, and delivers those enhancements across multiple EMCO releases.

To get EMCO closer to production state, two important areas that need enhancements are Observability and Resiliency.

Observability

Observability is the property of a system that allows an external observer to deduce its current state, and the history leading to that state, by observing the externally visible attributes of the system. The main factors relating to observability are: logging, tracing, metrics, events and alerts. See the Observability page for more details.

For logging, we already have structured logs in the code base and fluentd in deployment. But:

- We need EMCO to work with a logging stack such as Elastic Search, Fluentd and Kibana. (e.g. see this EFK article).
- We need a good way to persist the logs.

These don't need any code changes. We can do a PoC for a deployment of EMCO with log persistence and a log stack, and document the ingredients and the recipe. Perhaps, the needed YAML files can be checked in as well.

We also want to add user id to logs - this requires configuring Istio Ingress to pass the user id, and making EMCO code changes to include user id in the logs.

We also need to investigate an events framework. This is TBD.

Resiliency

Database persistence

Today, db persistence defaults to local storage. We need cluster-wide storage for different use cases. So, we need to validate with cluster-wide persistence and provide a reference solution ('experience kit') for easily deploying mongo/etcd/etc. with cluster-wide storage. The experience kit will ideally have documentation in EMCO repo and perhaps a set of YAML files that can be applied to configure NFS with appropriate PVs.

- We have tested with NFS-based PV in the past, and we still have the NFS-related YAMLs. So there is consensus around using NFS as the default cluster-wide storage..
- With persistence enabled, we cannot rely anymore upon developer-oriented troubleshooting and workarounds based on re-installing EMCO to blow away the db. Developers should also
  test with persistence enabled.

Recovery from crashes/disruptions

Scenarios to validate:

- Restart each microservice, when it is processing a request
- In particular: Restart orchestrator when a DIG instantiate request is in flight
- Restart all microservices together
- Restart the node on which EMCO pods are running (assuming it is 1 node for now)

rsync can restart after a crash. Aarna, as part of EMCO backup/restore presentation, has tested blowing away the EMCO namespace (incl. EMCO pods and db), and restoring it.

Some known gaps:

- Mongo db consistency: Some microservices may make multiple db writes for a single API call. So, if the microservice crashes in the middle of that API call, we will have an inconsistent update in mongo. We need to scrub for such scenarios and fix them.
- Resource Cleanup: The orchestrator creates entries in the appcontext during DIG instantiation; it needs to cleanup any stale context on restart after a crash.

Graceful handling of cluster connectivity failure

Without the GitOps model, rsync should apply configurable retry/timeout policies to handle cluster connectivity loss. We have the `/projects/.../{dig}/stop` API but that is a workaround -- the user needs to invoke that API manually. rsync has retries/timeout for cluster connectivity, and we considered validating it.

However, the non-GitOps approach has many issues. So, it is preferable to make the GitOps approach fully functional (ideally in 22.09), and deprecate other apparoaches later.

So, there is no need to validate rsync retries/timeout for cluster connectivity.

Storage Considerations

We need storage in the cluster where EMCO runs for:

- mongo db
- etcd
- logs
- metrics? (assuming we aggregate cluster metrics in the central cluster via federated Prometheus, Thanos, Cortex, ...)
- ?

TBD: Enable db replication and/or sharding. This requires further thought.

Upgrades

It should be possible to upgrade in-place from one released version to the next released version. The primary concern is any database schema changes between versions.

A smaller concern is a deprecation schedule for removing APIs and/or components.

- It would be ideal to make schema changes without breaking backwards compatibility - add new fields, don't rename/delete fields, etc.
- For schema migration, the standard way is to provide a (set of) script(s) that are run to update the db.

This needs to be taken up for consideration.

22.09 Release Tasks for Productizability

- Tracing for core components (orchestrator, rsync, clm and dcm)
- Metrics for core components (orchestrator, rsync, clm and dcm) - HTTP request rate etc., error counters, etc. based on this discussion.
- Experience kit for Logging (EFK stack): documentation with YAMLs
- Experience kit for cluster-wide persistence with NFS (for mongo, etcd, logs and m3db): documentation + YAMLs
- Validation that each microservice can restart with integrity after a crash
- Investigation of which microservices make multiple db writes for a single API call

Also, start planning for:

- Considerations for acquiring db lock for multiple db writes in a single API call
- Upgrades