Winventory: microservices architecture case study

Sebastian Bukowiec[1,*] and *Pawel Tadeusz* Gomulak[2,**]

[1]CERN
[2]Lodz University of Technology

Abstract. In the CERN laboratory, users have access to a large number of different licensed software assets. The landscape of such assets is very heterogeneous including Windows operating systems, office tools and specialized technical and engineering software. In order to improve management of the licensed software and to better understand the needs of the users, it was decided to develop a Winventory application. The Winventory is a tool that gathers and presents statistics of software assets on CERN Windows machines and facilitates interaction with their individual users. The system was built based on microservices architecture pattern, an increasingly popular approach to web application development. The microservices architecture pattern separates the application into multiple independently deployable units that can be individually developed, tested and deployed. This paper presents the microservices architecture and design choices made in order to achieve a modern, maintainable and extensible system for managing licensed software at CERN.

1 Introduction

At CERN we have a large number of applications that are used by different user groups in the organization. Many of these applications require some type of administrative action like phase-out or license management. We do gather metrics about the applications or licenses usage but more information is required to understand the different use cases. The Winventory system was built in order to help in these administrative actions by facilitating the contact with the users and to understand the various use cases for these applications. Additionally, we wanted to identify and track Windows operating system versions that approach their end-of-life. In 2019 it was the case for Windows Server 2008 and in the coming years, other versions will follow this pattern.

The objective of the Winventory project is to gather user input and build a comprehensive inventory of software assets across CERN. The system is built on a microservices architecture pattern, which separates the application into multiple and independently deployable units that can be individually developed, tested and deployed.

This paper presents details of the architecture and design decisions taken in order to achieve a modern, maintainable and extensible system for managing licensed software at CERN.

[*]e-mail: sebastian.bukowiec@cern.ch
[**]e-mail: 190091@edu.p.lodz.pl

2 Architecture

As architectural pattern for the Winventory system, we selected the microservices software development techniques. The overview of the Winventory architecture is shown in Figure 1. In Winventory we successfully connected microservices based on two different frameworks Flask and .NET Core written in two different programming languages Python for the former and C# for the latter.

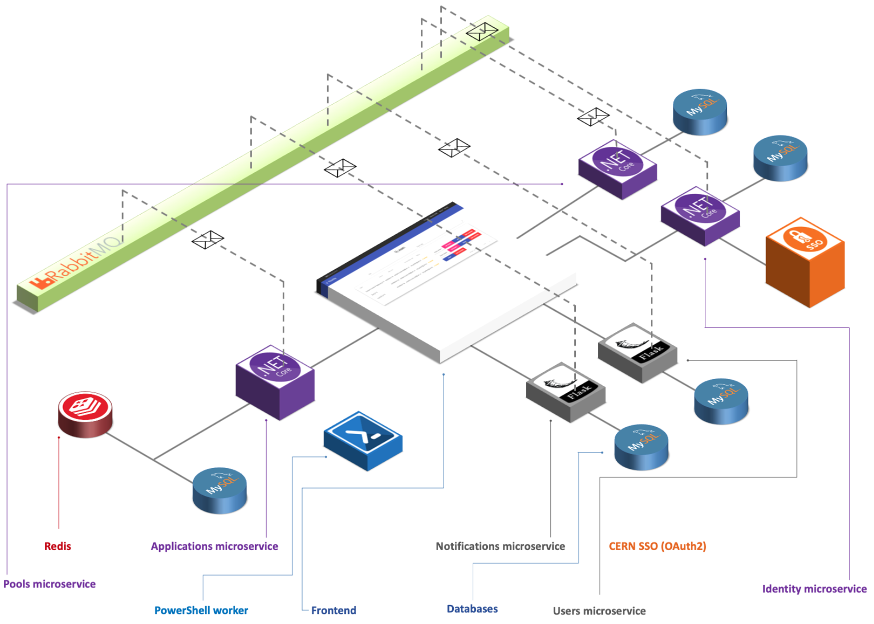
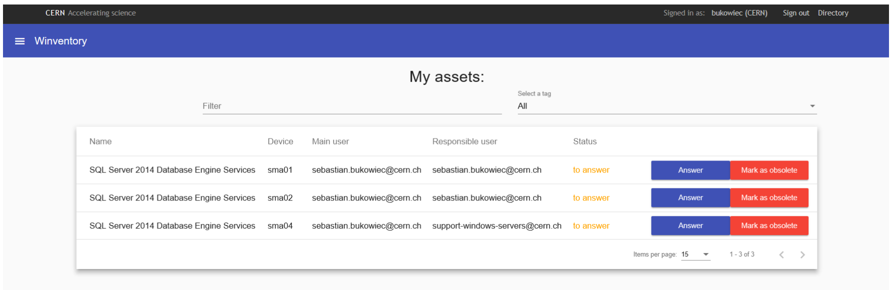


Figure 1. The Winventory system architecture

The Winventory currently has one data producer written in PowerShell. The data about specified software and device where it is installed is taken from the database of the Computer Management Framework (CMF), a software created at CERN and installed on every Windows machine that is a member of the CERN domain. Data not older than three months is collected once a day and pushed to the Winventory system. The data producer is deployed in the orchestration system described in the following paper [1].

The Winventory business logic consists of several backend services. Each backend service has a REST API and its own private datastore, which facilitates loose coupling and independent development. The backend services include the following microservices:

- Applications - manages information about applications and devices where they are installed. Besides MySQL datastore, the service uses the Redis in-memory database for caching.
- Polls - manages polls sent to the user and gathers the answers.
- Identity - authenticate the user against the CERN SSO, manages role-based authorization and JWT tokens.
- Notifications - manages e-mail communication with the users including templates and aplication grouping.


CERN Accelerating science — Signed in as: bukowiec (CERN) Sign out Directory

Winventory

My assets:

Filter | Select a tag: All

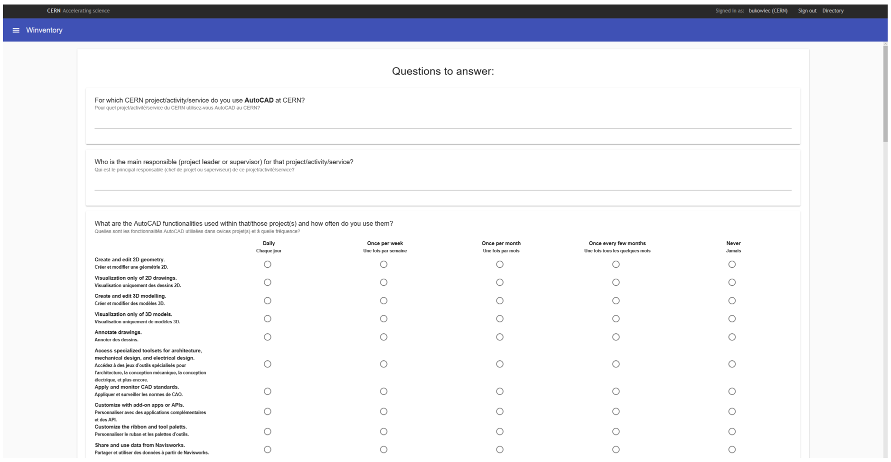
Name	Device	Main user	Responsible user	Status		
SQL Server 2014 Database Engine Services	sma01	sebastian.bukowiec@cern.ch	sebastian.bukowiec@cern.ch	to answer	Answer	Mark as obsolete
SQL Server 2014 Database Engine Services	sma02	sebastian.bukowiec@cern.ch	sebastian.bukowiec@cern.ch	to answer	Answer	Mark as obsolete
SQL Server 2014 Database Engine Services	sma04	sebastian.bukowiec@cern.ch	support-windows-servers@cern.ch	to answer	Answer	Mark as obsolete

Items per page: 15 1 - 3 of 3

Figure 2. The Winventory application frontend assets view

• Users - manages the information about the users and groups responsible for the devices where the applications are installed.

Single Page Application frontend is based on Angular 7 with Angular Material providing features like lazy-loading, XSS protection, HTTP interceptors, dependency injection, and routing. The exemplary screenshots from the Winventory application are presented in Figures 2 and 3.


CERN Accelerating science — Signed in as: bukowiec (CERN) Sign out Directory

Winventory

Questions to answer:

For which CERN project/activity/service do you use **AutoCAD** at CERN?
Pour quel projet/activité/service du CERN utilisez-vous AutoCAD au CERN?

Who is the main responsible (project leader or supervisor) for that project/activity/service?
Qui est le principal responsable (chef de projet ou superviseur) de ce projet/activité/service?

What are the AutoCAD functionalities used within that/those project(s) and how often do you use them?
Quelles sont les fonctionnalités AutoCAD utilisées dans ce/ces projet(s) et à quelle fréquence?

	Daily Chaque jour	Once per week Une fois par semaine	Once per month Une fois par mois	Once every few months Une fois tous les quelques mois	Never Jamais
Create and edit 2D geometry. Créer et modifier une géométrie 2D.	○	○	○	○	○
Visualization only of 2D drawings. Visualisation uniquement des dessins 2D.	○	○	○	○	○
Create and edit 3D modelling. Créer et modifier des modèles 3D.	○	○	○	○	○
Visualization only of 3D models. Visualisation uniquement de modèles 3D.	○	○	○	○	○
Annotate drawings. Annoter des dessins.	○	○	○	○	○
Access specialized toolsets for architecture, mechanical design, and electrical design. Accédez à des jeux d'outils spécialisés pour l'architecture, la conception mécanique, la conception électrique, et plus encore.	○	○	○	○	○
Apply and monitor CAD standards. Appliquer et surveiller les normes de CAO.	○	○	○	○	○
Customize with add-on apps or APIs. Personnaliser avec des applications complémentaires et des API.	○	○	○	○	○
Customize the ribbon and tool paletts. Personnaliser le ruban et les palettes d'outils.	○	○	○	○	○
Share and use data from Navisworks. Partager et utiliser des données à partir de Navisworks.	○	○	○	○	○

Figure 3. The Winventory application frontend survey view

3 Microservices patterns

Microservices are independently deployable services modeled around a business domain. They communicate with each other via networks, and as an architecture choice offers many options for solving the problems you may face. It follows that a microservice architecture is based on multiple collaborating microservices. Microservices also have the advantage of being technology agnostic [2]. Such characteristic provides greater flexibility in at least two aspects. First, the appropriate tool can be selected based on the problem to be solved. Second, at CERN the rotation of the personnel is quite high and some people are in the organization only for a short period. In such a scenario it is not ideal to learn new technology but instead leverage the existing knowledge in writing higher quality code. Additionally working on a much smaller codebase has a higher probability that the assigned task will be completed.

3.1 Decomposition strategy

A key characteristic of the microservice architecture is that the services are loosely coupled and communication happens only via exposed APIs and dumb pipes. To achieve loose coupling each microservice has its own database. Such an approach allows for an independent development that will not be blocked by another service. To avoid unnecessary traffic some data is replicated across multiple microservices. It is worth mentioning that not all data is duplicated but only fragments that are required for the given microservices to work independently. In the microservices architecture, it is tolerated to have duplicated data in opposition to the monolith systems where such behavior is not desired.

3.2 Communication patterns

The communication between microservices in the Winventory systems is done in an asynchronous way using event base communication build on the top of message broker which in our case is RabbitMQ. This is one of the most common asynchronous patters and its advantage is simplicity in the implementation comparing to other popular patterns like Command Query Responsibility Segregation (CQRS) or Event Sourcing. The microservices communicate via message broker over publish-subscribe pattern. It works well in our case as a message is delivered to all of the attached consumers. As an example when a user provides a response to the survey, the Poll microservice will verify it and store it to the database then it will send a message over message broker to Applications and Users microservices to update their state.

For the communication between microservices, asynchronous communication is used to implement a circuit breaker pattern using Polly library [3] for higher resilience and fault-tolerance. Using the circuit breaker we can specify the number of retries in case of failure, the time between retries. We found it very useful during deployment as even if all microservices should be available at the same time, in practice some delays are also possible but repeating the request within time windows specified in the circuit breaker allows to wait until application is available instead of receiving an error.

3.3 Push notifications

A SignalR Hub was implemented that subscribes to the message broker and pushes the content via the push notification functionality to the web client based on the received events. SignalR [4] is a software library for Microsoft ASP.NET that allows server code to send asynchronous notifications to client-side web applications. The library includes server-side and client-side JavaScript components.

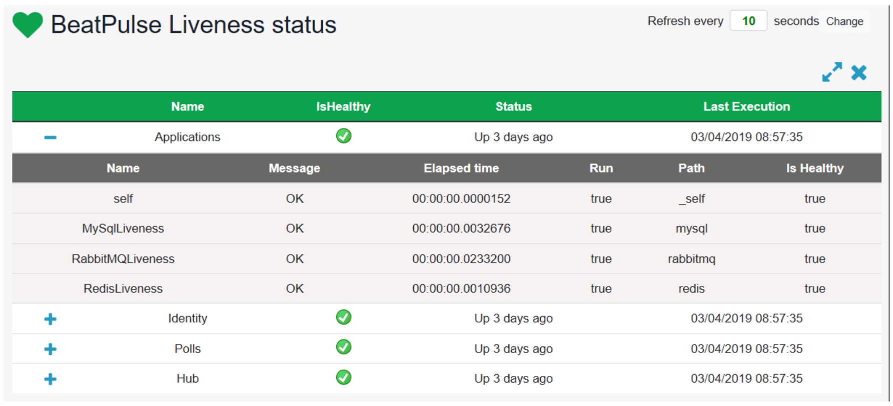
It was desired to provide a real-time update to the website when it is already loaded, without requiring the page reload as the system uses event notification and is asynchronous.

3.4 Continuous Deployment and monitoring

The GitLab CI/CD pipeline is used to automatically build new Docker images and deploy them to the OpenShift Container Platform after the new code is pushed to the Git repository.

To facilitate local development, a Docker Compose is used to build and run the Winventory multi-container application locally on the developer's computer.

Monitoring is a crucial component of every system. In the microservices architecture, it is more difficult to implement good monitoring as this is a distributed system. In the Winventory project, we benefit from a built-in health check system called Beat Pulse Liveness status. The web interface of the system is shown in Figure 4.



BeatPulse Liveness status

Refresh every 10 seconds Change

	Name	IsHealthy	Status	Last Execution
−	Applications	✓	Up 3 days ago	03/04/2019 08:57:35

Name	Message	Elapsed time	Run	Path	Is Healthy
self	OK	00:00:00.0000152	true	_self	true
MySqlLiveness	OK	00:00:00.0032676	true	mysql	true
RabbitMQLiveness	OK	00:00:00.0233200	true	rabbitmq	true
RedisLiveness	OK	00:00:00.0010936	true	redis	true

	Name	IsHealthy	Status	Last Execution
+	Identity	✓	Up 3 days ago	03/04/2019 08:57:35
+	Polls	✓	Up 3 days ago	03/04/2019 08:57:35
+	Hub	✓	Up 3 days ago	03/04/2019 08:57:35

Figure 4. The web BeatPulse Liveness interface of the Winventory system

4 Conclusion

The microservices architecture has many benefits, but it also brings additional challanges. Among the benefits we must mention the independent scalability and deployment. Smaller codebase per microservice is easier to maintain, but with an increased number of microservices we had to re-think how this code should be organized. We started the project with multiple repositories as it was easier to deploy, but over time we moved to monorepo as the maintenance of many repositories by a small team is harder, especially during heavy development.

Continuous integration and continuous deployment are extremely important and they should be considered from the very beginning of the project.

Another big benefit of this architectural pattern is technology freedom. In the scientific environment like CERN where teams consist of people with different skillsets, technology backgrounds, and code practices habits, it is important to provide a way to foster experimenting and innovation.

Future plans include improving tooling around the system with the focus on the observability.

References

[1] S Bukowiec, R Gaspar, T Smith 2017 Windows Terminal Servers Orchestration, J. Phys. Conf. Ser. 898 (2017), 082025

[2] Sam Newman, *Monolith to Microservices* (O'Reilly Media, Inc., 2019)

[3] The Polly Project, *.NET resilience and transient-fault-handling library*, http://www.thepollyproject.org/, Accessed: 2020-02-12

[4] The SignalR, *Real-time ASP.NET*, https://dotnet.microsoft.com/apps/aspnet/signalr, Accessed: 2020-02-12