ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA123

VACUUM CONTROLS CONFIGURATOR: A WEB BASED CONFIGURATION TOOL FOR LARGE SCALE VACUUM CONTROL SYSTEMS

A. Rocha†, I. Amador, S. Blanchard, J. Fraga, P. Gomes, G. Pigny, P. Poulopoulou, C. Lima
CERN, Geneva, Switzerland

Abstract

The Vacuum Controls Configurator (vacCC) is an application developed at CERN for the management of large-scale vacuum control systems. The application was developed to facilitate the management of the configuration of the vacuum control system at CERN, the largest vacuum system in operation in the world, with over 15,000 vacuum devices spread over 128 km of vacuum chambers. It allows non-experts in software to easily integrate or modify vacuum devices within the control system via a web browser. It automatically generates configuration data that enables the communication between vacuum devices and the supervision system, the generation of SCADA synoptics, long and short term archiving, and the publishing of vacuum data to external systems. VacCC is a web application built for the cloud, dockerized, and based on a microservice architecture. In this paper, we unveil the application's main aspects concerning its architecture, data flow, data validation, and generation of configuration for SCADA/PLC.

INTRODUCTION

In the early 2000's, during the construction of the LHC and anticipating a considerable increase in the number of vacuum devices to be controlled, a software application (vacDB-Editor) and a set of databases (vacDB) were developed to homogenize and automate the configuration of the SCADA and PLCs for CERN's vacuum systems. Figure 1 shows a simplified overview of this application with its main building blocks.

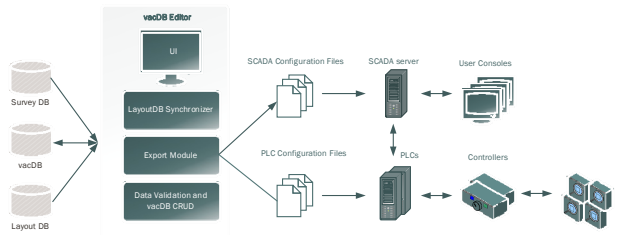

Figure 1: vacDB-Editor overview.

Users interact with the vacDB-Editor user interface, where they can modify the configuration of the control system (adding/removing equipment, modifying equipment attributes, configuring vacuum sectors, alarms). After validating user input, data is persisted in vacDB, and from it, an export functionality generates the configuration files that are used by both the SCADA (WinCC-OA) and PLCs (Siemens S7/TIA). These files allow PLCs to communicate with device controllers, enable the communication between the SCADA and PLCs, and configure all SCADA functionalities such as automatically-generated graphical interfaces, archiving, alarms, and sharing of data with external systems.

In addition, the vacDB-Editor provides a functionality to import equipment and sectorization data from CERN's Layout database. All of the described functionalities (user interface, configuration exporter, Layout DB synchronizer, and data validation & persistence) are packaged into a single monolithic application, which runs on each user's desktop computer.

The Need for Upgrading the vacDB-Editor

Over the past years it has become increasingly difficult to maintain and upgrade the existing vacDB-Editor application. This is due to its obsolete technology stack, written in Java 6, using an old version of Oracle's ADF framework, whose development is dependent on a no longer supported IDE, JDeveloper 10g, released in 2007 [1]. Because of mandatory upgrades of the Java runtime environment at CERN, the vacDB-Editor, using older versions of Java, became more and more unstable, with frequent bugs and crashes reported by the users.

In late 2018 it was announced at CERN that a mandatory update had to be performed to upgrade Oracle databases [2], from version 11g to 18c, on all of CERN's production instances, affecting vacDB. Since Oracle 18c requires a more recent JDBC driver, not available in JDeveloper 10g, an unsupported migration of the vacDB-Editor was performed to support the new driver. While this migration was extremely difficult to perform, it appears to have been successful. It is however impossible to be sure that it will continue working as new Java runtime environments get deployed at CERN. Combining the technical reasons above with the scarce user base of ADF and JDeveloper, it was decided to rewrite the vacDB-Editor application using modern technologies, on a micro-service architecture, allowing us to be more resilient to technological advancements in the future. The new version of the vacDB-Editor is called vacCC, short for *Vacuum Controls Configurator.*

HIGH LEVEL ARCHITECTURE

vacCC is based on a microservice architecture, where each functionality is handled by an independently deployable application [3]. Although more complex to implement due to the increasing number of software parts, interactions, and underlying infrastructure, the usage of microservice architectures has been shown to bring important advantages over monolithic applications. We

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA123

consider the following advantages to be the most relevant for our case:

- Allows for individual upgrades of each microservice, as opposed to monolithic applications, where the entire application needs to be upgraded at once.
- Improved service resilience as the failure of a microservice will not affect the behaviour of other functionalities.
- Allows different programming languages to coexist in the same application, enabling us to choose the right tool for the job.
- Provides–a logical separation of concerns, making software better organized, and therefore easier for software developers to master, develop, test and maintain their code.

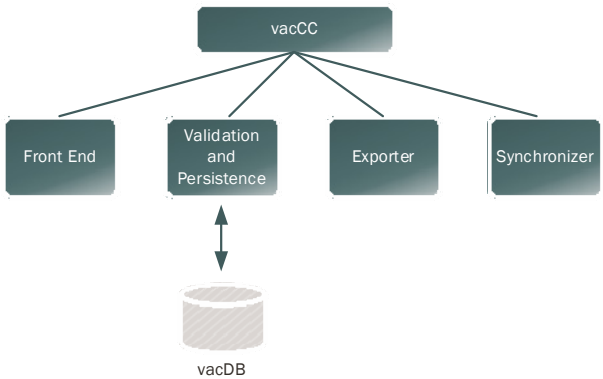

Figure 2: vacCC microservices.

According to Figure 2, the application logic was divided into 4 microservices:

- **Front end**: the web graphical interface that allows users to edit the configuration of the control system.
- **Validation & persistence**: exposes RESTful APIs [4] for other services to interact with vacDB, ensuring the validity of data used in CRUD operations.
- **Exporter**: generates SCADA and PLC configuration files from vacDB data.
- **Synchronizer**: synchronizes CERN's accelerator equipment database (Layout DB) with vacDB.

All vacCC microservices are containerized and orchestrated in Openshift [5], a Kubernetes [6] application platform.

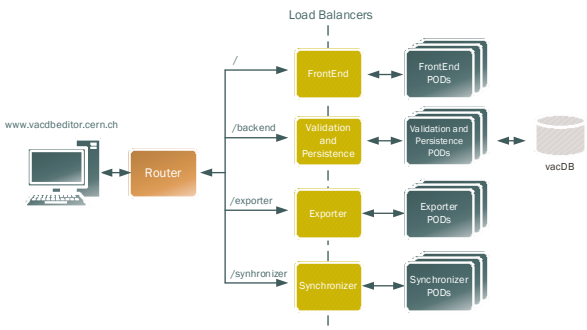

Figure 3: Deployment architecture.

Figure 3 illustrates the architecture of vacCC in production and development environments. An Openshift router exposes the application to CERN's General Purpose Network (GPN). Based on the request URL, the router will forward HTTPs requests to the appropriate load balancer, which will in turn forward the request to a microservice Pod [7] in a round-robin fashion. In addition to forwarding requests to Pods, load balancers provide a service discovery feature. Every time a new Pod of a certain microservice is created, the load balancer will automatically detect it and will start forwarding requests once it becomes available. This feature, combined with the stateless nature of all microservices, allows vacCC microservices to scale horizontally and to perform zero-downtime updates.

The following sections explain with a greater detail the implementation of the front end, validation & persistence, exporter, and synchronizer microservices.

FRONT END

The front end microservice provides the user interface of the application. It allows users to be abstracted from the complexity of vacDB, enabling them to modify vacuum machine parameters (e.g. equipment and their attributes, sectors, archiving, data sharing, etc.) that are required for the export of the SCADA and PLC configuration for the control system. The application is built as a single page application, implemented using the React Framework [8]. The graphical styling is provided by the Ant Design framework [9], which includes pre-made enterprise-class UI design components that can be added to the application with minimal configuration, enabling significant time-savings during development, while ensuring a consistent look and feel of the application.

Figure 4 illustrates the architecture of the front end application. The application is organized following the React model, where web elements such as pages and their elements (buttons, tables, forms, etc.) are hierarchically organized into components. Components interact with backend services (validation & persistence, exporter, and synchronizer microservices) using REST and WebSockets [10], the latter being used in special cases of long lasting requests with intermediate steps that need to provide feedback to the UI. Request data is stored in the application store, using Redux [11], that components can access directly.

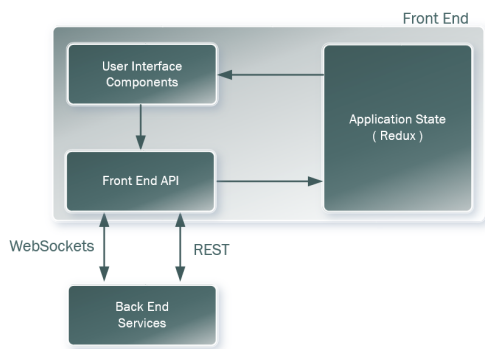

Figure 4: Architecture of front end application.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA123

VALIDATION & PERSISTENCE

The *validation & persistence* microservice is responsible for providing the interface between other microservices and vacDB. It achieves that by exposing RESTful APIs that allow other microservices to indirectly perform CRUD operations on the database. It is implemented with Spring Boot [12], a widely used framework for building web and enterprise applications, with an extensive user base, a rich set of features, with a convention-over-configuration approach that fosters simplicity and standard coding practises. Prior to any create, update, or delete operation on vacDB, data validation is performed to ensure that user intentions will result in a valid configuration, aiming to reduce the possibility of errors during subsequent vacDB SCADA and PLC exports. While vacDB itself provides a form of validation by the usage of constraints on the database level, this microservice makes use of attribute boundaries defined in MasterDB (the metadata source of vacDB) to improve the detection of user input errors.

Architecture

The architecture of the *validation & persistence* microservice is composed of 3 layers, as illustrated in Figure 5 below:

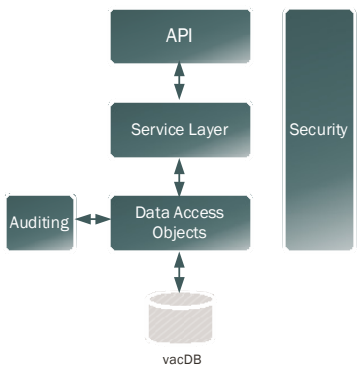

Figure 5: Architecture of *validation & persistence* mcroservice.

- **API**: exposes REST endpoints for other services to interact with vacDB.
- **Service Layer**: handles requests from the API layer, performs data validation, and when necessary, combines results from multiple Data Access Object (DAO) operations to serve API requests.
- **Data Access Objects**: provides objects that allow direct access to vacDB. This layer is implemented with Spring Data and Hibernate.
- **Auditing**: records modifications in the configuration of the vacuum control system.
- **Security**: provides authentication and authorization services for the entire application, ensuring that only users with the required priviledges are able to perform database operations.

Security

For user authentication and authorization, the Spring Security module is used and customized to obtain user authentication from CERN's active directory database. Upon a successful authentication, a JSON Web Token (JWT) [13] is issued to the client application. On every request, the client sends back the token, which is decoded, analysed, and matched against current authorization permissions in vacDB: this allows the microservice to confirm the identity of the client without the need of sharing credentials on every request. A great advantage of JWT is its stateless nature. With its usage, no sessions are maintained between the microservice and its clients, enabling any Pod to serve a particular request. Authorization for a particular operation (create, update, delete) on a specific entity (equipment, equipment attributes, sectors, etc.) is configured to use method-level security. DAO persistence methods are annotated with the role required to execute the method operation and matched against the requestor's granted roles.

Data Auditing

The *validation & persistence* microservice maintains audit logs for every change made in the configuration of the vacuum control system. Every time a configuration parameter changes, on any entity (e.g. equipment, equipment attributes, sectors, etc.), a log is added to an audit table containing a description of what changed (e.g. "timeout value changed from 10s to 30s"), when it was changed, and which user performed the change. Given that this functionality was common to all DAO objects, it was implemented using Spring's AOP (Aspect Oriented Programming) paradigm [14]. Each create, update, or delete operation in any DAO object is captured by an AOP advice (which is essentially a trigger function) that compares the current version of the object to be modified to its future version, and stores a change log result in the audit table. AOP allows the DAO classes to be completely unaware of the data auditing mechanism.

vacDB

In order to ensure that the configuration of the vacuum control system is always possible during CERN's Long Shutdown 2, where the configuration of the control system is changing on a daily basis, one of the project requirements was to make vacCC compatible with the database used by the vacDB-Editor. This provides users with the possibility of using the vacDB-Editor in case of problems in the new application, especially important during the validation stage of the application. The easiest approach for this problem was to make both applications share vacDB, avoiding the need of creating custom software to keep vacDB and a new database consistent.

As illustrated in Figure 6, vacDB has 2 types of databases.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA123

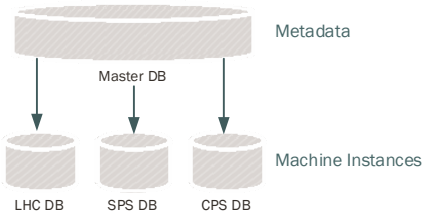

Figure 6: High level overview of database interactions.

Master Database The Master DB is a metadata database that defines information common to all accelerators. Examples of metadata found in Master DB are the attributes that define each control equipment type, default values for attributes, datapoint definitions, and the types of data for each datapoint element (uint, float, etc.). A single instance of Master DB is shared amongst all Machine databases.

Machine Databases Machine databases contain data concerning a particular vacuum installation (LHC, SPS and ComplexPS). These databases share definitions from the Master DB in read-only mode. In Machine databases, we find definitions of vacuum sectors, equipment and their attributes, archiving, alarm settings, and all other data needed to generate all of the export files required for the configuration of the vacuum control system.

Control System Configuration Versioning Machine databases contain several versions of data for a given accelerator and, as illustrated in Figure 7, all database entities point to a version. This feature is necessary to allow future versions of the control system to be edited without affecting the current production versions, and also to maintain history of previous configurations of the control system.

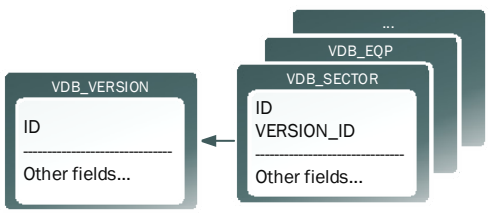

Figure 7: Versioning in Machines databases.

When a new database version is required, it is usually copied from an existing version – typically from the production version. A new database version ID entry is created on the versions' table and all database entities that point to the source version ID are duplicated with the new version ID field.

EXPORTER

The *Exporter* microservice is responsible for generating the configuration files for both the PLCs and for the SCADA.

For each PLC, the exporter generates function block calls for each vacuum device connected to it, along with device datablocks; these contain all relevant information that will allow PLCs to connect and interact with device controllers. Two special datablocks are generated to enable bilateral communication with the SCADA: the Read Register, for the SCADA to read from the PLC, and the Write Register, for the SCADA to send data – commands or configuration - to the PLC. PLC functions copy data from device datablocks to the Read and Write registers on specific memory positions specified in the PLC configuration files.

For the SCADA, the exporter microservice generates configuration files with the data that will allow the configuration of all datapoints for every vacuum device. Each datapoint will be configured with the archiving settings defined in vacDB, and each datapoint element that requires communication with a PLC will be automatically configured to point to its corresponding memory location, within the appropriate PLC Read or Write register. In addition to the configuration required to enable the communication between SCADA and PLCs described in the previous paragraphs, other files are generated to configure the display of equipment in the SCADA synoptics, alarms, long term archiving, and data sharing through middleware protocols with other control systems.

SYNCHRONIZER

The Layout DB is a CERN-wide database that models the architecture of CERN's accelerators. It contains data concerning most accelerator subsystems, including RF, beam instrumentation, magnets, cyogenics, and vacuum. The purpose of the synchronizer microservice is to automatically import vacuum data from the Layout DB into vacDB, ensuring that the official, approved layout of the vacuum system is reflected in vacDB. Fig. 8 shows the data-flow between the Layout DB, the synchronizer microservice, and vacDB.

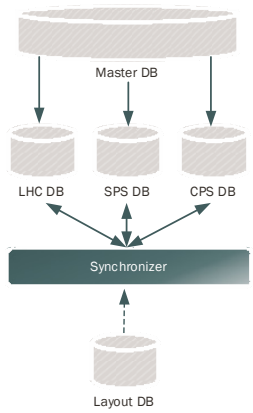

Figure 8: VacDB relation with Layout DB.

Through vacCC's user interface, users can trigger a differential analysis between vacDB and the Layout DB. The differences detected in the analysis are based on the create, update, and delete operations made on the Layout DB that are not reflected in the Machine DB, concerning vacuum sectorizations, and equipment and their attributes (position,

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA123

type, and hierarchy). The analysis process provides users with a list of actions that need to be performed on vacDB to bring it up to date with the Layout DB. Users can use the synchronizer service to automatically perform the suggested updates.

CI/CD

A continuous integration and continuous delivery (CI/CD) philosophy is used in all vacCC microservices. With continuous integration, developers push their code daily, if not multiple times per day, into the master repository of each microservice. Code is automatically compiled, and unit, integration, and linting tests are performed at each commit. This methodology allows compilation errors and non-conformities to be detected early, thus minimizing the risk of faulty code reaching production and staging environments. Continuous delivery, on the other hand, is the process of automating code deployment. By eliminating human intervention in the deployment, we can guarantee that code is always released in a standard manner, thus reducing risk and minimizing deployment times.

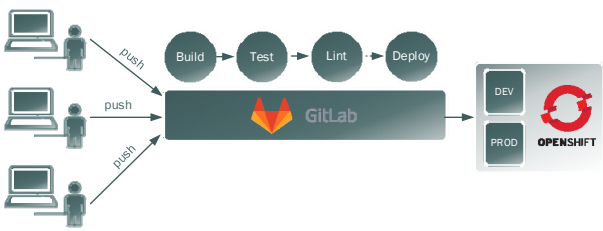

Figure 9: Continuous integration / continuous Delivery.

As Figure 9 illustrates, every change made to the code repository passes through a pipeline that will build (compilation and creation of Docker images [15]), test, and lint the code. In case of errors, the pipeline will stop and the developer will be alerted. Commits pushed to the master branch of the repository that pass the build, test, and lint stages are automatically deployed to the staging environment, a replica of production, where developers can perform additional testing. After validation in the staging environment, a tag of the master branch is created, and developers can trigger an automatic deployment to production.

CONCLUSION

The *front end* and the *validation & persistence* microservices of vacCC are in production since March 2019 and have completely replaced the vacDB-Editor as the tool for editing the configuration of the control system. Users have reported a significant increase in productivity using the new interface, which is especially important during the Long Shutdown 2 of the LHC, when tens of thousands of configuration changes are expected.

We are currently in the validation phase of the exporter microservice and expect to complete the development stage of the synchronizer miroservice on late 2019, when the vacDB-Editor will be completely replaced by vacCC.

The adoption of a microservices architecture in vacCC brought several advantages. It allowed to split a big problem into smaller, independent, and more easily manageable pieces of software, enabling developers to work simultaneously on the different system components. Future upgrades of vacCC to new technologies can now be carried on a service by service basis, without the need of a big team of software developers uniquely dedicated to upgrading the whole application at once.

The usage of Openshift/Kubernetes to manage our application containers made our applications self-healing in case of hardware problems, brought zero-downtime deployments, and allows for dynamic horizontal scaling to ensure a consistent performance of the application.

The development of continuous integration and delivery pipelines, with integration testing and automatic deployment to staging and production environments, allowed developers to be more confident on the changes they make. This is now the standard for all new applications developed for vacuum controls at CERN.

In summary, we expect these architectural and technology choices to result in more agility to react to new user requirements and technological changes, making software in vacuum controls ready to face the upcoming years.

REFERENCES

[1] Oracle JDeveloper, https://www.oracle.com/database/technologies/developer-tools/jdeveloper.html

[2] Oracle DB 18, https://docs.oracle.com/en/database/oracle/oracle-database/18

[3] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis and S. Tilkov, "Microservices: The Journey So Far and Challenges Ahead" *in IEEE Software*, vol. 35, no. 3, pp. 24-35, May/June 2018.

[4] REST, Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, USA, 2000.

[5] Openshift, https://www.openshift.com/.

[6] Kubernetes, https://kubernetes.io/.

[7] Pod, https://kubernetes.io/docs/concepts/workloads/pods/pod/.

[8] React Framework, https://reactjs.org/.

[9] Ant Design Framework, https://ant.design/.

[10] WebSockets. [RFC6455] I. Fette, A. Melnikov, "The Websocket Protocol", December 2011.

[11] Redux, https://redux.js.org/.

[12] Spring Boot, https://spring.io/projects/spring-boot

[13] JSON Web Token, https://jwt.io/.

[14] Spring AOP, https://docs.spring.io/spring/docs

[15] Docker, https://www.docker.com/.