BUILD SYSTEM FOR FDA-REGULATED SOFTWARE DEVICES

Summary

The Customer was building an FDA regulated medical device provided as a "Software as a Service" (SaaS) solution. The Customer had created a solution based on various scripting technologies which required a large amount of manual system configuration, which was difficult to replicate reliably on different developer systems. There was no automated build server validating the source code and running test suites.

Some standardization had begun by using heavyweight virtualization technologies (like Virtualbox) which helped, but the overall system cost (in terms of CPU and memory usage) negatively impacted developers, and the solution was expensive to maintain. The technology and methodology being used also limited the number of developers that could work on the project which further limited the scope of the project. Additionally, the versions of system libraries were rarely, if ever, constrained or tracked in the build system, making it difficult to demonstrate traceability of the provenance of the generated executables which would be a show-stopper for the FDA regulatory process.

Name: US-based Fortune 500 company

Industry/Sector: Pharmaceuticals

Project Type: Regulated software medical device

Technology Used: Docker, Atlassian Bamboo, Linux

Project Requirements

For FP Complete to make this project a success, the project needed to be functional, scalable and accountable in order to meet FDA standards. To meet these objectives, the project requirements had to fulfill the following parameters:

Reliably compile and run the software in development

- Ensure reproducibility of the build process based on an auditable, recordable, and tightly maintained list of components used.

Run unit and integration test suites in a predictable environment

- Automate the builds and test suites on a central build server using standard Continuous Integration (CI) practices
- Design and implement a method for deploying the built device for testing and production

Deploy the device software to a cloud server for reliable hosting to customers

Provide audit information for FDA filings

- FDA Regulated Devices require strong auditability guarantees- each version of the software for the device must have clearly traceable source code, build tools, system library, and configuration steps.

Developers can easily and reliably perform builds

Enough flexibility is available to let developers make modifications, such as enabling profiling, disabling optimizations, or enabling advanced debugging capabilities

The Solution

FP Complete used lightweight containerization, which leverages some aspects of the host OS (like the kernel), which allowed for faster initialization, lower memory and CPU usage, simplified disk management, and easier upgrades.

This allowed FP Complete to ship a complete filesystem for a working Linux operating system to each developer, build server, and production machine which neatly codified all system library versions, build tool versions, and command line utilities creating several benefits:

Minimized dependencies on the host build system. This meant that very little of the host build system affected behavior within the container, therefore limiting the impact on the resulting executable, making for a strong reproducible build story.

Simplified spinning up of additional build servers, making it easy and cheap to speed up the Continuous Integration (CI) process (often a bottleneck on large teams).

By minimizing host system dependencies to a few well supported tools, and having all other tools provided by the container, an easy path was created for both onboarding new engineers to work on the project, as well as upgrading dependencies across an entire team.

Support for development from any operating system and machine which could run the containerization system, making it easier for developers to contribute from their choice of host OS and hardware.

New Challenges for FP Complete

Unfortunately, Continuous Integration systems provide much weaker tooling around tracking build plans themselves. As part of the effort to have reproducible builds and also meet FDA auditability requirements, the build scripts were moved into the source control repository for the device software. These scripts included records of the container version and build tool versions required for completing the build successfully.

The result of this was that each revision of the source code fully identified all tooling dependencies and build plans needed to produce it. Finally, the build process was configured to inject extra metadata about the source code revision into the generated device software which enhanced the auditability of the code.

The upshot of all of this was:

- Every revision of the software device source code contained a fully specified recipe for building the software reproducibly.
- The Continuous Integration system contained only the most minimum logic necessary to call into the build system contained in the source code repository.
- For any deployed or tested version of the device, it could be determined which version of the source code generated it, allowing the Customer to rerun that build at any point in the future.
- By ensuring matching environments with CI, test, and production, the frustrations of trying to fix a bug that doesn't reproduce locally were reduced.
- Containerization provided developers with a simple, straightforward set of instructions to get up and running quickly, avoiding costly setup times and delays in productivity.
- The build system was able to follow all of the same code review and change request procedures required on this project, simplifying the development process and removing yet another hurdle to obtaining FDA approval.

Conclusion

Since introducing the build system, the Customer's software team has been able to grow and has gone from around half a dozen developers to nearly three dozen developers, QA engineers, and system administrators working on the code base and reliably building on a daily basis. The continuous integration solution regularly builds the device software, and provides the foundation on which the pull request and code review process ensures that the master branch of the software remains stable, decreasing integration friction amongst the team immensely.

Leveraging add-on technologies like Docker Compose, we have set up a simple addition to the build process which allows engineers to launch fully configured devices, including database servers, message queues, and the device software itself. This has allowed the development and QA teams to have a simple, reproducible environment for reporting and fixing bugs. FDA compliance has been assured.

FP Complete continues to maintain this build system, adding features as desired. The structure of the solution makes it easy to extend the system, test within our Continuous Integration environment, and deploy to all engineers with a simple push to the master branch.