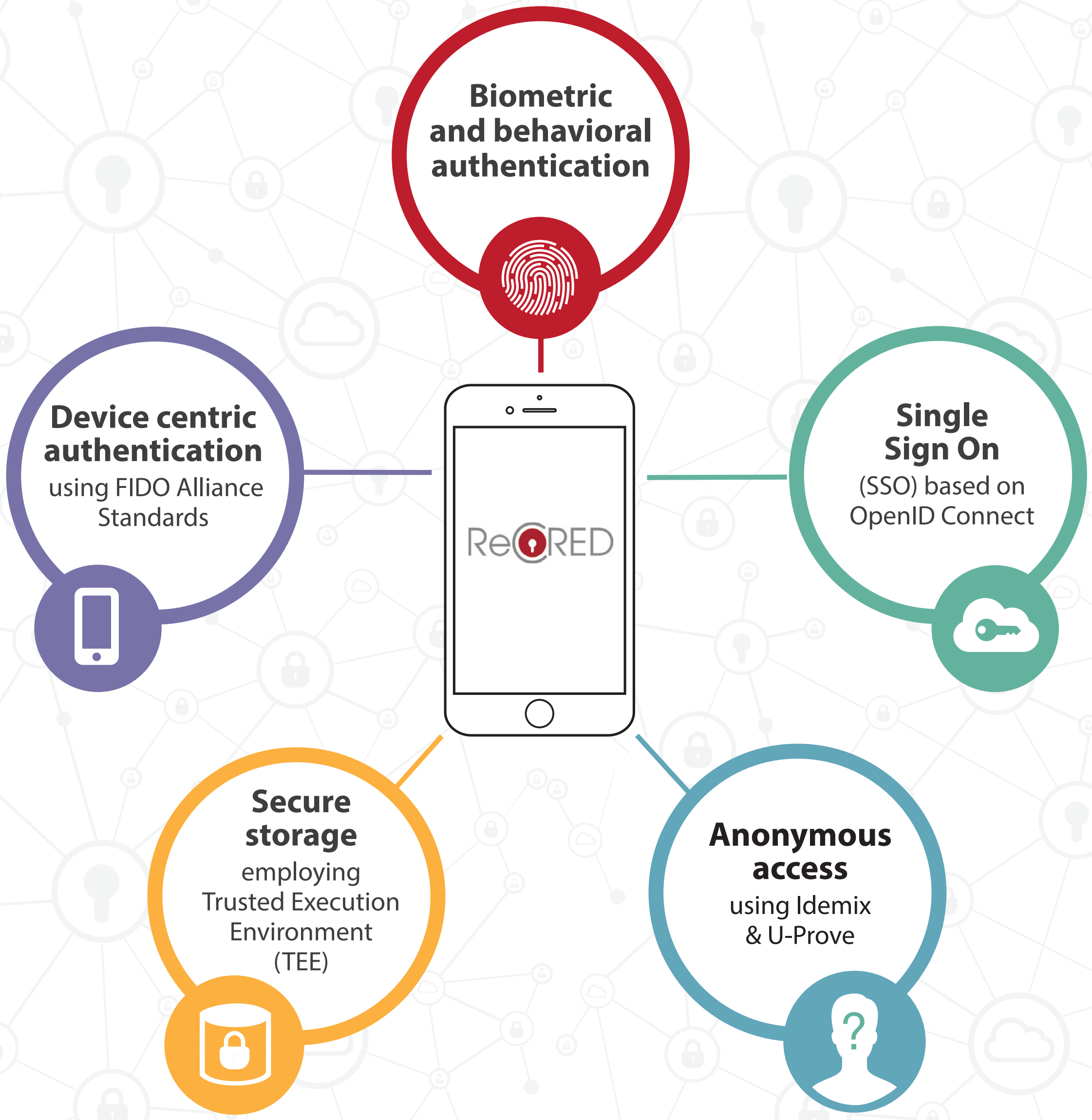
ReCRED

From Real-world Identities to Privacy-preserving and Attribute-based CREDentials for Device-centric Access Control

Makes your digital life **safe** and definitely **easy**!



Biometric and behavioral authentication
Device centric authentication using FIDO Alliance Standards
ReCRED
Single Sign On (SSO) based on OpenID Connect
Secure storage employing Trusted Execution Environment (TEE)
Anonymous access using Idemix & U-Prove

www.recred.eu

European Commission | Horizon 2020 European Union funding for Research & Innovation

The complex integration problem

As computer science advancements solves even more daunting tasks that modern society is facing, an increase of implementation complexity is imminent. Given that commercial or research projects have reached a level of complexity that can only be solved with expertise that is spread across multiple scientific fields that require years of practice and implementation and is held only by various universities and corporations that are located across the globe, a new type of problem has emerged. It concerns with integrating software components that are part of a complex system and are developed in the same period of time by groups of engineers that are geographically distributed across the globe and not following public standards (like RFCs, IEEE standards, etc.). In addition, if component development is incremental and continues after the first integration by adding new functionality to the software stack, it must not result in breaking compatibility of the components that are already integrated. From this perspective, the importance of a correct implementation for continuous integration concepts keeps getting bigger.

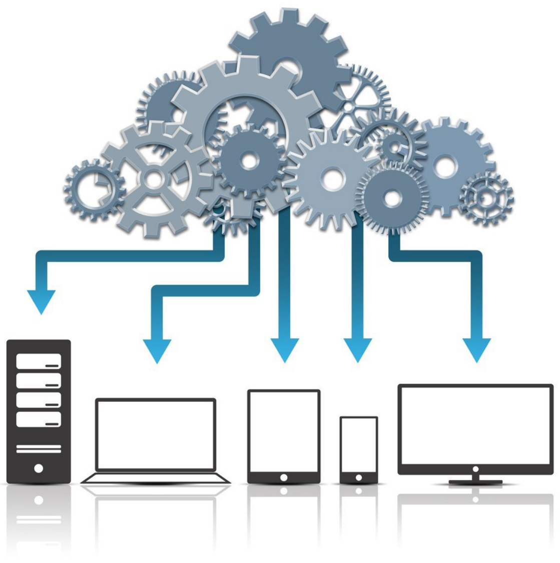
Continuous Integration

Facing the same problems in ReCRED, we implemented Continuous Integration (CI) principles using the latest technologies available at the start of our development process. Using GIT as Source Control Management (SCM), Jenkins as CI platform and Docker Containers as virtualized deployment environments, we managed to transform source code developed by different organizations in the same time frame into fully integrated and tested complex systems ready and available for deployment at any location.

To solve integration issues, we decided to take advantage of the continuous integration process and the functionalities offered by the CI tools.

Adopting Continuous Integration as a mechanism of software development requires spending some effort at the beginning, but for any project that is longer than a few weeks it is proven to pay off regarding the extra time spent on the other tasks than development tasks itself. Not using CI requires some specific milestones in project life, where the development process should stop and the entire team concentrate on integrating all the software modules and solve defects that are detected in the integration process. This moment of time can repeat periodically, or can happen at the end of the project, but the point is that the amount of software written can get so big, that the integration becomes uncontrollable and unpredictable, with so many examples of taking months to integrate without knowing when it will be finished. Adopting and using CI transforms the integration process into a continuous one that will treat integration as a non-event. The CI is not a process that concerns a part of the team or a specialized "integration" team, it is a process that must be adopted by all the software development teams in ReCRED and that must follow some general guidelines which result in a continuous stable state of the software. Although it is not necessary to use a set of specific tools, the software engineering industry evolved so much, that a lot of tools can be found that automate the process, saving a lot of time; for that we decided to use a dedicated server that runs Jenkins as the main CI tool and represents what we call the Integration Platform.

What this platform is responsible for, is to monitor the software repositories and for every commit it will checkout the new software version, build it, link the modules together, run a configured set of tests and finally perform a deployment in the virtualized environment. Therefore, the CI contains a high number of unit tests that are part of the software, which makes them automated into the software; in Extreme Programming (XP), this concept is known as self-testing code.



After writing the code and perform the commit in the source-code repository, the developer work is not considered done until all the tests are passed. If one of the tests fail, it will result in a failed build and it is the responsibility of the developer that performed the commit to solve the defect. It is not an exception to get clashes in the code written by two developers, but in this case, the clashes must be repaired immediately.

As part of the normal development process, at the beginning of the day, the developer must update its local working copy of the source code and start a build. In case the build fails, then someone else did not do his job and left the code in an unstable state in the previous day. After the build, the developer starts working on one single issue (following the branching model in ReCRED) and when the specific task (or subtask) is implemented, he must build again. After the build succeeds, he must pull the repository again, integrate the changes that other developer performed and build again. Only after this build succeeds, he can commit to the repository (in case the task is developed on a separate branch, the commit refers in fact to the merging into the master branch). Although it is a good idea to keep the number of branches at minimum, in ReCRED we usually follow the GIT branching model that we also explained in this document. Further, the developer must wait for the CI platform to perform the "commit" build and only after this build succeeds, the job can be considered done.

In case the CI build process fails, it is the highest priority of the development team to solve the issue and not leave the software into an unstable state "over the night". It is very important for every single developer to know where the source code resides and for that, in ReCRED, we keep a close tracking of every software module and the repository URL that contains it. What lies in the repository is not only the source code, there are other modules, including the unit tests, all the automated tests, the test scripts, the configuration files, the database schema, the installation scripts, third party libraries. The main rule is that a minimum amount of effort should be spend by someone checking-out the source code on an unconfigured machine and starting the build, up to the point where the software modules run on that machine. The compiler and the language tools (e.g. python, java, gcc) are considered prerequisites and will not reside in the repository.

Having the software build on a machine by using the IDE configuration is not enough. There are many IDEs that have their own build mechanism and cannot be considered as having a build system in-place. Instead, every software module must contain console build scripts that will be executed in an automated environment without the IDE being available. What we also don't keep in the source repository is the result of the builds that represents the executable (which can always be recreated from the source code).

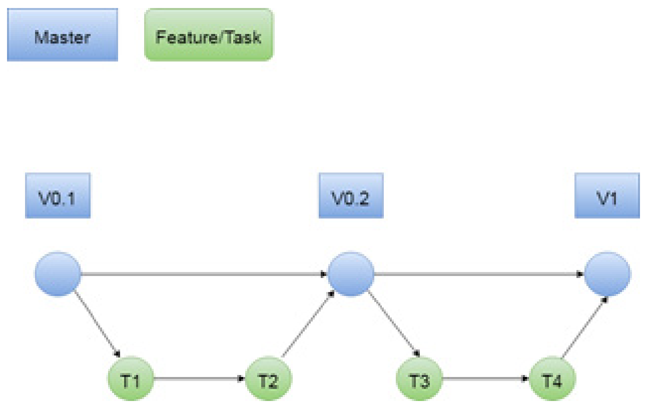
It is essentially to fix the build very quick; as a rule, the current task is not considered closed until the CI build passes all the automated tests. That, combined with the very frequent commits in the repository has as a result that a defect is identified and fixed in a very limited code area, which can even be solved through what is called diff-debugging.

As opposed, if the defect is not fixed immediately, it will reside in the source code for days or weeks. Even more, the defects might accumulate and the software module and further the entire project will end up with having so many bugs that will make their fixation a very difficult process. More than that, depending on the functionality where the bug appeared, it might be even impossible to continue development of another module that depends on the broken one.

Another important aspect of the build processes that run on the Integration Platform is the time spent by the system in one build. Taking into consideration that the developer must wait for CI build to finish, if this time is measurable in hours, then the developer will spend a lot of time waiting for that result. That, multiplied by the number of builds that can happen in a day and multiplied by the number of hours that a build takes to finish, can result in many hours per day wasted by the developers waiting for the builds to finish just for implementing two or three tasks. A reasonable amount of time accepted by the industry is less than 10 minutes. Although initially, when the CI platform is at the beginning of its setup, it is acceptable to spend 1 or 2 hours in a build, it is mandatory to lower this time, before the number of builds per day will increase. That might lead to creating different stages of build, a very fast one that we can call "commit" build stage and a slower one for which the developer doesn't have to wait to finish. In ReCRED, the platform already supports pipeline builds, in which a successful build will trigger a new build configuration and so the separation can be done. For now, the pipeline build is used to control the entire build process and to add dependencies of the software modules.

If we consider the production environment to be the pilot environment, it is also important for the integration process to have a build environment very similar to the production one, mostly because the executable artefact is linked against environment libraries and because it is the build system that also runs the automated tests. Having virtualized environments in ReCRED, we ensured the similarity of the two environments; that also saves all the elements necessary for a build artefact to run, so a new environment does not have to be recreated.

All the build artefacts are copied on the local machine in a special folder that is made available through HTTP directory listing to all the members in the project. As a result, at any moment, anyone can take the last software version and can test or install it in a different environment, with the purpose of smoke-testing or use it in production. As the build platform has a web interface, not only the artefacts are published, but the entire build process is visible, including the result. As a result, anyone can see the status of the builds (fail/success), the module that failed and even the entire build process; this can ease the process of fixing a defect, as the developer responsible for the new defect will also be informed, even if the build was triggered as a result of the commit of a different developer, eventually in a different software module.



As important as the automated build process is also the deployment process. As the deployment is happening just as often as the builds, executing the same commands over and over again will result, after a few days in a waste of time. By using Docker in ReCRED, we take advantage of the build-in commands to automate the deployment and even to version the containers as well.

For that, we chose to use the multi-branch pipeline approach in Jenkins which permitted us to create builds for each branch of each software component. More than that, the entire build and deploy processes is customizable through specific configuration using Jenkins files. Using build configuration in a file that is also committed alongside the source tree has a very important advantage, that of versioning the file.

There is no doubt that using a source versioning control (SVC) system is a must for a long time already, in software development; the versioning systems offer more than just a history of the source code, they offer the possibility to fork the software at some point and continue the development of a specific feature without affecting the features already developed. In ReCRED we chose to use GIT as SVC because it has a better branching mechanism and because of the loose connection with the central repository.

In ReCRED, every project has a main branch named “master” and for each feature that is developed, the master branch is forked and a new branch is created. All the development and the commits for that specific feature are applied on the new branch, keeping the master branch untouched by the daily commits. When the feature implementation is finalized, the feature code is merged into the master branch.

In software development, a better approach is to fork not from the master branch, but to have a new branch called “development” and to fork every new feature from that branch.

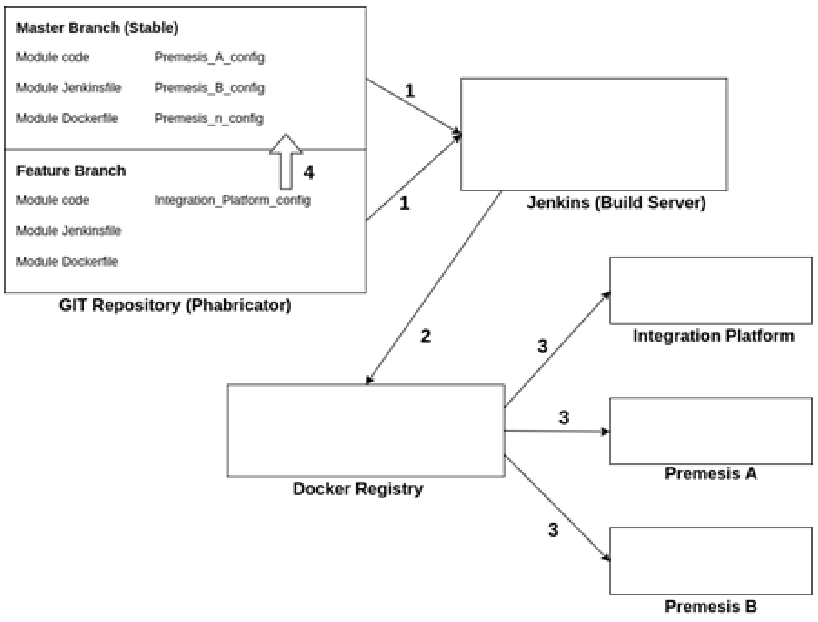
There are no strict rules that are followed in ReCRED regarding the branching model, as the software development does not target a production environment, but only prototypes and pilots. We did stick to that of having new features on different branches, to get the CI system to build each branch and deploy multiple instances of the same software module to ensure the integrated system is running all the time.

Conclusion

To summarize, the base rules of CI that we follow in ReCRED are:

- Keep all the source code in GIT repositories;
- Automate the build process and the creation of the executables;
- Every developer must commit his code after every new feature;
- Every commit must end with a successful build that also includes the successful passing of the automated tests;
- Every build with which brings integration issues or bugs must be fixed before continuing with future developments;
- Build duration should not be longer than an acceptable threshold;
- Deploy the build in an environment that is very similar to the production environment
- It is mandatory that the build result is known and available to everyone;
- The deployment process needs to be automated.

The following diagram helps to form a proper view of how the source code travels from the GIT repository through the integration system and ends deployed on a production environment:



1. Jenkins detects new feature branches, or merges to the stable (master) branch;

2. Given the Jenkins file which defines the build logic, module source code, configuration files for every deployment location and the Dockerfile, Jenkins can build a preconfigured image and push it to the Docker repository with appropriate version tags for stable or developmental iterations;

3. Preconfigured images are available for deployment on production location, or at the integration platform to test that the new feature does not break the system functionality as a whole;

4. After the new feature has passed the tests on the integration platform, its code is merged on the stable branch and the process repeats from step 1.

ReCRED

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ UNIVERSITY OF PIRAEUS

EXUS

institute IMdea networks

cnit

Cyprus University of Technology

Telefonica Telefónica I+D

THE PRODUCTIZERS

certSIGN

wedia

verizon

upcom Bringing business and IT together

Baker McKenzie.