The following requirements are based on several assumptions. The first is that each developer has their own complete dev environment, including a local Git repository, running on their dev computer. The second is that a central master Git repository is used for Continuous Integration of the source code stored in the local Git repositories on each development workstation.

The prerequisite for Continuous Integration (CI) requires trunk-based development. CI means that developers synch their local Git repositories to the main Git repository at least once per day. The prerequisite for Continuous Deployment (CD) is automated testing of all of the logic in a system (both positive and negative tests). For DGP, CD means building and deploying to the test environment at least once per day. Automated end-to-end full regression tests of all API methods in a system are performed each time a build is deployed to the test and QA environments (not production). Running the automated tests in the production environment would combine useless test data with production data, and is therefore to be avoided. The QA environment is a segment of the production environment with its own separate data storage to avoid this problem.

Refer to: https://trunkbaseddevelopment.com/

Requirements

1. Trunk-based development

What: *Each developer has their own local Git repository when using Visual Studio. Code that does not yet build successfully is not pushed up to the central Git repository until it is ready, while the developer continues to pull changes made by other developers down to their local Git repository. This process eliminates the need for short-term feature branches.*

Why: *Branches can cause problems that are referred to as "merge hell". The individual local Git repositories on each developer workstation provide the same net result as feature branches in the code, so branches in the various Git repository are not needed. This assumes that a single developer works on a feature, which is a valid assumption given the fact that all functionality is implemented as immutable API methods, and the small number of developers on each team.*

Testing: *The build, deployment and automated testing in the test environment proves the CI-CD-CT process.*

2. Automated testing

What: *Automated end-to-end full regression testing is required to support continuous deployments to the test and QA environments. In this case, automation refers to the ability to run full regression tests of all of the functionality in a system using a tool or application – it does not need to be scripted or scheduled for execution to meet this definition.*

Why: *It is simply not feasible for manual testing to keep up with a cadence of daily builds, deployments to the test and QA environments, and full end-to-end regression tests of all API methods in a system. Automated full regression tests are the only viable option.*

Testing: *The build, deployment and automated testing in the test environment proves the CI-CD-CT process.*

3. Fast, easy deployments

What: *Manual processes to deploy each tier of code must be fast and easy, and well documented. Once they have been proven, they can then be scripted to automate the deployment and testing process.*

Why: *Starting with manual processes for deployment and then scripting them as needed allows the manual processes to be used as a backup mechanism if the automated scripted versions encounter any problems.*

Testing: *The manual and scripted deployments can be tested in the test and QA environments.*

4. Fast, easy rollbacks

What: *Manual processes to rollback new deployments must be fast and easy, and well documented. Once they have been proven, they can be scripted to automate the process.*

Why: *The rollback mechanisms are most important for production environments to minimize the negative effects of problems that only occur in production. If any such problems are found in the production environment, the latest deployment must be rolled back to the previous working version as quickly as possible.*

Testing: *Rollbacks are obviously most important in production environments, but the rollback process is practiced and verified in the test and QA environments.*

5. The overall distributed system must tolerate inconsistent versions of code and data during deployments

What: *During deployments of new releases, the system will be in an inconsistent state between locations, and also between servers in a location (web server farms, database clusters, etc.).*

Why: *Deployments of code and synchronized data are done incrementally, one server at a time or one location at a time, while the overall system remains in continuous use. The system must provide a combination of mechanisms such as feature toggles that restrict access to new functionality and new data until the deployments are complete for all parts of the distributed system and it is once again consistent.*

Testing: *Full regression API tests can be run while servers and data are inconsistent in the lower environments.*

6. Easy synchronization of security data across environments and locations (including production)

What: *The data-driven RBAC security built into each web service cannot function without a base set of security data (methods, roles, admin users, role membership, etc.). This data must also be synchronized between all of the dev environments, as well as the test, QA and prod environments.*

Why: *Scripts or tools must be used to maintain a master set of API methods, roles, admin users, role membership, and group membership data so that the base security data allows admin access to a new location. However, manual entry of the same identical data in each environment provides an alternative with the best overall security.*

Testing: *The security admin tools can be used to verify security data is synchronized between environments, and also fix the security data if it ever gets out of synch.*