Continuous Integration, Continuous Deployment, Continuous Testing

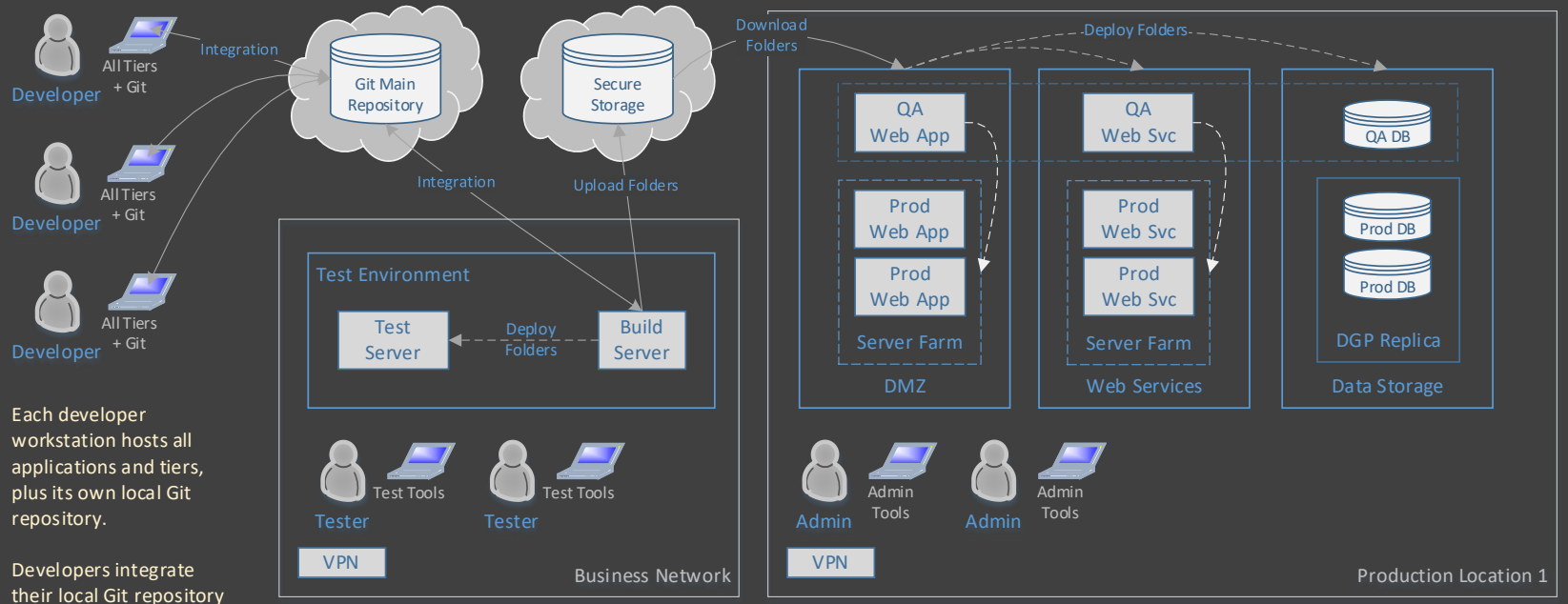
Continuous Integration (CI), Continuous Deployment (CD), and Continuous Testing (CT) are a combination of techniques that are meant to speed up the delivery of new features, functionality, and capabilities to the users of software systems. System evolution produces a constant stream of extensions and improvements for a successful software system, which must then be integrated, deployed and tested without ever breaking backward compatibility or causing any system downtime.

Prerequisites for CI/CD/CT:

- Automated testing. In practice it is virtually impossible for manual testing to keep pace with daily deployments to the Test environment, especially for larger systems. The ability to quickly and easily run full regression tests of all of the functionality in a system, per environment, is required. This capability is provided by DGP's API Tester test harness application.
- Trunk-based development. Creating branches defeats the purpose of continuous integration, which is to force any integration problems to be dealt with immediately (daily), as they occur and while they are still small enough to be easily fixed. Integration problems can remain hidden within branches until they belatedly are attempted to be merged back into the main trunk. From experience at some businesses, such attempts to merge branches can eventually become impossible.
- A Feature toggle mechanism. Not all features and functionality will always be able to work the same way in all environments, and also sometimes need to be deployed before they are completely finished or have been adequately tested. A simple toggle mechanism to easily turn features on and off in different environments is required. DGP's data-driven Role Based Access Control (RBAC) security system provides this functionality.

The diagram below shows a typical infrastructure with the various network segments and security measures incorporated into the implementation of a CI-CD-CT process. The process was created and verified using manual steps, after which some of the manual steps were then automated to the degree allowed by Infosec. For example, no push from the secure storage into the production environment was ever allowed – only a pull performed by admins of the production environment was permitted, for security reasons.

Continuous Integration, Continuous Deployment, Continuous Testing : Environments



Each developer workstation hosts all applications and tiers, plus its own local Git repository.

Developers integrate their local Git repository to the central master Git repository multiple times per day (trunk based development).

By default, development environments are VM's with IIS, SQL Server and Visual Studio, standardized for consistency plus easy recovery from problems that may occur. Developers can test and debug all parts of the system locally.

The Test environment can be on premise or in the cloud, and can have all tiers installed on a single computer (like developers) or have separate machines per tier (like QA).

For .NET applications, the "build server" is typically a development computer that uses Visual Studio to create the builds (folders), and can also be used to integrate changes made in the test environment back to the main Git repository (useful for bug fixes, changes to documentation and test files, etc.).

Full regression tests of all API methods in a system are run in Test. If successful, the build folder is uploaded to the secure staging storage.

Once the testing in the Test environment is successful, the new executables (folders) are saved to secure cloud storage, which is accessible to both testers and the system admins for the production locations. This is the mechanism that allows new folders to be downloaded into production locations. Folders cannot be pushed to QA/Prod environments, only pulled down by admins for security reasons.

The QA environment is a section of the production environment and infrastructure that has its own separate web services and database (or separate servers). This is the environment that allows full regression tests to be run in "production" without mixing test data with prod data. Developers can be given temporary access to QA to help debug problems in the production locations.

Full regression tests are run in QA. If the tests are successful, then the release folders can be deployed to Production.

Continuous Integration

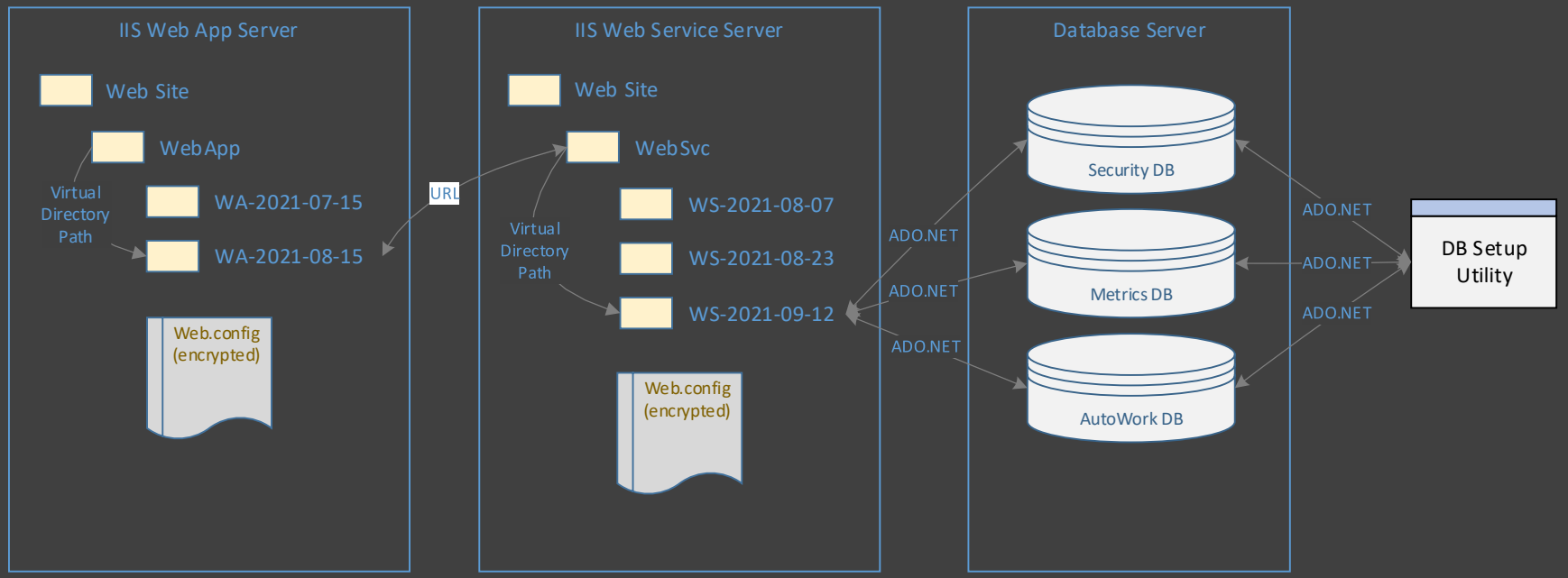
The purpose of continuous integration is to force the code created by multiple developers to be integrated on a frequent (daily) basis so that any integration problems are addressed while they are still small enough to be fixed relatively easily. The technique of trunk-based development is used to simplify the continuous integration process. Branches interfere with this merging/integration process, and so alternatives that provide the same benefits as branches are used instead.

For DGP systems, each developer workstation is a self-contained dev environment which has all dev tools (Visual Studio, IIS, SQL Server, etc.) and its own local Git repository (Git for Windows, managed by Visual Studio). Each developer workstation is also connected to the main Git repository through Team Explorer in Visual Studio. All development occurs in the main branch (trunk) of the local Git repository. This represents an independent and autonomous dev environment for each developer, synchronized with each other through the daily integration of the local Git repository with the main Git repository.

Branches are replaced with a different mechanism that produces essentially the same results while still using trunk-based development. A developer working on a new feature only pulls changes from the main Git repository down to their local Git repository, but does not push their work up to the main repository. The new code can be pushed up to the main repository as soon as it builds successfully in the local dev environment thanks to DGP's feature toggle (security system authorization) mechanism, which is able to "turn off" access to that functionality until it is complete and ready for testing.

Continuous Deployment

First, DGP follows an immutable append-only convention for many subsystems, and that includes deployments/rollbacks. DGP deployments are based on .NET folders (XCOPY deployments). XCOPY deployments have no installers and make no changes to the configuration of the host computer. Deployments simply copy a folder under a parent directory on each computer. Web apps and web services are deployed as folders in a similar way, with the added step of editing the physical path of the IIS web app virtual directory to point to the new version folder. Each version folder remains separate and distinct from one another, with no changes or merging once they have been deployed (refer to the diagram below). Rollback of a deployment simply requires that the path of the virtual directory be pointed back to the folder of the previous version.



1. Data

In DGP software development, each environment (Dev, Test, QA, Prod) will have its own separate databases for security, metrics, automated work, etc. In DGP, security database schemas and the security data follow an immutable append only convention. The DGP DBSetup utility application applies all additions to schemas using idempotent logic, and the security data is optionally synchronized using the omnidirectional replication logic. New fields added to existing tables must be nullable. Default values for new fields can be assigned incrementally to existing records using automated processes whenever that proves to be necessary.

Some of the global security data, such as APIs, API methods, etc. need to be synchronized not just between the locations of a given environment, but also between multiple environments. The DBSetup utility is one way to synchronize security data between different locations and different environments of a system by reusing the replication process logic to simulate that data being replicated into each destination from a “virtual” master database that only exists in the code. Another way is to use automated data replication between environments. The final way is to manually recreated core data in each environment (which provides the best security).

2. Web applications and web service APIs

In the .NET Framework, web applications that are published are compiled to assemblies of MSIL. All resources that are needed to run the application are collected into a folder in the file system. This folder is then deployed to the target computer as a simple copy and paste operation. None of these version folders are ever merged – they remain immutable once they have been created and remain separate and distinct when they are deployed to each computer. This mechanism is used by DGP to follow an immutable append-only pattern for deployments of applications, Windows services, web applications, and web services.

In these types of deployments, the version folder is copied below the parent application folders, alongside the folders of any other previous versions. Then the physical path of the application virtual directory is set to the new version folder. Rollbacks are accomplished simply by setting the path of the virtual directory back to the previous version folder.

By following this convention, the addition of new APIs, or new methods to existing APIs will never break backward compatibility. Also, in order for API methods to be used they must first be authorized in the data-driven RBAC security system. In this way, the security functionality is used as a mechanism to implement feature toggles at the API level. New methods are assigned only to test roles at first, so only testers are authorized to execute those methods in a given environment. If they pass the tests, they can then be added to other roles for more widespread use. Each environment has its own security database, so authorization of API methods is limited to the scope of the locations of each separate environment.

3. Native applications and Windows Services

Native .NET applications and Windows services each have a parent directory, and the individual deployment/version folders are copied under that parent folder. Therefore, the parent folder will end up containing a collection of one or more version folders. For native applications, a shortcut is created that points to the .exe file (assembly) in the latest version folder. For rollbacks, the physical path of the shortcut is pointed back to the .exe in the previous version folder. This same basic mechanism is used for Windows services with the .NET Framework `installutil.exe` utility, uninstalling the previous version and then installing the new version with a new path to the executable. The Windows service will very rarely need to be updated. Generally, the .NET applications are not published, since that process automatically adds quite a bit of unwanted functionality. Instead, the “release” folder under the “bin” directory is simply copied and renamed as a version folder. This is actually the easiest way to use the XCOPY deployment process for WinForm apps, console apps, Windows services, etc.

Continuous Testing

Continuous, partially automated testing is a prerequisite for the frequent deployments of CI/CD/CT processes. Each deployment requires that a full regression test of all system functionality be executed in each environment. In DGP, these full regression tests plus ad hoc tests are all run by the API Tester test harness that is included in the DGPDrive application. Access to the test harness functionality within the DGPDrive application is controlled by role membership in the security system. The same security controls access to the testing API methods used by the test harness as well.

Almost all functionality in a DGP system is implemented as web service API methods, and every API method contains various built-in tests (analogous to a unit test that is permanently built into each API method). The API Tester test harness uses test files to create API request messages that call the API method just like a DGP application would. The test files contain both positive and negative tests for each API method, exercising the primary and alternate paths through the code.

The two-way communication of "extra" data between the client and the server that occurs when using message-based APIs is the functionality that makes the automated testing of all API methods possible. In this way, the method results, the results of the built-in tests, and the server performance metrics are all returned as part of every API response message. This functionality provides the foundation for the logic of the API test harness app built into DGPDrive.

The test harness is generally run after deployments to each environment (except Prod to avoid creating large volumes of test data mixed in with real production data in the data storage of production systems). Even though the test files are built to delete all of the test data they create, if any problems occur during testing, test data that was not deleted would then be visible in the production systems. How much of a problem this represents would vary from one system to the next. If test data can be partitioned and hidden from users in production environments, then the API tester could also be run in those production environments, and the QA environment would not be needed.