Prerequisites and Limitations

Clock Synchronization

The system clocks of the computers in the server tiers of an environment's multiple locations must be set to use UTC (consistent across all geographic locations plus no seasonal changes) and also be closely synchronized with each other for the "Last Write Wins" (LWW) logic to work properly. In practice, a drift of 1 second or less will generally be fine for most systems, and that can be achieved just by using the normal mechanism for synching system clocks to a network time server. Windows Server 2016 and later has the capability to synchronize and maintain distributed system clocks to within 1 MS for systems that require that level of accuracy, at the price of some additional hardware and increased complexity.

https://docs.microsoft.com/en-us/windows-server/networking/windows-time-service/support-boundary

https://docs.microsoft.com/en-us/windows-server/networking/windows-time-service/configuring-systems-for-high-accuracy

Note: the fact that DGP preserves every version of every replica record minimizes the consequences of the LWW logic, since the only data affected is the value assigned to the record state field during replication. If the wrong state value is applied, it can be corrected fairly easily at any time afterward.

Important Note: some of the data created by steps in the setup process must be saved securely. The values to be saved include the credentials for the DGP super admin account, encryption keys, ADO.NET connection strings, and so on. Losing those values can lock administrators out of their own systems.

Git Ignore

Git repositories need to be configured to ignore *.config files so that configuration data is not merged into the master Git repo.
https://docs.microsoft.com/en-us/azure/devops/repos/git/ignore-files?view=azure-devops&tabs=visual-studio

IMPORTANT NOTE: since the Git repository is configured to ignore all *.config files when synchronizing code in Visual Studio, a generic sample of each .config file is included in the source code as *.config.template. Create a copy of the file without the ".template" extension, and then edit the generic entry values to match a given environment of the location.

ASP.NET 4.8.1 Garbage Collection

Much of the ASP.NET GC configuration will be done automatically based on the Host and the number of cores. Background collections are the default for both Workstation and Server GC. Workstation GC will be used on Windows 10, and Server GC on Windows Server.

Setup Overview

New software system built using DGP as a foundation will generally have multiple environments (Dev, Test, QA and Prod). environments. The DBSetup utility is used to build the database schemas and populate core data in each environment.

Preparation

1. Prepare the Windows 10/11 or Windows Server 2016 (or later) operating system
2. Install the full .NET Framework 4.8.1 for Windows (runtime or developer pack)
3. Install the Windows IIS 10.x web server, configured to use ASP.NET
4. Install SQL Server Express or Standard edition 2016 (or later) database engine

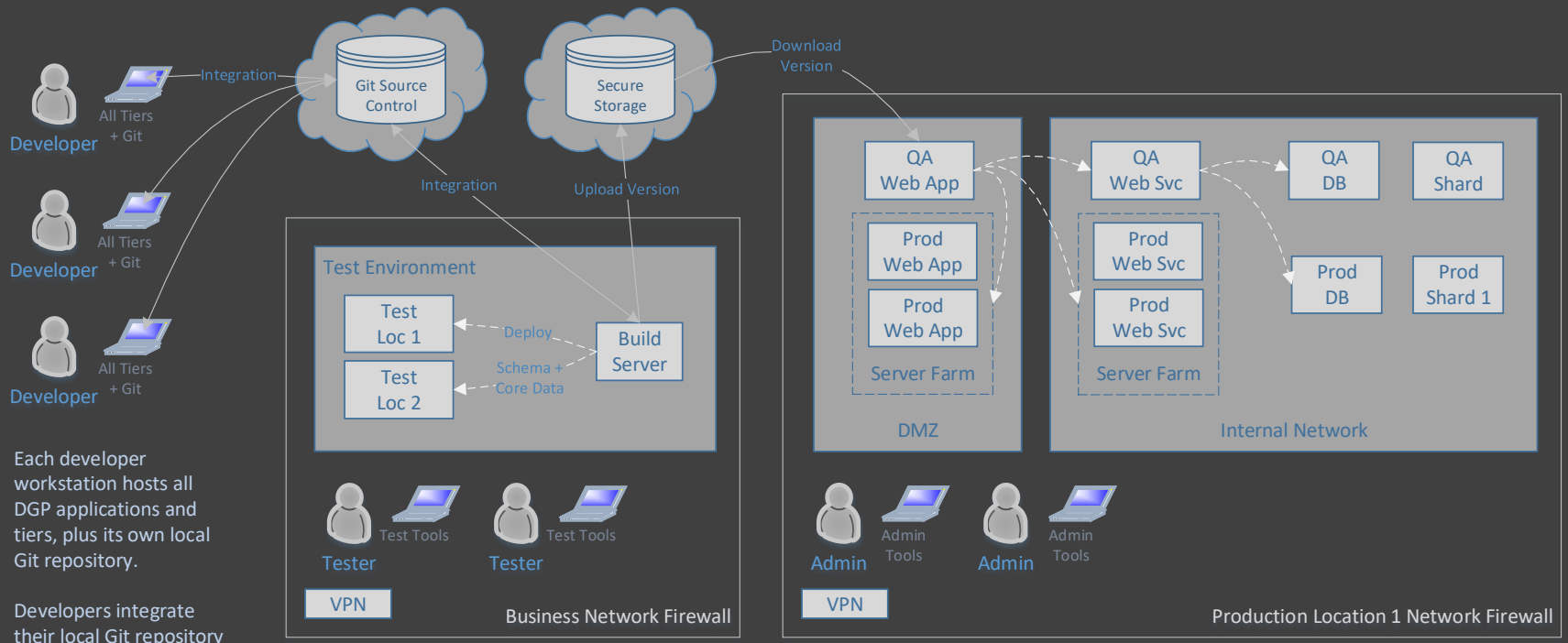
DBSetup

5. Use SSMS to create a set of empty DGP databases (documented in the DBSetup user guide). The database names can follow whatever naming convention is desired, but the must be unique within the locations of a system (for DGP data replication).
6. Run the DGP DBSetup utility to build the database schemas and populate the core data in the empty databases. This utility must be run by administrators from a location inside the firewall and have access to the database server.
7. Save information from the DB Setup utility for later configuration steps (DB names used, admin accounts created, ADO.NET connection strings generated, service encryption key used, etc.)

Web apps and web services

8. Create a parent directory for the DGPWebApp and DGPWebService
9. Copy the DGP web app and web service release folders as subdirectories below their parent directories
10. If they do not already exist, create the IIS WebApp and WebSvc applications using the release directories

11. Edit the web.config file of the web app reverse proxy to use the URL of the web service
12. Use the data saved from the DBSetup utility to edit the WebSvc Web.config file
    a. If the web.config section is encrypted, decrypt it with the aspnet_regiis.exe utility.
    b. Edit the .config file entries for the location and environment
        i. Svc_Key entry: this is the AES encryption key used for application-level encryption. If DGP omnidirectional replication is used, the key values used for each location **must be identical.** The keys are managed by following the immutable append only convention (add new keys to the collection, never editing or removing any previous key values; then set which key label should be used as the active (latest) encryption key).
        ii. The set of ADO.NET connection strings
    c. Optionally, use the aspnet_regiis.exe utility to encrypt the section of the web.config file.



Each developer workstation hosts all DGP 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

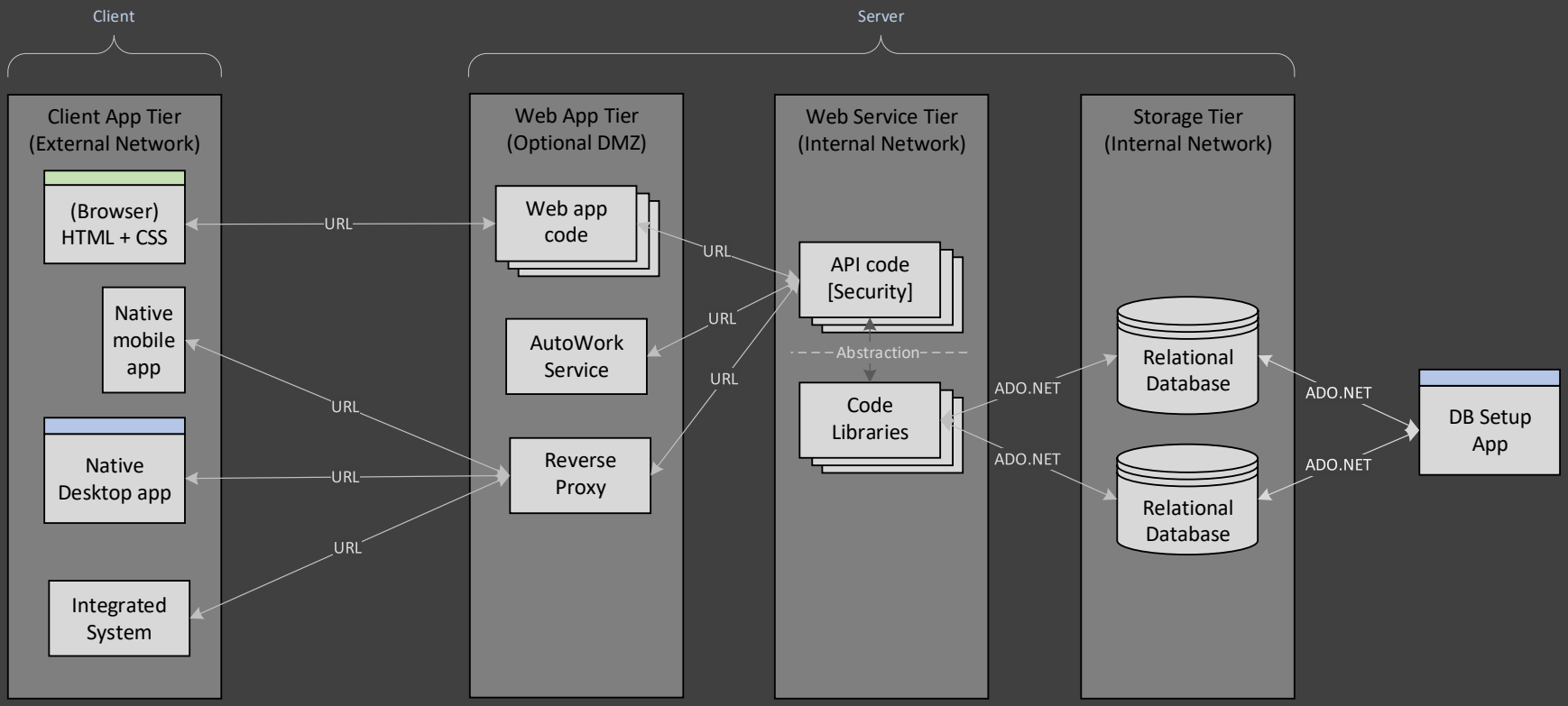
The build server in the test environment pulls source code from the central Git repository to create clean daily version builds (no merging). Each build is saved in its own daily version folder, named for the date (YYYY-MM-DD).

Testers also push edited test files and/or documentation up to the central Git repository, to be integrated with each development environment. In practice, the build server will generally run Visual Studio for manual integrations, and tools such as Jenkins to orchestrate scripted processes.

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

Production admins pull the version folders down from the storage to a QA web server in the DMZ, which is then used to deploy to the other QA servers. The DB Setup utility is deployed to the QA Web Svc server, and the utility is used to update the QA database schemas + core data. If testing in the QA environment is successful, then the version folders are deployed to the production servers. The DB Utility is again used to update the production database schemas + core data. All of these steps will generally be performed as part of the periodic location maintenance processes.

DGP tiers use either URL or ADO.NET connection string endpoints to call RPCs in order to work together. When all tiers are installed on a single computer, these endpoints are configured to use the localhost loopback network address for the server, which directs the network call back onto the same host computer. As a location scales in size to deploy each tier on its own separate computers, the endpoints are configured to use the IP addresses of those other computers as the server rather than the loopback address.



For software development, a system will also have multiple environments, each of which has its own separate databases and data. The most important part of this arrangement is the separate security data in the SysInfo database of each environment. In practice this means that each environment has its own separate user accounts, role memberships, etc. The DB Setup utility maintains the core security data to be consistent across all environments and locations, but user accounts (other than the main DGP system admin account) are created and maintained by the DGPDrive admin UI's in each environment. Replication synchronizes the data between the multiple locations of each environment, as applicable (the Dev environments will have only one location).

Since the RBAC security system is used as the primary feature-toggle mechanism in DGP systems, this means that each environment can have different sets of API methods enabled and disabled compared to the other environments.

Windows Host (Windows 10 and Windows Server)

Windows 10 is fine to use for development machines, but due to its built-in limitations for the number of concurrent IIS connections it may be unsuitable for the Testing, QA and Production environments. Windows Server Standard edition should be used for those environments. DGP has been designed and built using only Microsoft products, and has no dependencies to any 3rd party tools.

DGP dependencies:

- Windows 10/11 or Windows Server Standard (2016 or later)
- Full .NET Framework 4.8.1
- IIS 10.x Web Server
- SQL Server Express or Standard (2016 or later)
- Visual Studio Community Edition or Professional (2019 or later)

By only using (closed source) Microsoft products, this allows Windows updates to patch every tier and part of DGP systems in a unified way. Each version of Windows has an option to update other Microsoft products as part of the Windows update process, and that option should be enabled for DGP systems. This allows for the "unified patching" of all elements and tiers of a DGP system as part of the standard Windows updates. In addition, avoiding the use of any open source projects means that there is no need for expensive software tools to scan the entire tree of dependencies of each open source project looking for injected malicious code.

DGP Data Setup Steps

SSMS is used to create empty databases so that they can follow the desired naming conventions, storage file locations, and so on. This is also a security feature, in the same way that there is no default admin account username/password. Most aspects of a DGP installation are customized per location and environment for each system.

1. Use SSMS to create a set of empty DGP databases (documented in the DBSetup user guide). The database names can follow whatever naming convention is desired, but the must be unique within a system (for DGP data replication).
2. Run the DGP DBSetup utility to build the database schemas and populate the core data in the empty databases. This utility must be run by administrators from a location inside the firewall and be able to connect to the database server.
3. Save information from the DB Setup utility for subsequent application configuration steps (DB names used, admin accounts created, ADO.NET connection strings generated, service encryption key used, etc.)

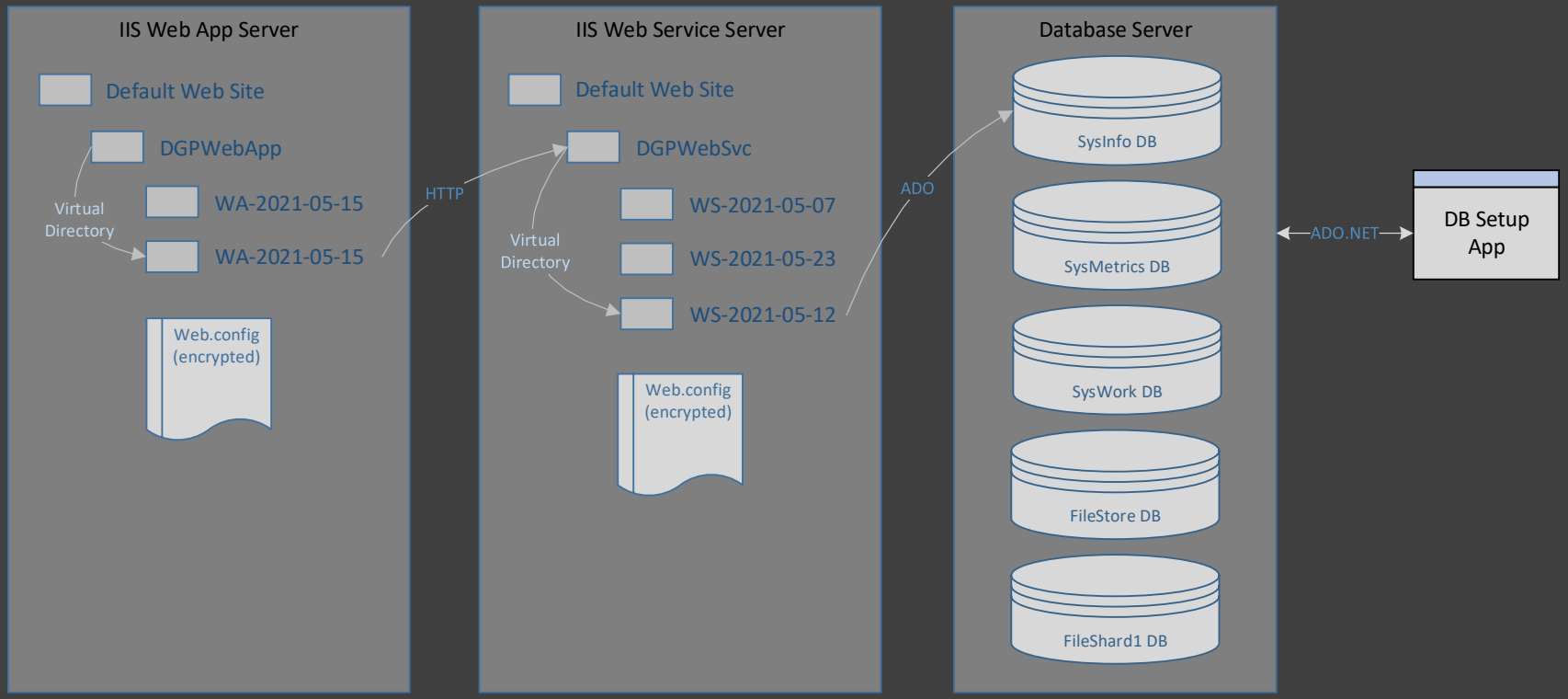
DGP Application Deployment Steps

All DGP software uses the XCOPY deployment mechanism of the .NET Framework assemblies. Using this mechanism, apps and services are deployed by copying a folder containing the .NET assemblies to the local storage of the target computer. Uninstalling a DGP app is done by deleting the folder. The date of the build is used as the version number, and each new version folder is added as a subfolder under the parent folder of the respective application or service. These folders are the parent directories that will be used for deployments of specific versions of their respective DGP apps:

1. On the computer used for IIS Web Services
    a. Create a folder \Inetpub\wwwroot\DGPWebSvc
        i. Copy the assembly folder dgp_websvc_date below the parent folder (add it to the collection of other date subfolders – do not change or delete previous subfolders)
        ii. Edit the Web.config file, or copy it from a previous version
        iii. Edit the path of the web service virtual directory to use the content of the new date subfolder
        iv. Rollback: edit the VM path to use the previous date subfolder and delete the failed date subfolder

2. On the computer used for IIS Web Apps
    a. Create a folder \Inetpub\wwwroot\DGPWebApp
        i. Copy the assembly folder dgp_webapp_date below the parent folder (add it to the collection of other version subfolders – do not change or delete previous subfolders)
        ii. Edit the Web.config file, or copy it from a previous version
        iii. Edit the path of the web app virtual directory to use the content of the new date subfolder
        iv. Rollback: edit the VM path to use the previous date subfolder and delete the failed date subfolder

3. On the computer used for Client Applications
    a. Create a folder DGP_ClientApps
    b. Create a subfolder DGP_ClientApps\DGP_AutoWork
        i. Copy the assembly folder dgp_autowork_date below the parent folder (add it to the collection of other version subfolders – do not change or delete previous subfolders)
        ii. Edit the App.config file, or copy it from a previous version
        iii. Create a shortcut to the .exe of the new version, or edit a previous shortcut path
        iv. Rollback: edit the shortcut to use the path to the previous .exe and delete the failed date subfolder
    c. Create a subfolder DGP_ClientApps\DGP_DBSetup
        i. Copy the assembly folder dgp_dbsetup_date below the parent folder (add it to the collection of other version subfolders – do not change or delete previous subfolders)
        ii. Edit the App.config file, or copy it from a previous version
        iii. Create a shortcut to the .exe of the new version, or edit a previous shortcut path
        iv. Rollback: edit the shortcut to use the path to the previous .exe and delete the failed date subfolder
    d. Create a subfolder DGP_ClientApps\DGP_Drive
        i. Copy the assembly folder dgp_lattice_date below the parent folder (add it to the collection of other version subfolders – do not change or delete previous subfolders)
        ii. Edit the App.config file, or copy it from a previous version
        iii. Create a shortcut to the .exe of the new version, or edit a previous shortcut path
        iv. Rollback: edit the shortcut to use the path to the previous .exe and delete the failed date subfolder



Each published build of a .NET web app/service produces a folder containing the assembly and any other files it needs to run under the .NET framework installed on each host. These build folders are named for the date they were published. Deployment consists of copying the entire build folder below the directory of the web app or web service on the web server. The web.config file of each app or service contains URL endpoints, ADO.NET connection strings, etc. The web.config of the latest date folder must be modified to include the correct endpoint configurations, usually by copy/pasting them from a previous version. NOTE: In some environments, sections of the web.config file will be encrypted to protect sensitive data using Microsoft's ASPNET_REGIIS.exe utility.

Once the latest date folder web.config file has been edited correctly, the physical path of the web app/service virtual directory is changed to point to that folder. A full regression test is then run to verify all funcitonality. If any problems are encountered, the deployment is rolled back by changing the virtual directory path back to the previous folder, and deleting the failed build folder.