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

ASP.NET 4.8 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 have Dev, Test, QA and Prod environments. Dev and Test and Prod are self-explanatory, while QA is a subset of a production environment used to allow all tests to effectively be run in Prod without contaminating production data with large amounts of test data. New systems set up to run Lattice and/or verify DGP can skip the Dev, Test and QA environments and only create a Prod environment.

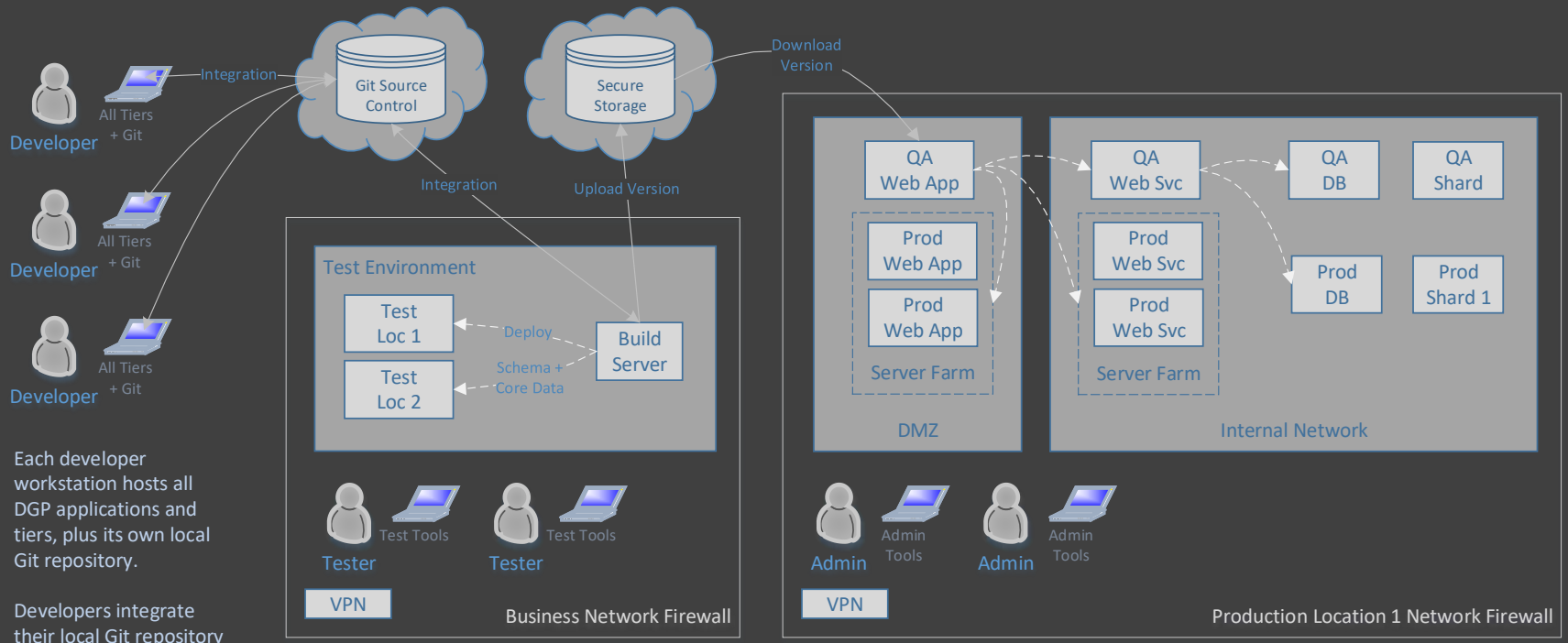
The steps to setup and initially configure a new DGP system are as follows:

1. Prepare the Windows 10 or Windows Server 2016 (or later) computer
2. Install the full .NET Framework 4.8 for Windows
3. Install the Windows IIS 10.x web server, configured to use ASP.NET
4. Install SQL Server Express or Standard edition
5. Deploy the DGP web apps to a local directory
6. Create the IIS WebApp and WebSvc applications using the local directories (use default web site or create new web site)
7. Use SSMS to create a set of empty DGP databases
8. Deploy the DGP client applications to a local directory
9. Run the DGP DBSetup utility from the local directory to create the database schemas and populate the core data.
10. Save information from the DB Setup utility for configuration.
11. Use the data saved from the DBSetup utility to edit the WebApp and WebSvc Web.config files
12. Edit the System List file for the DGP Lattice application to use the URL endpoints of the WebApp proxy and WebSvc

To verify that the deployment has been setup and configured correctly, use the DGP Lattice application to connect to the new WebSvc and WebApp proxy. Run a full regression test using the API Tester.

DGP systems are distributed grids that are spread out across multiple separate locations. Within a location, at the smallest scale, all DGP tiers and apps are installed on a single Windows computer or VM. The Windows computer functions as a web server and database server, and optionally can also run the DGP client applications. This configuration is used for software development, testing and debugging, and very small-scale production systems.

CI-CD-CT Example:



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).

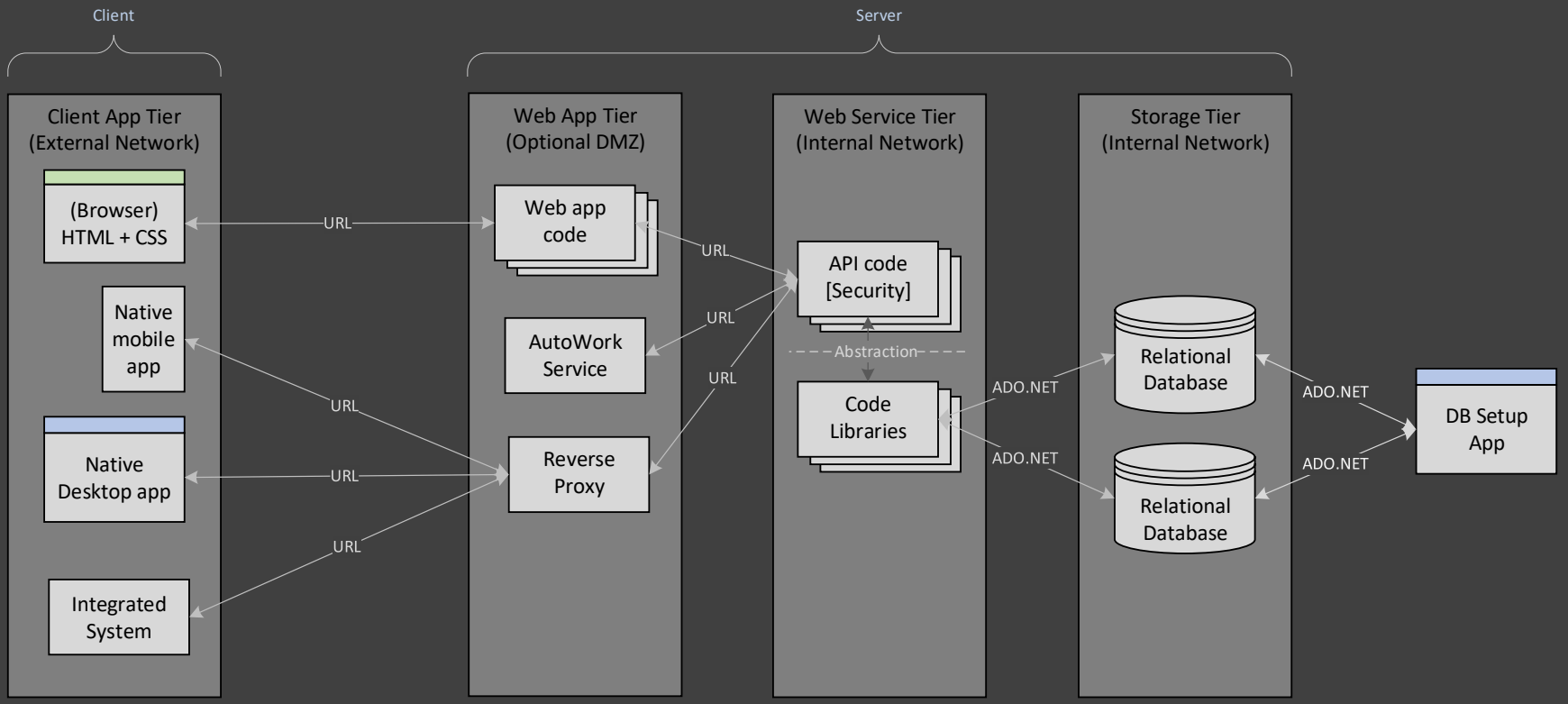
Production admins pull the version folders down from the storage to a QA web server in the DMZ, which is then used to deploy to the other QA servers. The DB Setup utility is deployed to the QA Web Svc server, and the utility is used to update the QA database schemas + core data. If testing in the QA 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.

System	Environment	Locations	Network	Description
DGP Lattice	Dev: Windows 10	1	localhost	The dev environment is a self-contained single-server install that includes all DGP client applications, the IIS web server tier and SQL Server database server tier. All endpoint URL's and ADO.NET connection strings are configured using the localhost loopback network address.
	Test: Windows Server	2	Internal (VPN)	The Test environment separates the client applications (deployed on the build server) from two single-server locations that host the server tier applications. The client applications will use an internal network endpoint URL's, while each server will use localhost endpoints.
	QA: Windows Server	2-3	Internal (VPN)	The QA environment is part of the Prod infrastructure, and is used to test new deployments in Prod without comingling test data with production data. Client apps will use external URL endpoints to connect to a location, while the tiers will use localhost or internal network URL and ADO.NET endpoints. If a DMZ is present, the external URL will use a reverse proxy to access the internal API's.
			External	
	Prod: Windows Server	2-3	Internal (VPN)	
			External	

All the tiers use URL or ADO.NET connection string endpoints to call the RPC's used to work with each other. When all tiers are installed on a single computer, these endpoints are configured to use the localhost loopback network address, 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 computers rather than the loopback network address.

For software development, a system will also have multiple environments, each of which has its own completely separate deployment 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 an 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 Lattice admin UI's in each environment. Replication synchronizes the data between the multiple locations of each environment, as applicable (the Dev environments will each 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. This is especially useful in the production environment to enable the limited initial use of new API methods to specific roles before they are authorized (by assigning role membership) for more widespread use.



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 is 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.

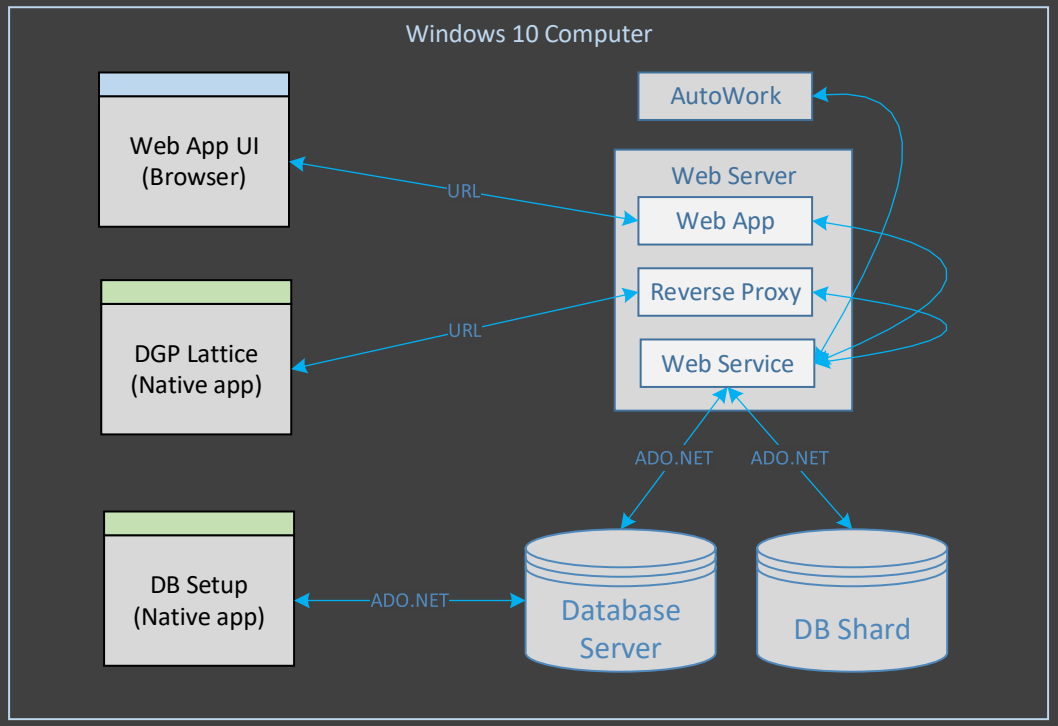
DGP dependencies:

- Windows 10 or Windows Server Standard
- Full .NET Framework 4.8 (closed source)
- IIS 10.x Web Server
- SQL Server Express or Standard
- Visual Studio Community Edition or Professional

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.

Each development computer is its own separate, stand-alone DGP environment and location. The development machines run Windows 10, and are the easiest to repair (or replace) if anything goes wrong with updates, especially if they are VM's. Automatic updates are fine for Windows 10 development computers. Each developer is responsible for Windows updates on their own development computers.

The other environments will all use Windows Server, and updates should be manual for the Test, QA and Production computers. Any update problems found for Test computers allows those problems to be resolved before they are applied to the QA environments. The same is true for update problems found for the QA computers being resolved before applying those updates to the production computers. Since the QA computers are close to identical to Prod computers and share the same infrastructure, updates should work for the production computers with no problems at that point.

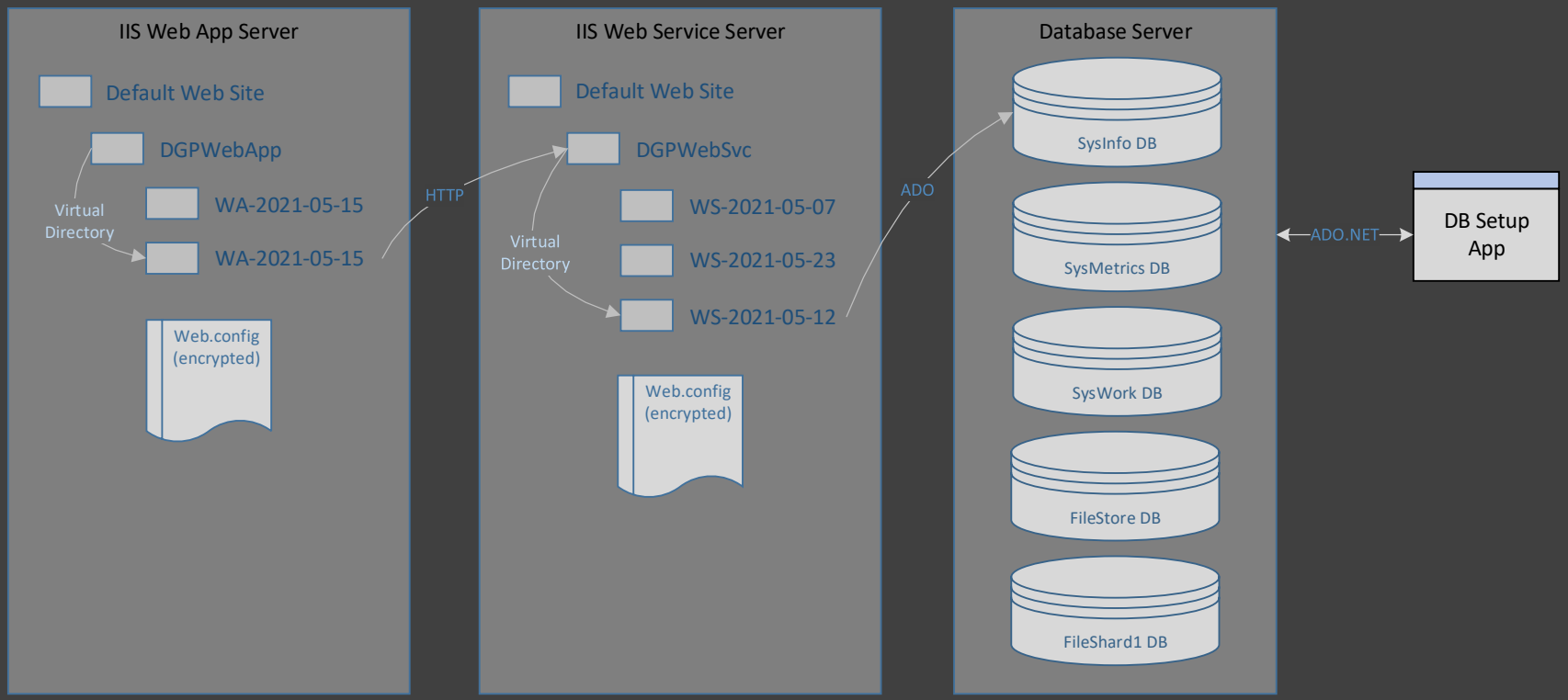


A Distribware system is comprised of 4 tiers:

- Client tier
  - Web app UI
  - Native apps
- Web app (DMZ) tier
  - Web app logic
  - Reverse proxy
  - DGPWinSvc scheduler
- Web service (API) tier
- Storage tier
  - Database server
  - DB shards

The configuration in the diagram above shows a development computer hosting all of the tiers of a DGP system. Small-scale systems are similar, but use Windows Server and will generally only host the server tier components. This configuration will be used for a minimum of two servers (and ideally three) set up in different locations. Once installation is complete, replication and data check processes are configured between the various locations for data redundancy and overall system fault tolerance.

Once a system outgrows a single computer per location, the logical tiers and apps are deployed to their own separate physical tiers. From that point onward, each physical tier is able to scale independently of the other tiers, as needed. Also, the network topologies of each location change with the introduction of a DMZ, etc.



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.

Setup 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 each 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 Staging
    a. Create a folder DGP_Staging
    b. Create the subfolder DGP_Staging\DGP_date, where date is replaced by the date of the build or release.
    c. Download the .zip file from the documentation web app into the matching DGP_date subfolder
    d. Calculate the MD5 hash value to verify the .zip file https://docs.microsoft.com/en-us/troubleshoot/windows-client/deployment/compute-md5sha-1-cryptographic-hash-value
    e. Extract the contents of the zip file into the DGP_date subfolder which will create the following directory structure:

```
DGP_date
    DGP_Apps
        dgp_autowork_date
        dgp_dbsetup_date
        dgp_lattice_date
    DGP_WebApp
        dgp_webapp_date
    DGP_WebSvc
        dgp_websvc_date
        DGP_source_date
            (source code folders)
```

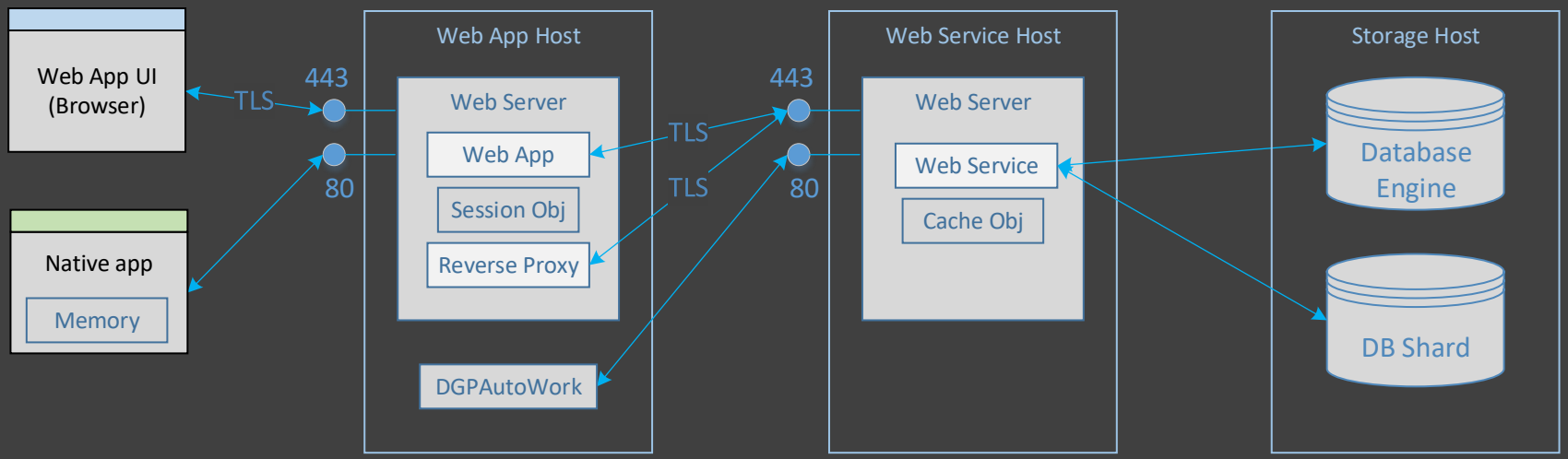
2. On the computer used for IIS Web Services
    a. Create a folder \Inetpub\wwwroot\DGPWebSvc
        i. Copy the assembly folder dgp_websvc_date below the parent folder (add it to the collection of other date subfolders – do not change or delete previous subfolders)
        ii. Edit the Web.config 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

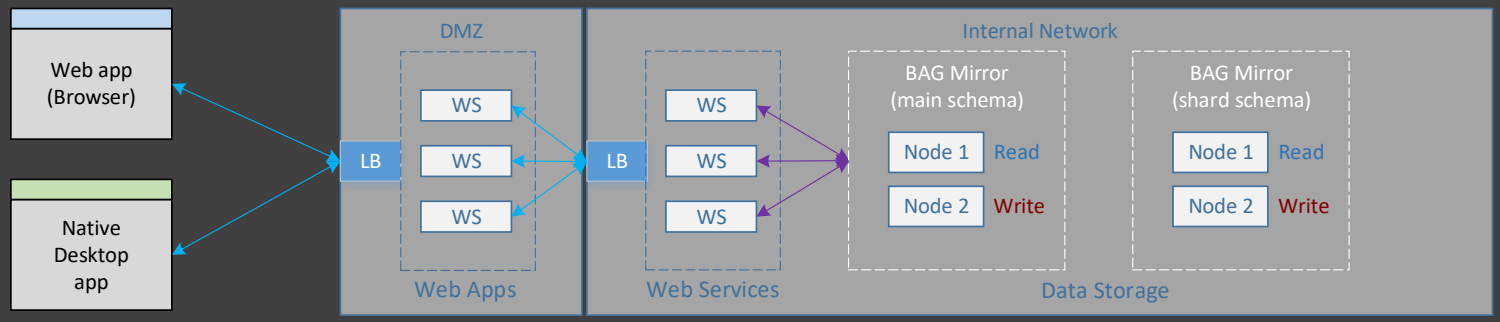
3. On the computer used for IIS Web Apps
    a. Create a folder \Inetpub\wwwroot\DGPWebApp
        i. Copy the assembly folder dgp_webapp_date below the parent folder (add it to the collection of other date subfolders – do not change or delete previous subfolders)
        ii. Edit the Web.config file, or copy it from a previous version
        iii. Edit the path of the web 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

4. 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 date 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 date 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_Lattice
   i. Copy the assembly folder dgp_lattice_date below the parent folder (add it to the collection of other date 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





LB = Load Balancer | WS = Web Server | FS = File Server

Scaling the 4-tier architecture within a location is a matter of first separating each tier onto its own hardware, from which point each tier is able to scale independently of the others. The web server farms handle both N + 1 fault tolerance and incremental horizontal scalability for both the DMZ and internal network. The SQL Server Availability Groups provide redundancy for the various pairs of database servers. The main schema servers can only scale vertically. The shard schema servers provide horizontal scalability for shard data by incrementally adding more shard servers as needed.

DGP Data