DGP Maintenance

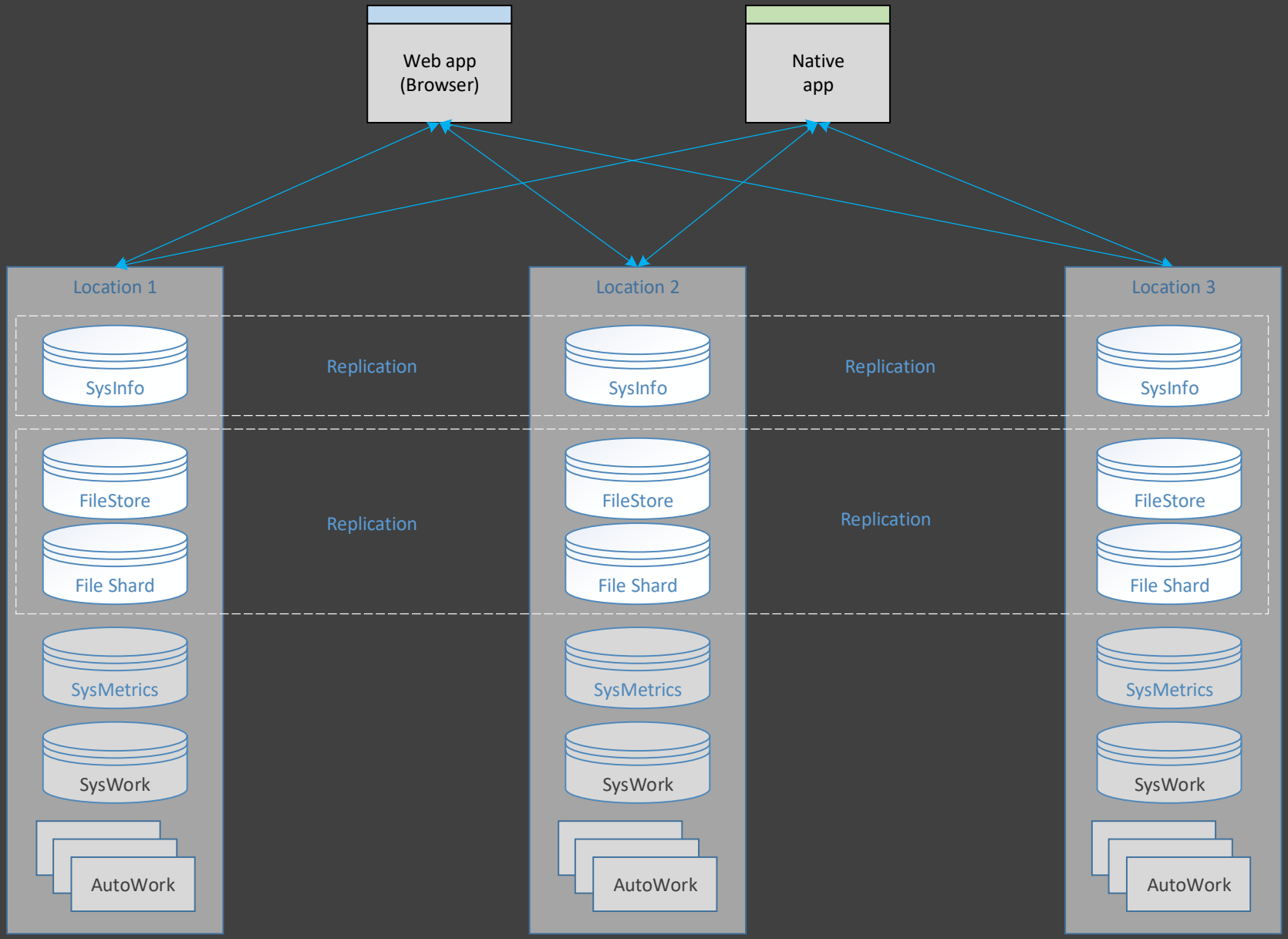
Maintaining a distributed system across multiple locations adds a few extra steps to the usual maintenance processes, but otherwise is fairly standard as long as a system is maintained one location at a time, sequentially.

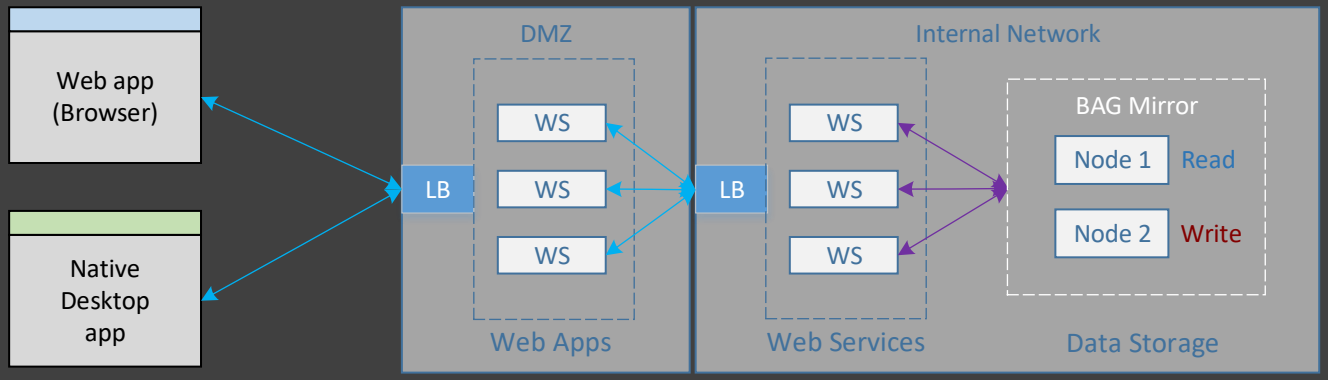
The documentation for maintaining the production environment a DGP system assumes that 3 locations are up and running and that they are separate infrastructure from the rest of the business systems. Whether they are VM's or bare-metal servers does not matter for this type of maintenance (but servers hosting VM's also need to be updated, patched and maintained on the same monthly schedule).

For small scale systems, a location must be "turned off" prior to the maintenance,. This means stopping the DGPAutoWork services in that location, and also disabling the web apps and services in that location. The app_offline.htm file can be used to shut down an ASP.NET app or service, which will effectively shut down a location. Once a location has scaled up to use load balancers, disabling the web apps and services can be done at that level as an option as well.

The DGPAutoWork windows services at other locations will not be turned off, so their automated processes will be affected when the web service API's at a location are disabled. Those processes will produce a special "offline" error, which cause the next scheduled iteration of the process to be slowed down (the default is a 10 minute interval, but that is configurable). This slowdown will end as soon as the location is brought back online after maintenance and no longer returns an "offline" type of error.

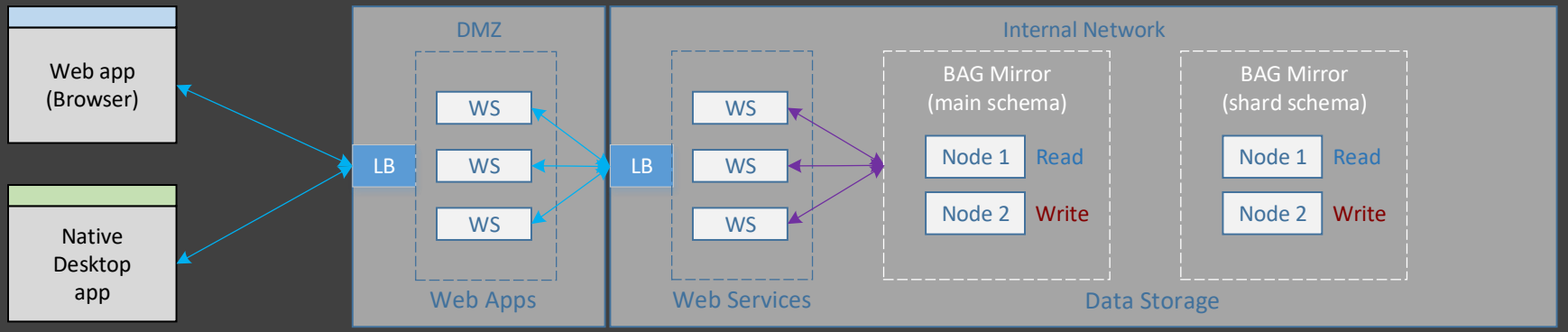
Large scale systems will usually have redundancy within each location in addition to the redundancy between locations. In this situation, it should be possible to perform maintenance sequentially on the computers in each location while the location remains in use. The diagram below shows a location with this level of redundancy. Each web server in the DMZ and Internal server farms can be removed from the farm, one at a time, updated and maintained, and then put back into the server farm. Each web server can be maintained sequentially using this methodology while the location remains in use. The same is true for the nodes in the SQL Server Availability Group (mirrored databases). Individual nodes can be removed from the AG, maintained, and then added back into the group when done.





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

The FileStore application in Lattice uses the FileSeg schema with file storage shards, as shown below.



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

Windows Host Maintenance

Different environments for a system will have different configurations. For example, development computers will be single-server Windows 10 installations. The testing environment will be similar but will use Windows Server instead of Windows 10. The QA environment should have one Windows Server per tier. The production environment will be identical to the QA environment, but each tier will be scaled to handle production workloads with the introduction of load balanced server farms, and so on.

The single most important activity to increase the security of a system is to apply updates to all elements and tiers of that system quickly, as soon as they become available. DGP has been designed and built using only Microsoft products, and has no dependencies to any 3rd party tools (no dependencies to open source).

DGP dependencies:

- Windows 10, Windows Server Standard or better
- Full .NET Framework 4.8
- IIS 10.x Web Server
- SQL Server Express or better
- Windows File Sharing
- Visual Studio Community Edition or better

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.

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.

Windows updates in general are very reliable. In the recent past there were internal problems at Microsoft that caused some update problems for a period of time, but those internal problems have largely been resolved, and Microsoft has since reestablished its emphasis on Windows as a whole, so Windows updates should continue to be very reliable going forward. This is an important advantage that Windows has relative to other operating systems. As stated previously, the single most important practice to significantly improve the security of a system is the quick application of updates and patches. In order for that to be feasible in production, the update process itself must be very reliable over time.

The Test and QA environments can be updated at any time by coordinating the maintenance downtime with the testers. Production environments should have at least 2 and generally 3 locations. Each location should be taken offline sequentially, coordinating the maintenance downtime with users to inform them of the need to use other locations of the system during the time a given location is unavailable. The page app_offline.htm file can be used with the IIS web server and ASP.NET to shut down a web application, unload the app domain, and not accept any new requests.

While a location is offline and after the individual Windows servers have been updated, additional processes can be performed to maintain the various application servers running on the Windows hosts. These processes for IIS web server and Windows file server maintenance are documented separately. The SQL Server database engine has the most extensive maintenance process, and is also documented separately.

SQL Server Maintenance

The Microsoft documentation for maintaining SQL Server should be used as the basis of any management plan.

[https://docs.microsoft.com/en-us/sql/relational-databases/maintenance-plans/maintenance-plans?view=sql-server-ver15]

Once all of the maintenance steps for SQL Server have been completed, then the DB Setup utility would optionally be run to maintain the DGP schemas and core data. As a system evolves over time, changes to the schemas and additional core data are added to the DB Setup utility, which then adds them to all of the databases of the system. Changing the database schemas can easily cause deadlocks if they are done while the system is in use, therefore the DB Setup utility should only be run when the database server has been taken offline for maintenance and patching.

Important Note: Many type of modifications to DGP schemas will use a SCH-M (schema modification) lock. In addition, some modifications will cause a table to be rebuilt, including all of its data – which can take a long time to complete if the table contains many records. For this reason, it is best to run the DB Setup modifications as part of a regular update and maintenance process, and NOT as part of a more frequent deployment and testing process. In other words, the database and/or host server being maintained should be offline for the duration of the DB Setup modifications.

Updates

Standard Windows 10 or Windows Server updates for standalone servers, and Cluster Aware Patching for both SQL Server Availability Groups and file server Failover Clusters, etc.

Scaling

DGP relies on primarily on the vertical scaling of SQL Server for most of its databases. Overall system performance and scalability are both based on the use of as many TB of solid-state storage within each server as needed, each providing many hundreds of thousands (up to millions) of IOPS to access that storage. This practice effectively removes storage and storage IOPS as the main performance and scalability bottleneck in a system, and moves the bottleneck to the network appliances (firewalls, load balancers, etc).

Some databases can use DGP's hybrid storage, which combines a primary database with a collection of file server or database server shards (Lattice uses this type of file server hybrid, for example). This provides a very simple and inexpensive mechanism for

incremental horizontal scalability to store very large amounts of unstructured data. However, not all schemas support the type of vertical partitioning that is required for the shards to be effective.

Fault-Tolerance

The multiple active locations of a DGP system provides the primary level of redundancy, and that is the only redundancy used in smaller systems. As the locations of a system scale up and out in capacity, additional redundancy is also needed within the locations themselves so that they do not become too fragile, reducing the reliability of the system as a whole. For SQL Server, Basic Availability Group mirroring of databases between two database servers is a good alternative.

Offline Backups

Having multiple online copies of all data does not eliminate the need for offline backups. That being said, a private read-only location used as an archive could in theory be well-protected enough to reduce (and possibly eliminate) the need for offline backups.

In-Place Data Repair

DGP includes automated processes to periodically scan all replica data for problems, and when found to log the problems to the error logs. Gaps in replication can be fixed by running an additional replication process (VERIFY) in parallel with the main replication to crawl through the existing data and replicate the missing records. The original mechanism to verify the state of replication was to periodically run the VERIFY scans to insure all records had been replicated, but that process can take a long time to complete. The COUNTCHECK verification is faster and more efficient, but the previous mechanism can also be used whenever it is needed.

Extra or duplicate records are found they must be analyzed by administrators to determine how best to fix that problem. It must be fixed in all locations at roughly the same time, or the next VERIFY scan would “restore” those extra records to each location.

Replication/Verification can be used to completely rebuild database tables, but that could take a very long time for tables with a large number of records. Restoring from a backup first, and then using replication to synchronize the remaining data not included in the backup would be the best solution in that case.

*Important Note: For DGP hybrid storage, replication and verification repairs both the replica database records and the file storage as well, fixing records missing from the database tables and/or any files missing from the file storage at the destination.

Inconsistent Schemas

In distributed systems that are constantly being extended and improved, it is inevitable that patching and schema modifications will not be exactly simultaneous across all of the databases in a system. DGP Lattice handles these inconsistencies by using a combination of the immutable append-only convention, the flexibility built into both the mapper methods and data access methods, and the ability to use the data-driven RBAC system as a pervasive feature toggle mechanism to restrict access to functionality as needed.

Following the immutable append-only convention means that you only append (add) new elements to a system and never edit or remove existing elements that can break backward compatibility. For example, you can add a new column to an existing table, add a new table to an existing database, or add an entirely new database to the system without causing any compatibility problems for the existing code.

Changes to the data storage are applied to a system first. During this process, no code that uses any of the new elements has been deployed. Only after all the storage in all the locations have been successfully updated can new versions of the API's that use those new elements be deployed.

After deployment of the new code, the data-driven RBAC acts as a feature toggle mechanism to make sure that only testers are authorized to call those new methods at first. If the testing is successful, then as the final step more roles can be authorized to use the new methods.

The final step is to deploy new versions of client apps that use the new API methods. The applications themselves have a feature toggle mechanism that checks the version date of the connected web service to enable/disable functionality that depends on specific (or newer) API versions.

IIS Web Servers (Web Apps and Web Services)

The IIS best practices documentation at Microsoft should be used to configure and maintain the IIS web servers.

[https://techcommunity.microsoft.com/t5/core-infrastructure-and-security/iis-best-practices/ba-p/1241577]

Updates

Standard Windows 10 or Windows Server updates for standalone servers is the same for single computers and web servers in a load-balanced server farm. In the dev environments, automatic updates can be enabled. In the Test, QA and Prod environments, manual updates should be used sequentially in the Test, QA and Prod environments.

Scaling and Fault-Tolerance

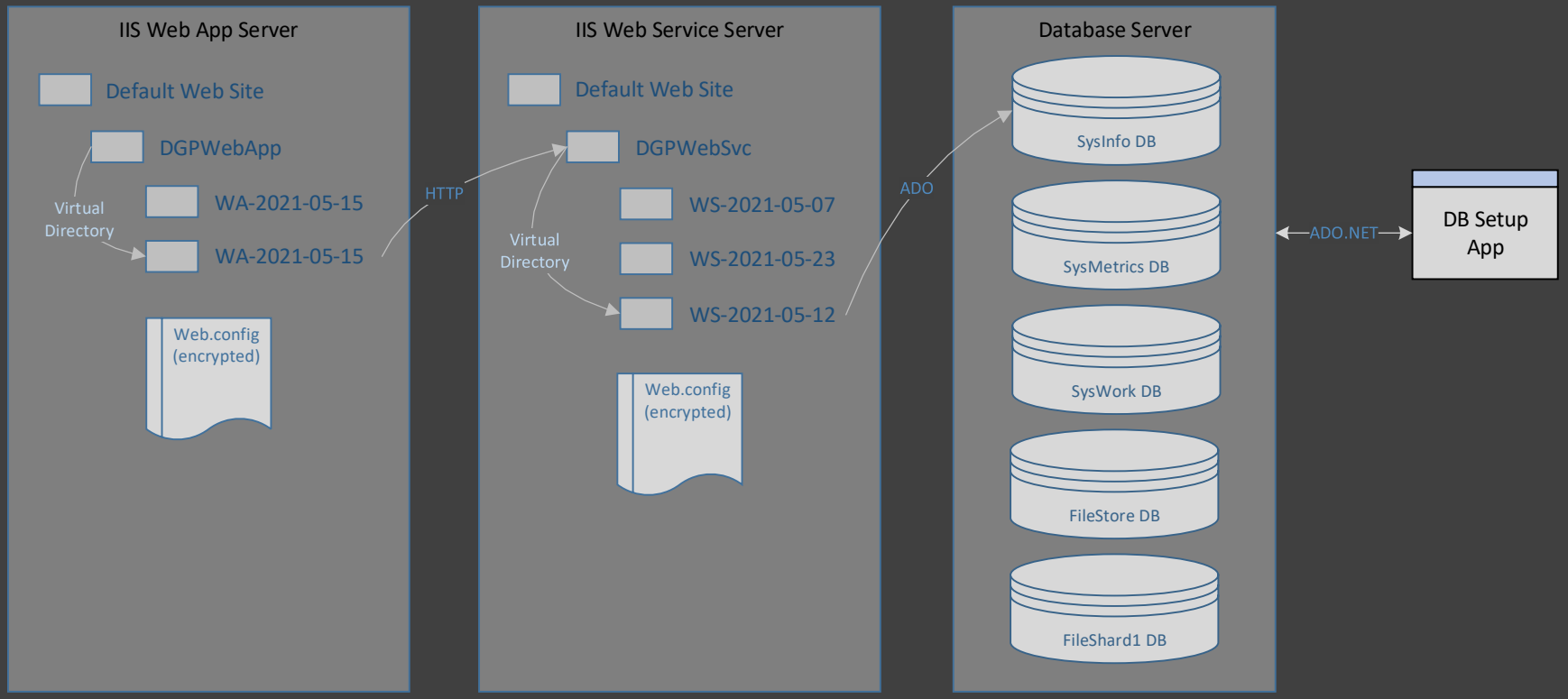
The multiple active locations of a DGP system provides the primary level of redundancy, and that is the only redundancy used in smaller systems. As the locations of a system scale up and out in capacity, additional redundancy is also needed within the locations themselves so that they do not become too fragile, reducing the reliability of the system as a whole. For web servers, load-balanced server farms provide both horizontal scalability and fault-tolerance when extra web servers are added to the server farms. Microsoft documentation explains how to reuse the same machine keys for all ASP.NET web servers in a server farm, for example.

Application Deployment

The deployment and maintenance of all of the tiers of a DGP Lattice system is both standardized and greatly simplified using two concepts. The first is the XCOPY deployment option for .NET applications which is used to create a folder that contains the application assemblies and associated files, and the second is the immutable append-only pattern used so frequently in all parts of DGP systems. All parts of DGP Lattice have been built with the full .NET Framework 4.8, and follow this same XCOPY immutable append-only deployment pattern.

The first step is to create a parent folder for an application on a computer. Native client applications are built with the “release” option in Visual Studio or a build server. This produces a streamlined assembly in the release folder of the solution that does not contain debug data. This release folder is then copied and renamed using the current date as its version number. Deployment of the app is done by copying the version folder under the parent application folder on the client computer. The config file in the version

folder is edited with values for the given environment. Frequently, the config file is simply copied from a previous version folder into the new version folder. Then a shortcut is created to launch the "exe" (.NET assembly) in the new version folder. Rollbacks are accomplished by pointing the path of the exe back to the prior version folder, which remains unchanged (immutable) during the deployment process.



The process for ASP.NET web apps and web services (shown in the diagram above) is very similar. A parent folder is created on the web server below the appropriate web site. Web apps and web services are "published" in Visual Studio or a build server into a local folder. The publish process in Visual Studio allows for several pre-compilation (Ahead of Time) options.

This folder is then copied and renamed using the current date as its version number. Deployment is done by copying the version folder under the parent folder of the web app. The config file in the version folder is edited with values for the given environment. The path of the virtual directory for the web app/service is then edited to point to the release folder.

New releases repeat this process, copying a new release folder under the parent folder of the web application. The .config file is edited in the new release folder. Frequently, the config file is simply copied from a previous version folder into the new version folder. Then the path to the virtual directory for the web app/service is then edited to point to the new version folder. To rollback a new version, the path to the virtual directory is pointed back to the prior version folder.

This is among the simplest mechanisms possible for ease of deployments and rollbacks. Each native app, web app and web service follow the immutable append-only pattern internally, which means that each new version contains all of the functionality of previous versions, plus some new functionality and/or bug fixes. This enables continuous system evolution and the CI/CD/CT processes without ever breaking backward compatibility for client applications and integrated systems.

Native client applications are configured to connect to the URL of either a reverse proxy in the web app tier, or directly to the URL of a web service in the web service tier. The endpoint URL's used by native applications are stored in app.config files and in system list files. Web applications store the endpoint URL of web services in their web.config file. Web services store ADO.NET connection strings, system account info, encryption key info and file storage paths in their web.config files. Many of these values were created using the DB Setup utility when a system was first set up, and saved securely afterward.

The content of the web service web.config files can be encrypted using the aspnet_regiis.exe utility in the locations of selected environments. This utility can also be used to encrypt the app.config files of native apps whenever appropriate.

Web App and Web Service Configuration

DGPWebApp Web.config Keys

Web.Config Key	Sample Value	Description
LocState	ONLINE	Current state of the web app reverse proxy, which can be used to effectively disable a web service for new applications calling the Login method
SvcURL	http://localost/DGPWebSvc	The URL of the DGPWebSvc used by the web applications and reverse proxy pages (if applicable)

DGPWebSvc Web.config Keys

Web.Config Key	Sample Value	Description
SvcKeyVersion	SvcKeyV1	The label of the current encryption key
SvcKeyV1	(32-byte encryption key)	The value of the specified encryption key (maintains a list of current and all previous encryption keys)
LocState	ONLINE	Current state of the web service, which can be used to effectively disable a web service for new applications calling the Login method
System	DGP	The name of the system that owns the web service
Environment	Dev	The environment that owns the web service
Location	Win10Dev	The location that owns the web service
WebSvcName	DGPWebSvc	The name of the web service to be returned by the Login method
WebSvcVersion	2020-11-22	The date string of the web service version to be returned by the Login method
EventSource	.NET Runtime	The name of the event source to be used to log entries to the Event Viewer (the default value works if no custom event source has been created
EventID	1000	The event ID that works with the event source, when an event ID value is needed

TTLCheckFlag	ON	Turns TTL check on or off in the message processing pipeline
TTLMS	10000	The maximum MS allowed for the TTL check
UserCacheFlag	ON	Turns caching of the UserInfo object on or off in the message processing pipeline
UserCacheSec	600	How long the UserInfo object can be cached until it becomes obsolete and is removed
RateLimitFlag	OFF	Turns the rate limit check on or off
MaxMethBatch	10	Sets the maximum number of methods in a single API request message allowed by the message processing pipeline
MaxReqMsgKB	64	Sets the maximum size of an API request message allowed by the message processing pipeline
MaxRespMsgKB	64	Sets the maximum size of a response message allowed by the message processing pipeline
MaxFailedLogin	5	Sets the maximum number of failed authentications allowed by the message processing pipeline
PasswordLength	8	Sets the minimum password length allowed for password resets
ExpireDays	90	Sets the number of days until a password expires
MaxClaimBatch	5	Sets the maximum number of records that can be claimed by the AutoWork test harness
MaxFileSize	10000000	Sets the maximum size of a file in bytes that is allowed to be stored in Lattice
MaxSegSize	45000	Sets the maximum size of a file segment when uploading and downloading a file in Lattice
MaxFavorites	100	Sets the maximum number of favorites that are allowed per user in Lattice
SysInfo	ADO.NET connection string	The ADO.NET connection string for the SysInfo database
SysWork	ADO.NET connection string	The ADO.NET connection string for the SysWork database
SysMetrics	ADO.NET connection string	The ADO.NET connection string for the SysMetrics database
FileStore	ADO.NET connection string	The ADO.NET connection string for the FileStore database

FileShard1	ADO.NET connection string	The ADO.NET connection string for the FileShard1 database
TestDB1	ADO.NET connection string	The ADO.NET connection string for the TestDB1 database
TestDB2	ADO.NET connection string	The ADO.NET connection string for the TestDB2 database

Important Note: The aspnet_regiis.exe utility is used to encrypt to AppSettings section of the config files in QA and Prod locations to protect sensitive data such as connection strings, encryption keys, etc.

Refer to the Web App and Web Serice documentation under the Server Tier section for more information.

AutoWork App/Service

The DGP AutoWork application is a Windows Service that can also be run as a console app for debugging, etc. It is deployed like any of the other native applications, with the added step of registering it as a Windows service using InstallUtil.exe. Information about the use of InstallUtil.exe to register and unregister the .NET application can be found in Microsoft documentation.

DGPAutoWork App.config Keys

Web.Config Key	Sample Value	Description
SvcURL	http://localost/DGPWebSvc	The URL of the DGPWebSvc used by the AutoWork service to call the API methods for each type of work queue
SvcAcctName		The DGP system account to use when calling the web service API's
SvcAcctPword		The DGP system account password
ClaimID		The unique ID used by the AutoWork instance to claim queue records in the work tables
ReplicaWork	LocalHost	The network area where the AutoWork instance is deployed, and should only claim queue records for the same area
ReplicaWorkMS	Int	The number of MS between each scheduled ReplicaWork interval

ReplicaMaxBatch	Int	The maximum number of records to be claimed in each batch
GeneralWork	LocalHost	The network area where the AutoWork instance is deployed, and should only claim queue records for the same area
GeneralWorkMS	Int	The number of MS between each scheduled GeneralWork interval
GeneralMaxBatch	Int	The maximum number of records to be claimed in each batch
QueueCheck	LocalHost	Flag value that turns the QueueCheck timer functionality ON or OFF
QueueCheckMS	Int	The number of MS between each scheduled QueueCheck interval
ErrIntervalSec	LocalHost	The long error interval that slows down scheduled iterations without disabling the automated process

When a location is recovering after a period of downtime (monthly maintenance, for example), the configured automated processes will have a backlog of work to be done. Under those circumstances, each set of work records (not claimed records) will be the maximum batch size allowed, and the process state will be "WORKING". Once the process has worked through the backlog and has caught up with the leading edge of new data (set of work records less than the maximum batch size), the state will be set to "CURRENT".

Refer to the AutoWork documentation under the Client Tier section for instructions on how to use the application.

Lattice Application

The DGP Lattice application is a .NET WinForm native UI used to administer, configure and maintain a DGP system, and is optionally also a system for collaborating and sharing files similar to OneDrive or DropBox. The first step is to use the standard XCOPY deployment pattern for the application. Create a parent folder, and copy a DGP Lattice version folder under the parent. Create a shortcut to the DGPLattice.exe file in the version folder.

In addition, several other folders should exist below the root version folder, and must be created if they do not already exist. A folder named "Data" must exist as a subfolder of the version folder. The Data folder must itself have 3 subfolders, DownloadTemp,

UploadTemp, and TestFiles. The TestFiles folder will have multiple subfolders and contain all of the test files needed to test all of the API methods in the Lattice web service. In addition, a sample system list file named SysList.xml should also be under the Data folder.

The default values in the App.config file must be edited for the location and network area in which they are deployed. The SysList entries must be edited to match the endpoint information of the local system. Refer to the Lattice application documentation for more information about the system list files.

Field Name	Field Values	Description
Network	LOCALHOST, INTERNAL, DMZ, EXTERNAL, OFF	The network area where the AutoWork Test harness application is running
AutoWorkLogging	ON, OFF	A flag value to turn logging for AutoWork processes on or off
AutoWorkMaxDurMS	50	The max duration for the logic to claim and process queue records (used by the AutoWorkTester test harness)
EventSource	.NET Framework	The default event source to use for logging info to the Event Viewer if a custom event source has not been created
EventID	1000	The event ID to use when logging info to the default Event Viewer
LocFilePath		Default path to root folder containing work directories, test files, etc.
TestFilePath		The default path to the local folder storing test files
UploadWorkDir		The default path to the local upload working directory used by the optional FileStore application
DownloadWorkDir		The default path to the local download working directory used by the optional FileStore application

Refer to the Lattice documentation under the Client Tier section for instructions on how to use the application.