DGP Maintenance and Support

Distributed software systems require multiple different types of maintenance and support work, which can be divided into three broad categories: code, infrastructure, and users. DGP code maintenance can be further broken down into client application code maintenance and server code maintenance.

- Code Maintenance
    - Client applications (thin UIs, integrated systems)
    - Web service APIs and code libraries (server)
    - Database schemas and core data (server)
- Infrastructure Maintenance
    - Application workstations and devices (OS patching, endpoint security)
    - App servers (web server, database server, etc. patching and maintenance)
    - Host servers (OS patching, endpoint security, monitoring)
    - Network (firewalls, load balancers, proxies, etc. updates and configuration)
- User Support
    - Account activation/deactivation
    - Account security maintenance (password reset help, authorization updates, etc.)
    - Documentation
    - Training

The process used to build a DGP system is different compared to conventional software systems. The best way to build most new DGP systems is by appending new APIs and new “spokes” of client apps and data storage to an existing DGP system, reusing all of its security, testing and other functionality as needed. This is easily accomplished by using DGP’s system evolution capabilities. The immutable append-only conventions then eliminate traditional (server tier) code maintenance in favor of appending new APIs and associated spokes over time in order to fix problems, improve existing functionality and add new capabilities over the life of the system. Thin client applications are independent, evolve in their own ways, on their own schedules, and are largely unaffected by changes to the APIs they depend on since all changes are guaranteed not to break backward compatibility.

At a high level, DGP's hub-and-spoke architecture behaves as a two-tier thin client/fat server, in which the client applications and server APIs are independent of each other thanks to the generic structured text API messages that they use for communication. The main requirement for client applications is that they be able to create API request messages and read API response messages, both of which are nothing more than structured text (XML fragments). The independence of the client applications from the server tier means that they can be built using any desired combination of operating system and programming language. It also means that maintenance of client application code is totally independent of the server code maintenance, and each can be done on their own independent schedules and timelines. This is why it is so important that the APIs never break backward compatibility.

The fat server tiers contain all of the logic and data in a DGP system. DGP's existing code (API methods), shared libraries, test files, documentation pages, database schemas and core data all become immutable once they are used in production, and afterward require no traditional maintenance work (since they never change) – only the work of adding new APIs, methods, libraries and data storage schemas, etc. to the server tiers of the system. Reducing the code maintenance workload to only appending new or modified functionality to a system correspondingly reduces the size of the team needed for software development and maintenance work.

Server Code Maintenance

The process to build a new DGP system and maintain that system over time are exactly the same. DGP server code maintenance work consists of appending additions to the source code of a system in order to add new features, improve existing functionality, fix problems, and so on. Most people do not realize that this process of improving and extending the code of a successful software system basically never ends, and generally will continue for as long as the system remains in active use.

Web Service APIs, Libraries, Schemas and Core Data

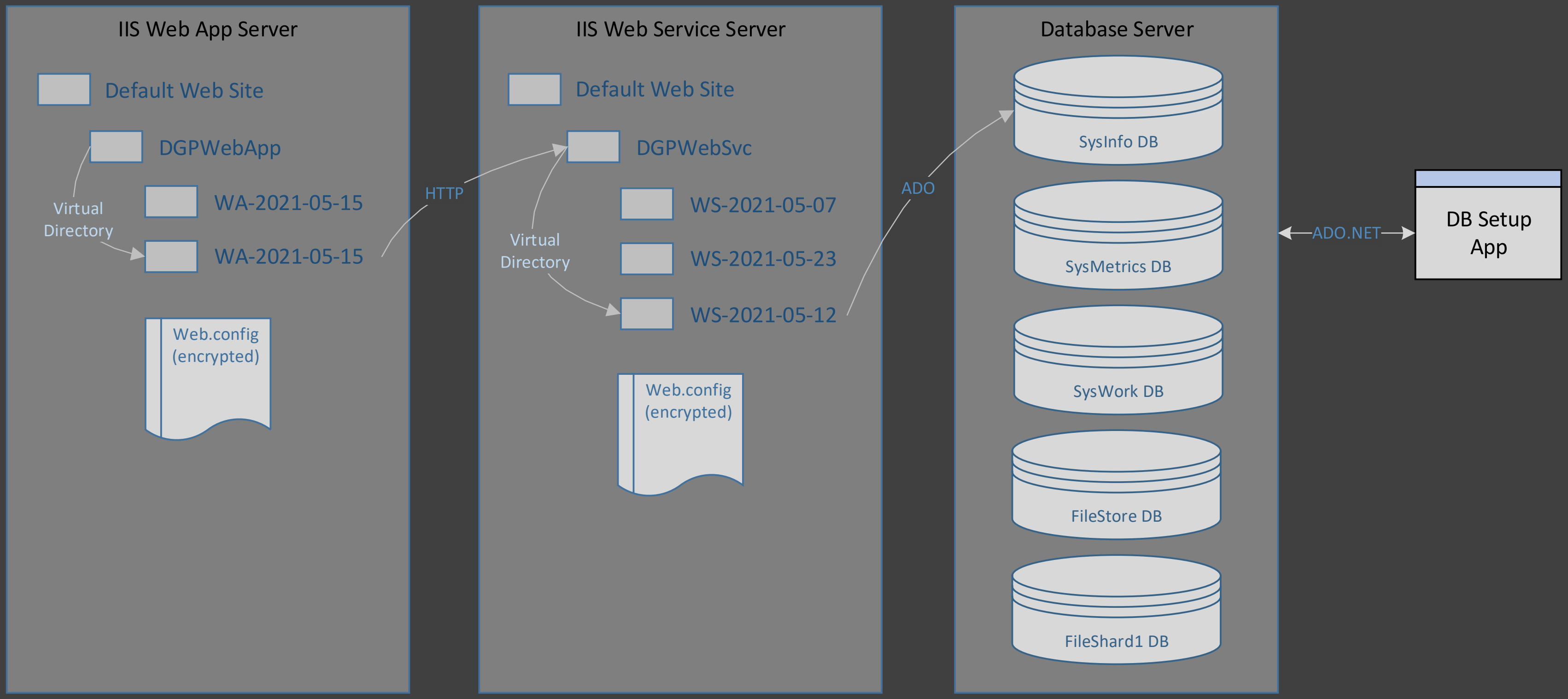
DGP has been designed in multiple ways to eliminate code maintenance work within a software system, and make other necessary types of maintenance as easy as possible, with no maintenance window planned downtime for the system as a whole (100% system uptime). The capabilities used to achieve 100% uptime in practice are a combination of hardware and software redundancy plus the "immutable append-only" convention. This combined pattern is used at many different levels throughout the architecture, design and deployment of DGP systems. One of the reasons it works so well is because it eliminates the problems caused to a software system whenever changes/additions are merged into existing code, data, and infrastructure. A side-effect of the immutable append-only convention used to achieve 100% system uptime is zero code maintenance.

Instead of merging changes into existing code, every current version of code is preserved unchanged, while modifications and/or new functionality are added (appended) to the system alongside them. The only exceptions to this rule are bug fixes to existing methods. The overall effects of these immutable append-only conventions on software systems changes the way that they are built and deployed. Every new method can be considered to be its own miniature experiment, and can be easily rolled back to its previous “known-good” version using security system authorization whenever one of the experiments fails. This helps to significantly improve the overall quality and reliability of DGP systems. As a result, the only development work for a DGP system consists of the creation of new extensions and improvements, along with an occasional fix for any problems that make it past testing. Also, by following this convention each existing method becomes more and more “hardened” and reliable (well tested) over time through constant use. The importance of this practice for reducing the total cost of systems while also significantly improving their overall quality and reliability cannot be overstated. Few if any current software systems are designed and built this way.

As an example of the differences compared to standard practices, the fastest way to build a new DGP system is to append new APIs, new database spokes and new client application spokes to an existing DGP system (such as DGPDrive), thereby significantly speeding up development while reusing many of the DGPDrive APIs, UI applications, etc. This is a very different way to build software systems that delivers much better results, which are repeatedly proven by the constant tests and measurements of the deliverables.

Web Service and Database Deployment

Both the native client applications and the web service APIs for the DGPDrive prototype have been built using the full .NET Framework 4.8.1, which is considered to be part of Windows itself, and are also performed by following the immutable append-only convention for deployments. No installation of an application or web service is necessary. Deployment consists of copying and pasting a release folder to the storage of the destination computer.



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.

Thanks to the immutable append-only conventions which prevent breaking changes to backward compatibility, plus the ability to use the RBAC security system as a feature toggle mechanism, deployment of new versions of the web service APIs can be done incrementally on each server, in each location.

In other words, a DGP system tolerates the fact that the servers within a location, and between locations do not all have exactly the same version of the code, etc. New methods must be "switched on" in the security system of each environment after all the servers in each location have been upgraded. The DBSetup utility is used to run the idempotent processes that maintain the database schemas and core data in each environment. These processes only append new elements to a schema, and prevent the same action from being performed multiple times (idempotent processes).

The same immutable append-only deployment process is used for both the client app and the web service APIs. A parent directory represents the application or web service, and each release folder (named for the date it was created) is added as a new subfolder below the parent directory. Editing the shortcut or web app configuration to point to the path of the new executable file (a .NET assembly) allows the new version of the application to be run. If any problems occur, pointing the executable path back to the previous version folder, plus deleting the new folder of the failed version constitutes a "rollback".

Client Applications

DGPDrive client applications attempt to minimize code maintenance work through the modularity of user controls, plus individual screens within the user controls. However, this level of modularity is generally going to be too coarse-grained to be practical for use of the immutable append-only convention to eliminate code maintenance (as it does in the web service APIs, etc.).

Thin client applications are independent of the server tier, so they can be built with almost any desired programming language, and their code maintenance is free to follow whatever practices make sense, on their own timelines. The DGP web service APIs never break backward compatibility thanks to the immutable append-only convention. All new releases of the web service APIs contain all of the unchanged versions of the API methods of all previous releases. This allows client applications to evolve to use new APIs on their own timelines, decoupled from the release schedule of the web service APIs themselves.

Infrastructure Maintenance

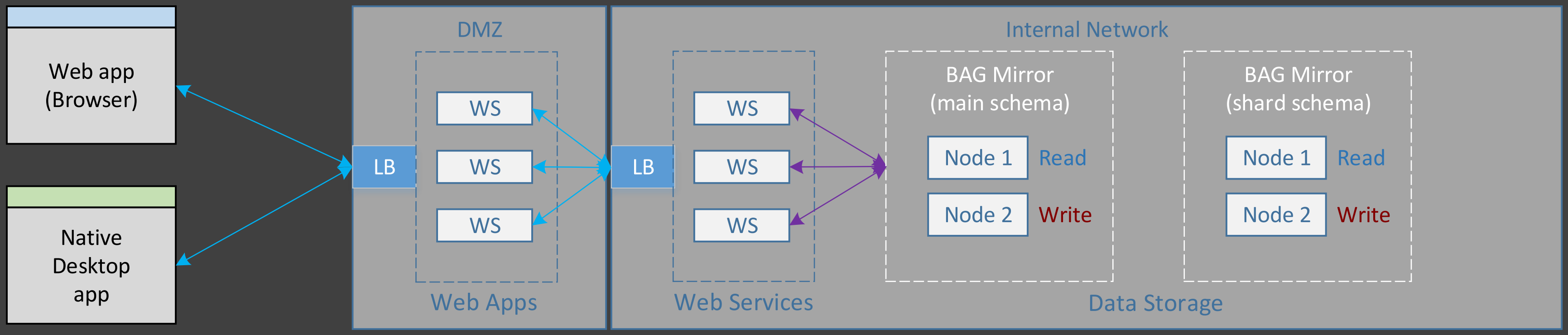
The other type of server tier maintenance work is to patch the infrastructure servers, operating systems, app servers, network appliances, etc. In addition, maintenance includes performing periodic database server maintenance, as well as fixing any problems with the servers of the distributed system. Non-redundant systems (single location) will require maintenance windows and periods of planned downtime to do so.

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. For systems with a single location, a maintenance window of planned downtime will be necessary.

The documentation for maintaining the production environment a DGP system assumes that 2 locations are up and running – a Primary location and a Backup location. All users connect to the Primary location, which results in a single writable data storage node (or cluster). This in turn provides strong data consistency, even though DGP is an "AP" system in terms of CAP theorem, focusing on 100% availability and partition tolerance (automatic recovery from server problems) at the price of eventual data consistency. This "hack" of CAP theorem used by DGP to deliver strong data consistency relegates the eventual consistency of the asynchronous data replication to only behaving as a continuous, incremental data backup mechanism running constantly in the background.

In this scenario, the entire Backup location is taken offline for maintenance, while everyone continues to use the Primary location. The offline status of the backup location pauses automated processes such as data replication. Once the Backup location maintenance is finished, it is brought back to an online status and the data replication is allowed some time to catch up with the work backlog. Once the data is in synch with the Primary, the status of the two locations are flipped. Changing this status (in the web.config files) will cause client applications to connect to the new Primary. Once that failover has occurred for all users, then the new Backup location is taken offline for maintenance, and the process is repeated.

The DGPAutoWork windows services have been designed to pause their own automated processes when a source or destination location has a status of "Maintenance" or "Offline". It will continue to check that status, and resume its work once the location status values return to Primary and Backup.



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

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. This option is not used when the Primary/Backup strategy is used for locations.

The FileStore application in DGPDrive uses the FileSeg schema with file storage shards, as shown at a large scale above. This demonstrates the use of shards to enable incremental horizontal scalability for the data storage tier. However, not all database schemas will be compatible with shards, so it is not a universally compatible solution. The DGPDrive file storage is compatible with shards, so it serves as a good reference implementation.

User Support

Adding user accounts to a system environment is initially performed using the DGPDrive admin tools. A super-admin account was created as part of the DBSetup utility process that populated the DGPDrive security databases for the first time. That account is then used to create admin user accounts for the DGP environment (each environment such as Dev, Test, QA and Prod all have their own security databases to manage their own user accounts, roles, membership, and so on). The admin accounts in each environment are used to create user accounts, assign role memberships that determine which API methods the account is authorized to call, and other support tasks such as resetting account passwords, recovering disabled accounts, etc.

Users are able to manage and reset their own passwords once their accounts have been created. Security logic in the front controller method of each web service will only authorize expired accounts to call their own password reset method, until their password is reset and is no longer expired.

Documentation of all of the tiers and applications of a DGPDrive system is also necessary. For DGPDrive, this documentation is provided in the form of a web application that is part of the open source DGPDrive system itself. Sections of the documentation contain user guides, setup and configuration instructions, original requirements and designs for the system, etc.

Windows Host Maintenance

Different environments for a system will have different configurations. For example, development computers will be single-server Windows 10 or 11 installations. The testing, QA and Production environments will be similar but should generally use Windows Server instead of Windows 10 or 11. Those desktop versions of Windows have a variety of scalability limitations that are not suitable for most production environments (only very small systems can work within those limitations).

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's closed-source products, and has no dependencies to any 3rd party tools (no dependencies to open source). What this means in practice is that there is therefore no requirement for software systems to scan the tree of open-source projects and their dependencies for malicious code.

DGP dependencies (Software Bill of Materials):

- Windows 10, 11, or Windows Server Standard (2016 or greater)
- Full .NET Framework 4.8.1 (part of Windows)
- IIS 10.x Web Server (part of Windows)
- SQL Server Express, or Standard (2016 or greater)

By only using (closed source) Microsoft products, this allows Windows updates to patch every tier and part of DGP systems in an automated, 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 so that SQL Server patches and updates are included. This allows for the unified patching of all tiers of a DGP system as part of the standard Windows update process.

Each development computer is its own separate, stand-alone DGP dev environment location. These development machines run Windows 10 or 11, and are the easiest to repair (or replace) if anything goes wrong with updates, especially if they are run as Hyper-V VM's. Automatic updates are fine for Windows 10/11 development computers. Each developer is responsible for Windows updates on their own development workstation.

The other environments will usually all use Windows Server, and updates should proceed sequentially from Test, to QA and then Production computers. Any update problems found for Test computers allows those problems to be resolved before they are applied to the QA environments, and so on. Since the QA computers are close to identical to Prod computers and usually share the same infrastructure, updates should work for the production computers with no problems once they are proven to work in QA.

Windows updates in general are very reliable. For a period of time there were internal problems at Microsoft that caused some problems with the update process, but those internal issues have largely been resolved. Also, Microsoft has since reestablished its emphasis on the Windows operating system, so Windows updates should continue to be very reliable going forward. This is an important advantage that Windows has relative to Linux, for example. 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, with no risk of “bricking” production servers.

The Test and QA environments can be updated at any time by coordinating the maintenance downtime with the testers. A redundant DGP system will usually have 2 locations, a primary and a backup. The Backup location can be taken offline for maintenance while the Primary location continues to function as normal. All nodes within the offline Backup location can then be taken offline to be patched, updated and maintained while all users continue using the Primary location with no interruptions.

When the maintenance for all servers is complete, the Backup location is brought back online, and allowed some time to synchronize data with the Primary. Once that data synchronization is complete, the Primary and Backup designations of the two locations can be reversed. This will gradually force client applications to switch from the old Primary to the new Primary location. Once that switch is complete, then the newly designated Backup location can be taken offline for maintenance, and the maintenance process repeated. Once that is completed, the Backup location is brought back online, but remains as the Backup going forward. In this way, the two locations alternate between Primary and Backup each time the maintenance process is performed.

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. 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.

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 types 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 10/11 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 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 (DGPDrive 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 ensure 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 DGPDrive 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 DGPDrive 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 many parts of DGP systems. All parts of DGP DGPDrive have been built with the full .NET Framework 4.8, and follow this same XCOPY immutable append-only deployment pattern. The directory structure below was created during the setup process:

1. On the computer used for Staging
    a. Create the subfolder DGP_Staging\DGP_date
    b. Download the .zip file from the documentation web app into the matching DGP_date subfolder
    c. 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
    d. Extract the contents of the zip file into the DGP_date subfolder which will create the following directory structure:

```
DGP_date
    DGP_assemblies_date
            DGP_Apps
                    dgp_autowork_date_asm
                    dgp_dbsetup_date_asm
                    dgp_DGPDrive_date_asm
            DGP_WebApp
```

```
                    dgp_webapp_date_asm
               DGP_WebSvc
                    dgp_websvc_date_asm
          DGP_source_date
```

2. On the computer used for Client Applications
    a. DGP_ClientApps\DGP_AutoWork
        i. Copy the assembly folder dgp_autowork_date_asm 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_asm subfolder
    b. DGP_ClientApps\DGP_DBSetup
        i. Copy the assembly folder dgp_dbsetup_date_asm 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_asm subfolder
    c. DGP_ClientApps\DGPDrive
        i. Copy the assembly folder dgp_DGPDrive_date_asm 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_asm subfolder

3. On the computer used for IIS Web Apps
    a. Inetpub\wwwroot\DGPWebApp

        i. Copy the assembly folder dgp_webapp_date_asm 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_asm subfolder
        iv. Rollback: edit the VM path to use the previous date_asm subfolder and delete the failed date_asm subfolder

4. On the computer used for IIS Web Services
    a. Inetpub\wwwroot\DGPWebSvc
        i. Copy the assembly folder dgp_websvc_date_asm 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_asm subfolder
        iv. Rollback: edit the VM path to use the previous date_asm subfolder and delete the failed date_asm subfolder

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 DGPDrive
MaxSegSize	45000	Sets the maximum size of a file segment when uploading and downloading a file in DGPDrive
MaxFavorites	100	Sets the maximum number of favorites that are allowed per user in DGPDrive
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.

DGPDrive Application

The DGPDrive 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 DGPDrive version folder under the parent. Create a shortcut to the DGPDrive.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 DGPDrive 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 DGPDrive 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