UPM Application Deployment Plan

Int.prd.001.01 | 10.08.24

UPM 24.x

ZINFI Confidential & Proprietary

Shared Under NDA

Contents

Introduction

We deploy our UPM Application to the MS Azure cloud environment for increased efficiency, agility, and scale. Our proposed architecture defines a fully managed platform for building, deploying, and scaling our UPM web application with Azure App Service and Azure SQL Database.

ZINFI's UPM Platform deployment to Microsoft Azure, the Cloud computing platform has been a groundbreaking technological advancement - transforming the business' use of technology with support for state-of-the-art operating systems, tools, databases, programming languages, and devices.

The move to Microsoft Azure allows ZINFI to address multiple enterprise requirements related to platform DevOps, performance, security, availability, disaster recovery and compliance, and to meet high-volume transactional demand that may vary from one season to another as channel programs and performance fluctuate. Additional capabilities that Microsoft Azure will enable on ZINFI's platform include:

- Improved DevOps: This allows ZINFI to collaborate with its customers for multiple instances—e.g., development, configuration, staging, pre-production and production—to create true enterprise-grade dynamic change management and version release capability.
- Higher performance: Vendors can now allocate database capacity and computational capacity to handle peak access needs from their partner base tied to end-of-quarter events and other seasonal requirements.
- Granular security: This includes advanced multi-layer protection from the web layer and application layer to the database layer to ensure access to encrypted data is managed both in motion and at rest.
- High availability: ZINFI UPM now has multiple failover modes to protect not only application availability, but also data scalability across the globe tied to high-usage demand mode.
- Advanced Disaster Recovery: This allows backup of the entire application and database to different datacenters. In the event of natural calamities or similar disasters, backup will instantly switch to alternate locations.

Deployment Components

Microsoft Azure is a fully managed computing platform optimized for hosting our web applications. Platform-as-a-service (PaaS) offering of Microsoft Azure lets us focus on the business logic while Azure takes care of the infrastructure and ongoing maintenance to run and scale our applications.

For the data tier of the UPM app, we chose Azure SQL Database for its dynamic scalability, built-in intelligence optimization, and global scalability and availability. We have leveraged Azure Database Migration Service to migrate our UPM Microsoft SQL Server Database to Azure SQL.

For the app tier, we chose Azure App Service, a PaaS service that enables us to deploy the UPM App with just a few configuration changes using Visual Studio.

Deployment Process

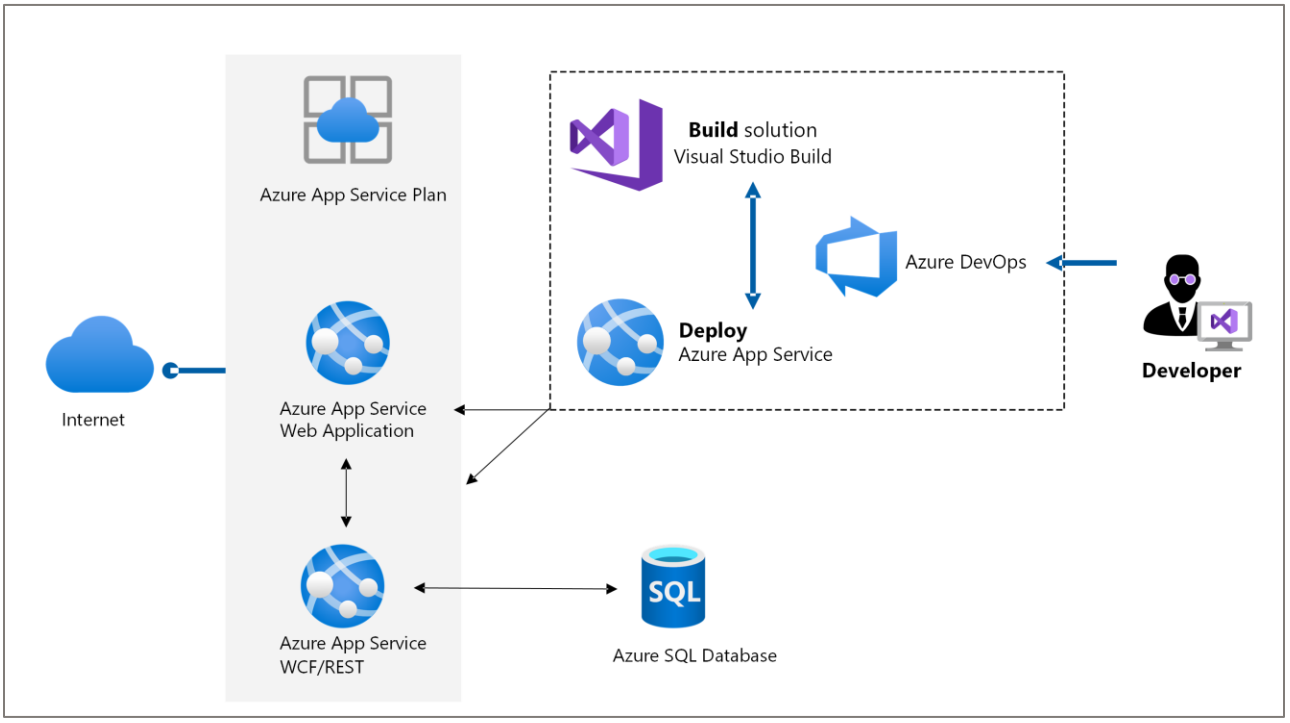
The Deployment Process is outlined through the primary processes as formulated below:

- Provision an Azure SQL Database instance in Azure. After the app website is migrated to Azure, the WCF/REST services app will point to this instance.
- Assess the database using Data Migration Assistant and migrate it using the Database Migration Service.
- Provision the two web apps.
- Set up Azure DevOps: create a new Azure DevOps project and import GitHub repo.
- Configure connection strings so that the UPM App and the SQL instance can communicate.
- Set up build and release pipelines to create the app and deploy to the web application.

Now, customers can connect to the UPM Web App. The load balancer automatically scales during periods of increased traffic to improve application uptime. Record sets are queried and pulled from Azure SQL Database.

- Customers connect to the UPM Web App
- Azure App Service provides security and automated management. DevOps capabilities are also possible such as continuous deployment from Azure DevOps.
- UPM Record-sets are queried and pulled from Azure SQL Database.

Deployment Architecture



UPM-Azure Base Deployment Infrastructure

The Base Infrastructure of the UPM deployment at MS Azure features the robust connection of the UPM Platform hosted at the Azure VM – Virtual Machine and Azure SQL Database utilizing App Service through a managed identity. Azure SQL Database is a fully managed platform as a service (PaaS) database engine with built-in capabilities:

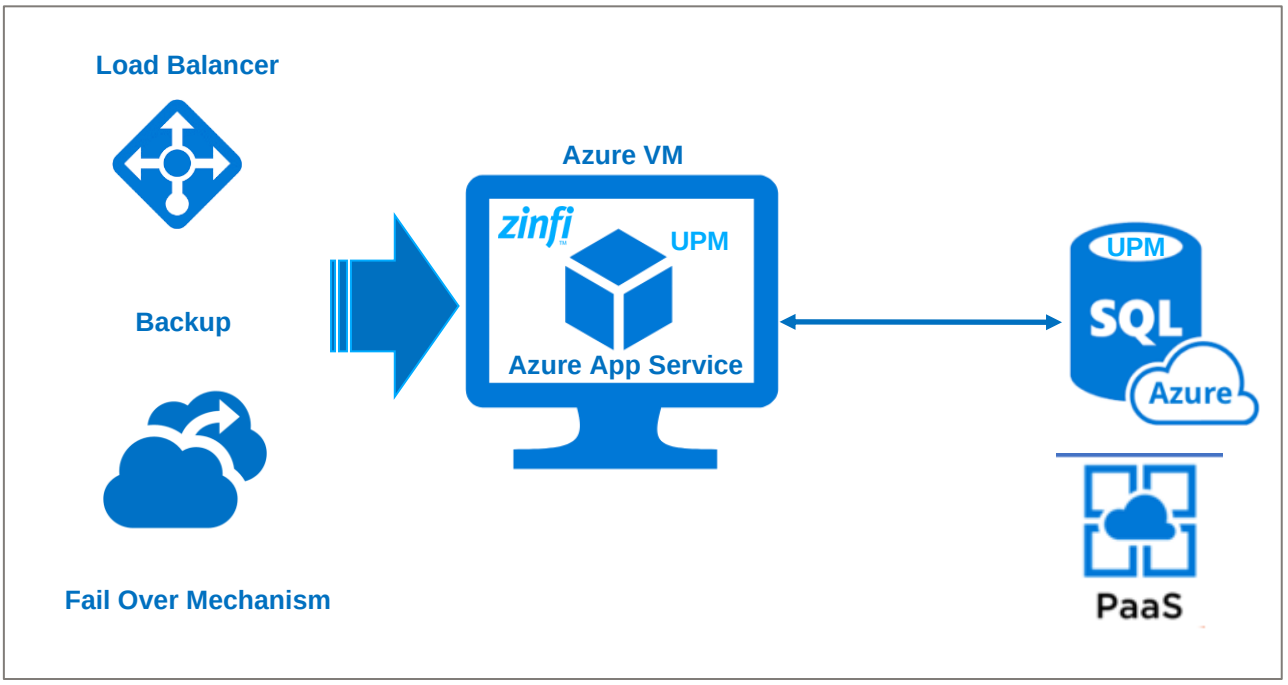
- **Load Balancer:** Azure SQL Database feature that allows us to create readable secondary databases of individual UPM databases on a MS SQL Database server in the same data center (region). Therefore, an online secondary can act as a load balancer for read workloads such as reporting.
- **Backup:** For each active database, the service maintains a backup chain that includes a weekly full backup, multiple daily differential backups, and transaction logs saved every 5 minutes. These blobs are replicated which guarantees that daily backups are available even after a massive failure in the primary region.
- **Fail Over:** Auto-failover groups is a SQL Database feature that allows us to manage replication and failover of a group of databases on a SQL Database server or all databases in a Managed Instance to another region.

Platform-as-a-Service (PaaS) provides a managed hosting environment, where we deploy UPM without needing to manage VMs or networking resources.

Azure App Service is a PaaS service providing a managed platform infrastructure for UPM. This means that Azure takes care of UPM application deployment and management, while the developer only needs to concentrate on app development.

We leverage **Azure Virtual Machines** to customize and control every aspect of the web server hosting the UPM Application.

Azure App Service is utilized to connect to Azure SQL Database - another PaaS offering where Azure provides SQL Database services service built using Database Availability Groups and the databases that we provision have High Availability built in. This allows us to deliver consistent services cost-effectively and without having to manage it.

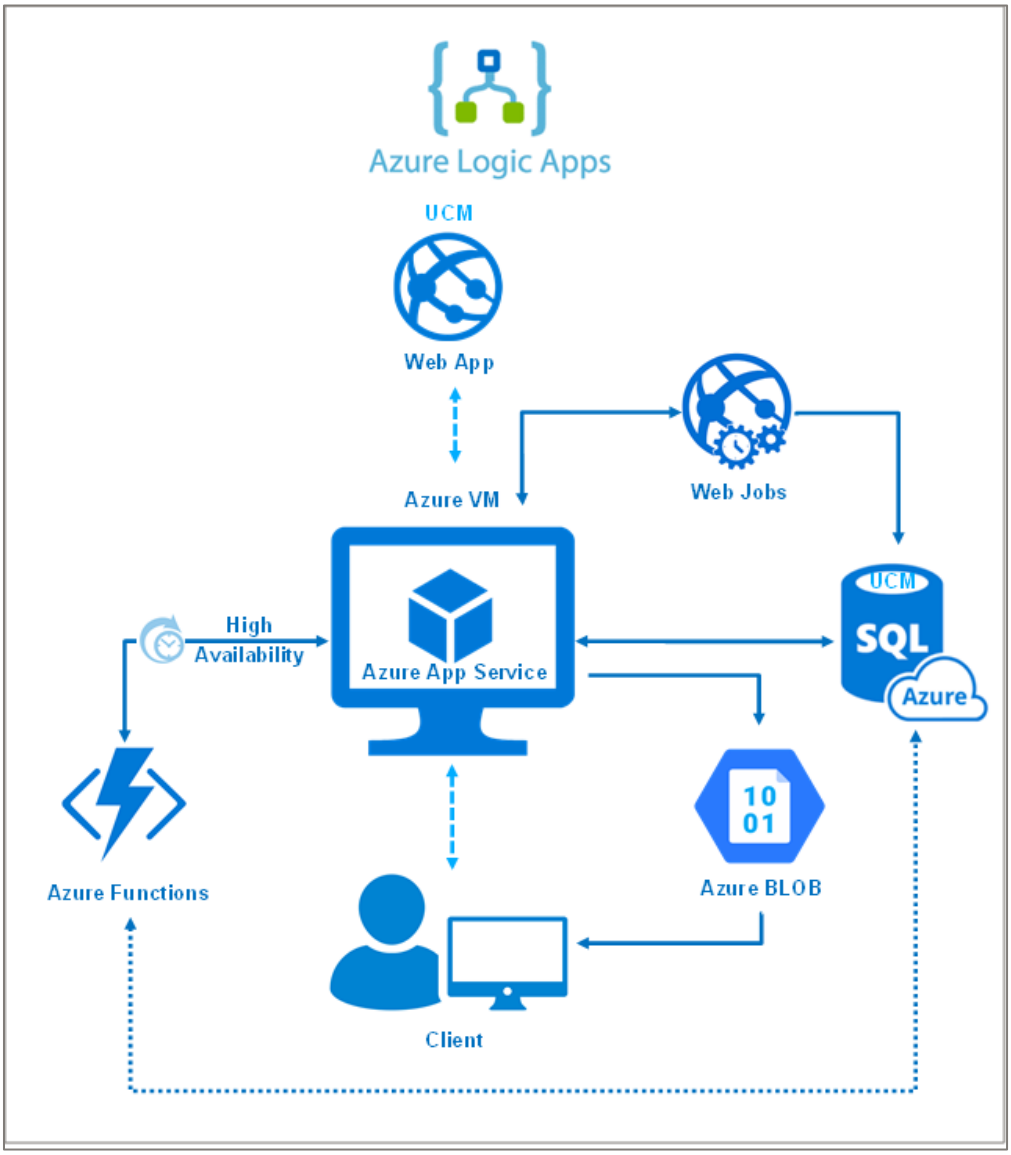


UPM-Azure Extended Deployment Infrastructure

The Extended Architecture of the UPM deployment at MS Azure extends the base Infrastructure to utilize the following Azure Services to provide the Client a Logical App:

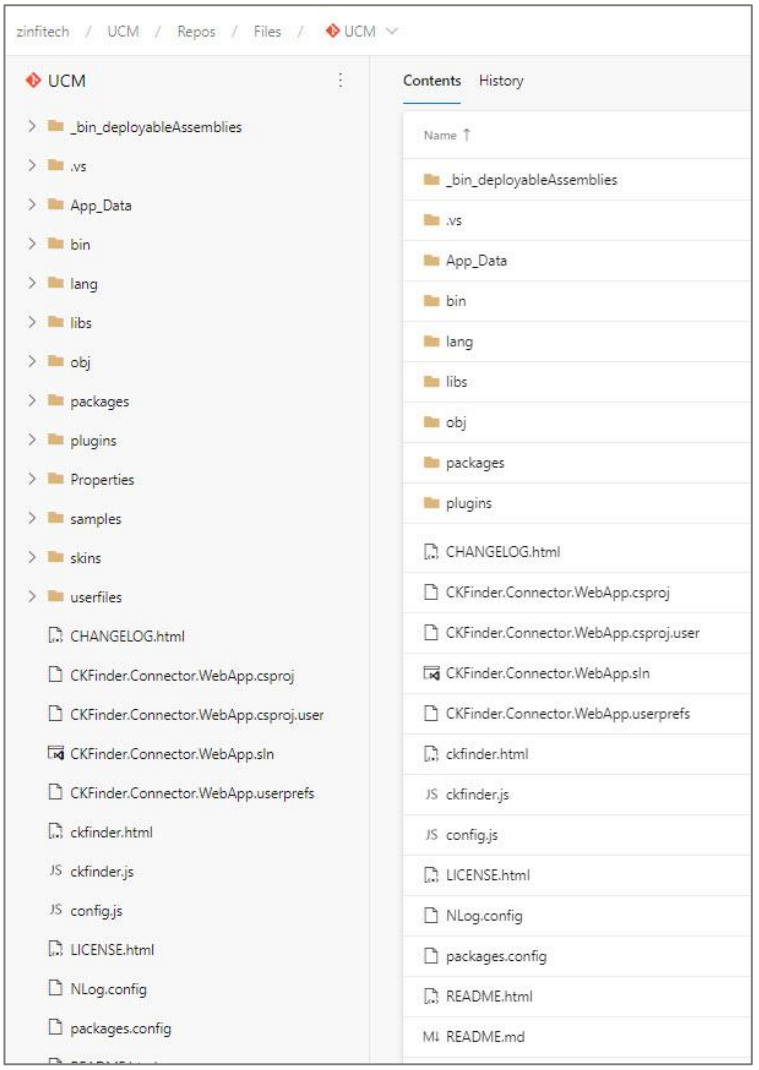
- Azure Functions
- Azure Web Jobs
- Azure BLOB
- Azure Web App

The Client accesses UPM - the Web App and respective App Services viz. Azure Functions, Web Jobs, BLOBs through the integrated Azure App Service hosted at Azure VM. The Client is provided Azure Logic Apps cloud services that helps the client to a faster UPM experience by scheduling, automating, and orchestrating tasks, business processes, and workflows when we need to integrate various apps, data, systems, and services across enterprises or organizations. Logic Apps simplifies how we design and build scalable solutions for app integration, data integration, system integration, enterprise application integration AI), and business-to-business (B2B) communication, in the cloud.



Deployment Directory Structure

The following illustration shows the enterprise deployment directory structure on the production server.



The following are utilized:

- For provisioning of the ASP.NET MVC UPM application and to deploy it to MS Azure, there are a set of assemblies we need to include with our application for it to run properly, unless they are already installed in the Global Assembly Cache (GAC) on the server. We utilize the _bin_deployableAssemblies folder to include third party assemblies (SQL Server) which are included in the bin folder on build and web deploy.
- .vs folder is required by Visual Studio deployments to store opened documents, breakpoints, and other information about state of our solution. which means It contains typical files like,
    - Temporary caches
    - IIS Express applicationHost.config file

- The intended use of App_data is to store application data for the web process to access. App_Data contains application data files including .mdf database files, XML files, and other data store files. The App_Data folder is used by ASP.NET to store the application's local database, such as the database for maintaining membership and role information.
- The bin folder holds binary files, which are the actual executable code for our application or library. The "bin" folder is the output folder for complete binaries (assemblies).
- LANG folder and files are used for localization. We localize our application by creating multiple, language-specific text files that map the programs' text strings to each language's version of that string. The program can then refer to the correct file when showing in-platform text, based on the user's language setting.
- Class libraries target a particular .NET Standard version. Platforms (e.g .NET Core, .NET (Full) Framework, Xamarin) implement particular .NET Standard versions. Our application targets platform versions and this determines which .NET standard libraries are available to the application.
- The "obj" folder is used to store temporary object files and other files used in order to create the final binary during the compilation process.
- The packages folder is for Nuget packages.
- The most important requirement of the plug-in architecture is that all plug-ins should exist as subfolders inside the plugins folder. Based on the customization requirements, plug-ins are automatically created inside this plugins folder. Each plug-in corresponds to a single customization task. A plug-in subfolder can contain a single or a combination of several files (such as configuration files, or ASP.NET files, etc.) that contain software code specific to the customization. This software code is then virtually merged into the appropriate file or files (such as configuration files, or ASP.NET files, etc.)
- The changelog summarizes updates for all versions of the Application.
- CKFinder files are utilized to integrate the CKFinder/CKEditor connector in the existing ASP.NET application.
- Config.js is a config utility for node.js, that uses a single JavaScript file with an export JavaScript object. After loading the JavaScript object from the configuration file, all properties are set constant, preventing changes. Config.js allows our developers to configure the applications in an XML block instead of hard-coding values inside their scripts or in JSON objects. The XML can be embedded inside an HTML document or in a separate XML file. The configuration block may contain strings, numbers, arrays and HTML.
- License.html is utilized to insert and manage licensing information.
- NLog is a flexible and free logging platform for the .NET platforms, including .NET standard. NLog makes it easy to write to several targets. (database, file, console) and change the logging configuration on-the-fly.
- The packages.config file is used in some project types to maintain the list of packages referenced by the project. This allows NuGet to easily restore the project's dependencies when the project is to be transported to a different machine, such as a build server, without all those packages.
- The README is a file that introduces and explains the project. It contains information that is commonly required to understand what the project is about.

Deployment Environments

A successful deployment relies on more than development of the application and the final packaging process. Our solution is tested in a controlled environment, and the deployment process often benefits from the use of staging servers. We manage the deployment of our solution between these environments, as well as to the final production environment. The following list briefly describes the characteristics of each environment, and explains how we use them to ensure a successful deployment of our solution:

Development Environment

This environment, as its name implies, is where our developers devote most of their coding and development effort. Because our developers require sophisticated development tools, such as Visual Studio .NET and various software development kits (SDKs), this environment does not usually resemble the one where the application will eventually be installed. However, the development environment does affect the deployment of the application. Our solution is typically compiled and built in this environment, as well as packaged for distribution to the other environments.

Test Environment

This environment is used to test the following:

- The actual deployment process, such as running Windows Installer files, copying, and configuring application files and resources, or other setup routines. Our test team needs to verify that application files and resources are installed to the correct locations, and that all configuration requirements are met, before testing the solution's functionality.
- After we are satisfied that the application installs correctly, our test team can then perform tests that verify the solution functions as expected.

To be certain that our tests have meaningful implications for how our solution will install and function in the live production environment, we ensure that the test environment resemble the production environment as closely as possible. They do not have the development tools installed that we used to produce the solution, because that could mask potential problems that will then only become apparent when we roll out our solution to the end users.

Staging Environment

This environment is used to house our solution after it has been fully tested in the test environment. It provides a convenient location from which to deploy to the final production environment. Because the staging environment is often used to perform final tests and checks on application functionality, it resembles the production environment as closely as possible. Usually, the staging network mimics the production environment in all respects, except that it is a scaled-down version (for example, it may have fewer cluster members or fewer processors than the server computers).

Production Environment

This is the “live” environment where our solution is put to work to provide benefits and functionality for the enterprise that uses them. This environment is undoubtedly the most complex. Consequently, it is the most difficult to manage and control; therefore, teams of administrators and information technology professionals are employed to ensure that users can take advantage of our application.

Process Model

At ZINFI, we have a dedicated process model with a composite support team of technical team, solution architects, solution managers, QA team, incident managers and IT team to traverse from production environment to staging environments. Highly iterative development based on AGILE process is performed on a Development Sandbox Environment with the requirements scope keeping in mind. Project Level Integration Testing is initiated on a Project Integration Sandbox which is the Sandbox for testing integration. With the completion of the Project Level Integration Testing, the UAT Sandbox and Pre-Production Environments are utilized for validating the release changes and to facilitate testing - including integrations - a controlled deployment approach, where Pre-Production Test/QA Sandbox and Demo Sandbox are provided for System and Acceptance Testing and product/services are confirmed. To integrate production support, we utilize a Hotfix Sandbox Environment for preparing and integration of hotfixes. A Performance Sandbox Environment is utilized for performing performance testing and finally, we move to a highly controlled deployment to a final Production Server.

UPM Server Environment Summary

Environment	Environment Summary
Dev	Sandbox for Release Development.
SIT	Sandbox for Testing Integration.
UAT	Sandbox used for validation release changes including integrations for the upcoming Release.
PreProd	Sandbox used for validation release (NRT) changes including integrations for the upcoming Release.
Hotfix	Sandbox used for preparing Hotfixes.
Performance	Sandbox used for performing Performance Testing.
Production	Live Environment for end-users.

