FDP Information System on Microsoft Azure

Overview

The Full Disclosure Project (FDP) information system has been implemented so that it can easily be hosted in the Microsoft Azure environment. For more information on Microsoft Azure, see: https://azure.microsoft.com/

This document is a work in progress. The configurations, steps and commands that it lists are intended only as demonstration; their implications, especially in regards to security and scalability, should be carefully considered.

Microsoft Azure Setup

The Microsoft Azure hosting environment is highly customizable, and so there are a range of environment configurations in which the FDP system can be hosted. For simplicity, the following setup is described; though, it is recommended that each project evaluates and selects the hosting environment configuration best suited to their specific needs:

- Run the web application as a *Web App* using an *Azure App Service*, and specify:
  - Runtime stack for Python 3.8
  - HTTPs only
  - Local Git deployment
  - Relevant settings as *Application Settings*
- Store data in an *Azure database for PostgreSQL* server, and specify:
  - Single server
  - Version 11
- Keep static and user-uploaded media files in an *Azure Storage* account, and specify:
  - One container called *static* with public access level: *Blob*
  - One container called *media* with public access level: *Private*
  - CORS rule for https://<web-app-name>.azurewebsites.net origin for GET method
- Store secret keys and passwords in *Azure Key Vault*, and specify:
  - Relevant key-value pairs as *Secrets*
  - Access policy for Azure App Service to *Get* any *Secret* and *List* all *Secrets*
- Enable *Identity Management* on the *Azure App Service*, and add:
  - *Storage Blob Data Contributor* role assignment for the *Azure Storage* account
  - *Reader* role assignment for the *Azure Key Vault* account

For more information on *Azure App Service*, see: https://azure.microsoft.com/en-ca/services/app-service/

For more information on *Azure database for PostgreSQL*, see: https://azure.microsoft.com/en-us/services/postgresql/

For more information on *Azure Storage*, see: https://azure.microsoft.com/en-us/services/storage/

For more information on how to *Authorize access to blobs and queues using Azure Active Directory*, see: https://docs.microsoft.com/en-us/azure/storage/common/storage-auth-aad

For more information on *Azure Key Vault*, see: https://azure.microsoft.com/en-ca/services/key-vault/

For more information on how to *Assign a Key Vault access policy using the Azure portal*, see: https://docs.microsoft.com/en-us/azure/key-vault/general/assign-access-policy-portal

Local Git Deployment

The FDP system can be deployed onto the Microsoft Azure environment using the local Git deployment method. The following steps are intended only as demonstration, and should be first evaluated.

1. Through the Azure portal, create the *Azure database for PostgreSQL*, the *Azure Storage* account and corresponding containers, the *Azure Key Vault*.

2. Download and install the Azure CLI: https://docs.microsoft.com/en-us/cli/azure/

3. Sign in with the Azure CLI using the **az login** command, see: https://docs.microsoft.com/en-us/cli/azure/reference-index?view=azure-cli-latest#az_login

4. Specify user credentials for a webapp deployment using the **az webapp deployment user set --user-name <user-name> --password <password>** command, see: https://docs.microsoft.com/en-us/cli/azure/webapp/deployment?view=azure-cli-latest

5. Create the webapp using the **az webapp create --name <webapp-name> --resource-group <resource-group-name> --plan <app-service-plan-name> --deployment-local-git --runtime "python|3.8"** command, see: https://docs.microsoft.com/en-us/cli/azure/webapp?view=azure-cli-latest

6. Note Git deployment URL returned by the command in step #4.

7. Add the desired Secrets to the *Azure Key Vault*, see: https://docs.microsoft.com/en-us/azure/key-vault/secrets/about-secrets

8. Set HTTPs Only for the *Azure App Service*, see: https://docs.microsoft.com/en-us/azure/app-service/configure-ssl-bindings#enforce-https

9. Enable system managed identity for the *Azure App Service*, see: https://docs.microsoft.com/en-us/azure/app-service/overview-managed-identity?tabs=dotnet

10. Through system managed identity for the *Azure App Service*, add role assignment to *Azure Storage* account as *Storage Blob Data Contributor*, see: https://docs.microsoft.com/en-us/azure/storage/common/storage-auth-aad-rbac-portal

11. Through system managed identity for the *Azure App Service*, add role assignment to *Azure Key Vault* as *Reader*, see: https://docs.microsoft.com/en-us/azure/app-service/app-service-key-vault-references

12. Add an access policy for the *Azure Key Vault* that allows the principal *Azure App Service*, identified through its identity object ID, permission to Get and List Secrets, see: https://docs.microsoft.com/en-us/azure/key-vault/general/assign-access-policy-portal

13. Add the desired Application Settings to the *Azure App Service*, see: https://docs.microsoft.com/en-us/azure/app-service/configure-common

14. For the *FDP_AZURE_STORAGE_ACCOUNT_KEY* Application Setting, use the reference to the corresponding Azure Key Vault Secret in the form of **@Microsoft.KeyVault(SecretUri=https://<key-vault-name>.vault.azure.net/secrets/<secret-name>/<secret-version>)** reference function, see: https://docs.microsoft.com/en-us/azure/app-service/app-service-key-vault-references#reference-syntax

15. To specify the Kudu build version used by Oryx, the key *KUDU_BUILD_VERSION* with the value *1.0.0* can be added as an Application Setting for the *Azure App Service*, see: https://github.com/Azure-App-Service/KuduLite/wiki/Python-Build-Changes

16. Add CORS rule for *Azure Storage* account, so that web application on *Azure App Service* can access it via the GET method, see: https://docs.microsoft.com/en-us/rest/api/storageservices/cross-origin-resource-sharing--cors--support-for-the-azure-storage-services

17. Create a new local Git repository with the code
    a. Either clone an existing repository using the **git clone** command, or
    b. Initialize a repository using the **git init**, **git add .** and **git commit -m "…"** commands.

18. Using the URL noted in step #5, add the remote repository using the **git remote add azure <git-deployment-url>** command.

19. Upload local repository to MS Azure using the **git push azure master** command.

20. If Django database migrations are pending, log in to the Web SSH (the virtual environment **WILL** be activated), see: https://docs.microsoft.com/en-us/azure/app-service/configure-linux-open-ssh-session
    a. Go to the web application's directory using the **cd $APP_PATH** command.
    b. If necessary, generate the Django database migration files using the **python manage.py makemigrations** command.
    c. Run the Django database migrations using the **python manage.py migrate** command.
    d. If necessary, create a Django superuser using the **python manage.py createsuperuser** command.

Legacy Kudu Build Version

Prior to March 1, 2021, a legacy Kudu build version was used by Microsoft during deployment. To revert to this legacy version during a deployment, the key *KUDU_BUILD_VERSION* with the value *0.0.1* can be added as an Application Setting for the *Azure App Service*.

If, after deployment with the legacy version, there are Django database migrations pending, then log in to the Web SSH (the virtual environment **WILL NOT** be activated), and:

1. Go to the web application's directory using the **cd $APP_PATH** command.

2. If necessary, generate the Django database migration files using the **/antenv/bin/python manage.py makemigrations** command.

3. Run the Django database migrations using the **/antenv/bin/python manage.py migrate** command.
    a. If necessary, create a Django superuser using the **/antenv/bin/python manage.py createsuperuser** command.

For more information, see: https://github.com/Azure-App-Service/KuduLite/wiki/Python-Build-Changes

PostgreSQL Setup

The database for the FDP system can be created on the Microsoft Azure environment using the *Azure database for PostgreSQL*. The following steps are intended only as demonstration, and should be first evaluated.

1. Create the PostgreSQL server through the Azure portal.

2. Note the admin username and password to access the PostgreSQL server.

3. Connect to the database using the **psql** command, and the username and password noted in step #2. An example may be:
    a. **psql "host=<azure-postgres-server-name>.postgres.database.azure.com port=5432 dbname=postgres user=<postgres-user-name>@<azure-postgres-server-name> password=<postgres-password> sslmode=require"**

4. Create the database using the **CREATE DATABASE fdp;** command.

5. Set the encoding for the user using the **ALTER ROLE <postgres-user-name> SET client_encoding TO 'utf8';** command.

6. Set the transaction isolation using the **ALTER ROLE <postgres-user-name> SET default_transaction_isolation TO 'read committed';** command.

7. Set the timezone using the **ALTER ROLE <postgres-user-name> SET timezone TO 'UTC';** command.

8. Ensure user has privileges to apply database migrations, using the **GRANT ALL PRIVILEGES ON DATABASE fdp TO <postgres-user-name>;** command.