**SOFTWARE ENGINEERING**

* Module: A file containing Python code
* Package: A collection of modules
* Library: A collection of packages
* Interpreter: A program that translates and executes code line-by-line
* Compiler: A program that translates code into an executable file, all at once, before the program Is executed
* Runtime: The phase when the program is running (after being compiled/interpreted)
* Runtime Environment: A collection of tools and resources needed (libraries, APIs, memory management) for the program to run properly
* URI (Uniform Resource Identifier): Identifies a specific resource on the web
* URL (Uniform Resource Locator): The address to the resource

Parts of a URL: <https://localhost:8000/path/to/resource?parameter1=par1&parameter2=par2#anchor>

* + Scheme (HTTP/HTTPS): The protocol used to access the resource
  + Domain ("localhost"): The server hosting the resource
  + Port ("8000"): The protocol port to which to send the request to access the resource
  + Path ("path/to/resource"): The path to the resource on the server
  + Parameters ("parameter1=par1&parameter2=par2): (Optional) Extra information provided to the server parameter1=par1&parameter2=par2
  + Anchor/Fragment ("anchor"): A specific part inside the resource
* API (Application Programming Interface): A mechanism that enables applications to communicate with each other via definitions and protocols
* API Endpoint: A specific URL to a resource
* API Router: An object that allows to define endpoints and apply common configurations such as path prefixes or dependencies
* API Request: An HTTP method sent by the client to the server (API endpoint) to request information or perform an action

HTTP methods:

* GET: Fetching data from the server
* POST: Submitting data to be processed
* PUT/PATCH: Updating data on the server
* DELETE: Deleting data on the server
* API Response: The response from the server
* API Payload: The data the client sends to/receives by the server
* API Gateway: A layer that sits between the frontend & backend, responsible for routing requests to the appropriate endpoint
* API Scopes: The specific types of access that the app is requesting from AAD (what actions it wants to perform) on behalf of a user during the authentication process (assigned in `Expose an API` in Enterprise App)
* API Permissions: The permissions that an application requests to access specific API resources on behalf of a user or itself (assigned in `API Permissions` in Enterprise App)
* User Permissions: The specific actions that a user can perform on resources (assigned on resource level using `Role-Based Access Control` (RBAC))
* API Authorisation: A layer of authenticating whether a user/resource is allowed to access the API’s resources
  + Basic Auth: You send your username & password with every request
  + API Key: A unique identifier passed along with API requests (API validates the key before processing the request)
  + Bearer Token: A temporary token to prove you are authorised
  + OAuth 2.0 (**O**pen **Auth**orisation): An authorisation framework that allows third-party applications to access resources or on behalf of users (without exposing user credentials)
    - Client app (frontend) is registered in AAD:
      * + Client ID: The unique identifier for the app
        + Client Secret: A password that is set up and used to authenticate the app
        + Define Redirect URIs: This is where the authentication responses should be sent
        + Define API permissions: These are the permissions the client app needs to access (e.g. openid, profile, User.Read)
        + Assign Users and Roles: Control which users can access the client application and what roles they have
    - API is registered in AAD:
      * + Define Redirect URIs (e.g. <https://oauth.pstmn.io/v1/vscode-callback>)
        + Define API scopes (i.e. **api://{API Client ID}/read**): The scopes/resources that the client application can request
        + Define Authorized Client applications (i.e. the frontend’s Client ID). This pre-authorises the client app, so users are not asked to consent when the client app calls the API.

User-based access:

* + - 1. User logs in via frontend (using `PublicClientApplication` from the MSAL.js library)
      2. Frontend sends a request to AAD with the required scopes to authenticate (specified in the frontend’s configuration)  
         [Auth URL - [**https://login.microsoftonline.com/{tenant}/oauth2/v2.0/authorize**](https://login.microsoftonline.com/%7btenant%7d/oauth2/v2.0/authorize), where tenant is the TenantID (can be found in AAD)]
      3. If the user is authenticated (they have a role assigned to access the app), AAD redirects back to the frontend with an authorization code (*Authorization Code Flow with PKCE*) via the Callback URL/Redirect URI
      4. Frontend sends a request to AAD’s token endpoint including the authorization code and client id and exchanges the code with an OAuth access token (this is often a Bearer token which includes the granted scopes)  
         [Access Token URL - [**https://login.microsoftonline.com/{tenant}/oauth2/v2.0/token**](https://login.microsoftonline.com/%7btenant%7d/oauth2/v2.0/token)]

Note: In a SPAs, **client secrets are not included** in the token exchange request for security reasons (as they run on the client side and aren’t secure enough to store secrets). The ACF-PKCE was developed specifically for SPAs to ensure security without requiring a client secret.

* + - 1. The frontend uses the OAuth token to make authenticated requests to the backend API

App-based access:

1. Frontend (or another backend service) sends a request to AAD (Access Token URL) including the client ID, client secret and scopes
   * + 1. If authenticated, AAD responds with an OAuth token
       2. The token is then used to make authenticated requests to the backend API

* CORS (Cross-Origin Resource Sharing): A browser-level security mechanism that controls which domains can access the server (backend/API).  
  Even if the user/frontend is authenticated and has the right permissions (based on their token), the browser enforces CORS.
  + CORS origins: These are the URLs that are permitted to make requests to the server, i.e. the domains/URLs from which the frontend may be hosted
  + Preflight Request: Before making a request, the browser may send a preflight (OPTIONS) request to check if the server accepts the request method (GET, POST, …) and headers from a certain origin
  + Allowed Methods/Headers: These are the HTTP methods (GET, POST, …) and headers (e.g. Authorization) defined from the server that are allowed
* Pydantic: A Python data validation and parsing library
  + DTO (Data Transfer Object): Represents the data being transferred between different components of the app (e.g. frontend <==> backend <==> database)

Example: fetch a person’s data from an API or send it to a database and want to ensure that ''age'' is an integer

* + Base Model: A Python class representing the schema of the DTO
* ORM (Object-Relational-Mapping): A technique for connecting databases (tables) and objects (classes representing tables)
* SQLAlchemy: A Python library that enables database interaction in an ORM-based approach (can also use raw SQL queries for more complex operations)
  + (ORM) Model: A Python class representing the database table
  + Declarative Base: A base class (schema) to define models
  + Engine: The starting point of the SQLAlchemy application - it manages a pool of database connections and provides a high-level interface for executing SQL commands
  + Session: An instance of database interaction - each session object manages its own database connection
* PyODBC: A Python library for connecting to ODBC (Open Database Connectivity) databases (SQL Server, MySQL, Oracle) – an alternative to SQLAlchemy
* Alembic: A Python library that helps with database migration (used with SQLAlchemy), It allows to apply changes to database schemas (adding/altering/removing tables) in a consistent and version-controlled manner without manually writing SQL.
  + Migration: A schema change
  + Revision: A specific version of the schema change
  + Upgrade: Moving the schema forward to a newer version
  + Downgrade: Rolling back to a previous schema state

Steps

1. In the terminal run **alembic init alembic** to initialise alembic

🡺 This creates an ‘/alembic folder’ with the config file ‘alembic.ini’

2. Edit the ‘alembi.ini’ file to include the database connection URL: **sqlalchemy.url = …**

3. Make changes to your models

4. In the terminal run **alembic revision --autogenerate -m "comment"** (add a comment to explain the change)

🡺 This creates a file in ‘alembic/versions’

5. Double-check the file to ensure that the changes are correct (sometimes, for complex SQL operations it’s not 100% correct)

6. In the terminal run **alembic upgrade head** to apply the change to the database

7. To roll back the last migration, in the terminal run **alembic downgrade -1**

* Status Codes: Used to indicate the success/failure of an operation
  + 200 OK: The request was successful
  + 400 Bad Request: Error on the client-side (e.g. wrong syntax, missing/invalid arguments)
  + 401 Unauthorized: Authentication is required - the request has not been authenticated or the credentials are invalid
  + 403 Forbidden: The server understands the request but refuses to authorize it (often due to insufficient permissions)
  + 404 Not Found: The requested resource could not be found on the server
  + 500 Internal Server Error: Error on the server-side (e.g. bug, config, database connection)
  + 503 Service Unavailable: The server is temporarily unable to handle the request, often due to being overloaded or undergoing maintenance

**DATA ENGINEERING**

* Dataset: A set of data
* Database: A collection of datasets designed to support transactional processing
  + Relational (SQL) database: Stores structured data in tables that are interconnected
  + Non-Relational (NoSQL) database: A non-relational database stores unstructured & semi-structured data, i.e. in whatever format is best for the type of data being stored (key-value, document-oriented, graph-oriented)
* Schema: The blueprint for a database – defines the organisation of data within a database, including tables, fields, relationships, views and indexes
* DBMS (Database Management System): A system that allows to create and manage databases
  + Relational DBMS: MySQL, SQL Server, SQLite, PostgreSQL, Oracle
  + Non-Relational DBMS:
    - Document-based (e.g. JSON): Azure MongoDB, Amazon DocumentDB
    - Key-Value pairs (like a dictionary): Azure Cassandra, Amazon DynamoDB, Redis
    - Graph-based (Azure CosmosDB (based on Gremlin), Neo4j, Amazon Neptune)
* SQL (Structured Query Language): The language used to manage [**C**reate, **R**ead (SELECT), **U**pdate (INSERT/UPDATE), **D**elete (DELETE) – **CRUD**] and query data in a relational DBMS
* Data Lake: A centralised repository that stores large volumes of structured, semi-structured and unstructured **raw** data, designed for big data/advanced analytics, machine learning (Amazon S3, Azure Data Lake Storage, or Google Cloud Storage)
* Data Warehouse: A centralised repository that stores large volumes of structured data designed for analytical processing (Snowflake, Amazon Redshift, Google BigQuery, and Microsoft Azure Synapse Analytics)
* Big Data: Datasets that are too large/complex for traditional data processing
* Data Pipeline: A series of processes where the output of one element is the input of the next one, designed to move data from one system to another
* Data Lineage: Tracking how data flows across the pipeline (from initial collection to final analysis)
* Data Modelling: The process of designing the structure of a database/warehouse to represent data entities, attributes, relationships and constraints
* Data Architecture: The design and organisation of data assets and resources within an organisation
* ETL (Extract-Transform-Load): the process of aggregating data from various data sources (databases, APIs), cleaning, formatting it and then uploading to target system (database/data warehouse)
* Data migration/ingestion: A one-time/continuous process of transferring raw data between systems
* Batch Processing: Processing of static data (e.g. reading a CSV file)
* Streaming Processing: Processing of real-time data streams (e.g. reading from an API)
* Normalization: The process of organizing data in a database to reduce redundancy and improve data integrity - involves **dividing large tables into smaller tables and defining relationships between them**
* Denormalization: The process of **combining tables to reduce the complexity of database queries** (often at the cost of data redundancy) - typically used in read-heavy applications
* Primary Key (PK): A column representing a unique identifier for each table record – it must contain **unique** values and **cannot be null**
* Composite Key (CK): A PK composed of two or more columns – it uniquely identifies records based on the **combination of these columns**
* Unique Key: A non-PK column that contains **unique values** – can contain a **single null value**
* Candidate Key: A **potential PK/CK**
* Alternate Key: A Candidate Key that was **not selected as the PK/CK** – remains a Unique Key
* Foreign Key (FK): A column (or group of columns) in one table that **refers to the PK in another table** – it establishes a **relationship between the two tables** and **can contain duplicate values**
* Natural Key: A Unique Key that is **meaningful in the real world**
* Business/Domain Key: A Unique Key that is **meaningful within the business** – might not have a real-world context outside the business scope
* Surrogate Key: A system-generated unique identifier – often an **auto-incrementing integer with no business meaning**
* Index: A database object that speed up data retrieval operations on a table – It creates a data structure for **quicker searches**
  + Clustered Index: **Physically arranges rows based on key values** - there can **only be one clustered index per table**
  + Non-clustered Index: Creates a **separate structure** for indexing columns, **pointing to the rows** - allows for **multiple non-clustered indexes per table**

Note: Avoid over-indexing as too many indexes can slow down INSERT, UPDATE and DELETE operations

* View: A **virtual table based on the result of a SELECT query** – can simplify complex queries and provide level of security by restricting access to specific data
* Transaction: A **sequence of one or more CRUD operations that are executed as a single unit**  
  Syntax: **BEGIN TRANSACTION [transaction\_name];  
   …**

**COMMIT;**

**ROLLBACK;**

* + For something to be a transaction, it needs to adhere to the ACID properties
    - Atomicity: A transaction must **succeed or fail as a whole** - either **all** operations are completed successfully, **or none** are applied
    - Consistency: After the transaction is completed, the **database is left in a valid and consistent state**
    - Isolation: The operations performed by the transaction **don’t impact other transactions, and other operations don’t impact this transaction**
    - Durability: The **result of the transaction is stored in the database** and won’t be lost if the database crashes or fails
  + Reads:
    - Dirty Reads: A transaction reads **uncommitted data**
      * Can lead to inconsistencies if the other transaction is rolled back (unsuccessful)
    - Non-Repeatable Reads: A transaction reads the **same row** twice but with **different value each time**
      * Can lead to reading different values if another transaction completes between the two reads
    - Phantom Reads: A transaction performs the **same query twice** but with **different set of rows retrieved each time**
      * Can lead to reading different values if another transaction completes between the two reads
  + Lock: A mechanism that **prevents multiple users or processes from interfering with each other when trying to access or modify the same piece of data**
    - Shared Lock (S): Used for read operations - **allows other users to read but prevents modifications**
    - Exclusive Lock (X): Used for write operations – **prevents any other operation** on the locked resource
    - Update Lock (U): Prevents deadlocks during updates - applied during initial read before upgrading to an exclusive lock
    - Intent Locks: Used to signal an intent to lock at a finer level (e.g., row) while maintaining locks at higher levels (e.g., table).
    - Schema Locks: Protect schema-related operations (e.g., modifying table structure)
  + Deadlock: Occurs when two or more transactions are waiting for each other to release locks, but none of them can proceed, creating a circular dependency.

1. Deadlock Detection**:** SQL Server automatically detects deadlocks using a **deadlock monitor** that continuously checks for circular dependencies.
2. Deadlock Resolution**:** To resolve the deadlock, SQL Server **selects one of the processes as the victim and terminates it**. The terminated process receives an error message, and its transaction is rolled back.
3. The other process is allowed to continue, preventing the deadlock.

To avoid deadlocks:

* + - Transactions access the tables in the same order
    - Transactions are kept short to reduce the time locks are held
    - Optimise queries (i.e. indexing) to reduce the need for locking large amounts of data
    - Specify locking behaviour (e.g. ROWLOCK, UPDLOCK)
  + Transaction Isolation Levels: Determines how transactions interact with each other, affecting concurrency and consistency. Isolation levels include:  
    Syntax: **SET TRANSACTION ISOLATION LEVEL [LEVEL];**
    - READ UNCOMMITTED (lowest level): A transaction **can read data that other transactions have modified but not yet committed**
      * Allows dirty reads
    - READ COMMITTED (default level): A transactions **can only read committed data** (prevents dirty reads)
      * Allows non-repeatable reads
    - REPEATABLE READ: Once a transaction reads data (via a query), **no other transaction can modify that data until the transaction completes** (prevents dirty and non-repeatable reads)
      * Allows phantom reads – other transactions might **insert** new rows that meet the query condition
    - SERIALIZABLE (highest level): Transactions are fully isolated from each other (it’s like they are executed sequentially). **No transactions can insert, update or delete rows that would affect the current transaction’s results** (prevents dirty, non-repeatable and phantom reads). The existing rows and the “range of values” that the query condition covers are **locked**.
      * Can lead to locking issues and reduced performance due to higher concurrency restrictions

A screenshot of a computer

Description automatically generated

* Common Table Expression (CTE): A **temporary, named result** that exists only within the scope of a single query and can be referenced within a CRUD operation – it **simplifies complex queries by breaking them into readable parts**
  + Recursive CTE: A type of **CTE that references itself** – useful for hierarchical data such as organizational structures
* Stored Procedure: A **precompiled collection of one or more SQL operations that can be executed as a single unit** - can accept parameters and improve performance by reducing the amount of code sent to the database
* Stored Functions: Similar to stored procedures but they **return a single value** - can be **used directly within SQL expressions**
* Trigger: A special type of **stored procedure that automatically executes in response to specific events on a particular table** (e.g., insert, update, delete)
* Join: A SQL operation that **combines rows from two or more tables based on a related column between them**
* Union: A SQL operation that **combines distinct rows from two or more SELECT statements**
* Aggregate Function: Computes a **single result from multiple rows** (e.g. MIN, MAX, COUNT, SUM, AVG, …)
* Grouping: Used to aggregate data based on the aggregate function
* Window Function: Similar to an aggregate function but does **not group** them into a single output (e.g. ROW\_NUMBER, RANK)
* Partitioning: A table is split into **smaller, more manageable physical storage units** (partitions) based on a specific criterion (e.g. date, range).  
  Each partition acts like a **subset of the table** which can be queried independently, optimising performance in large datasets.
  + Horizontal Partitioning: Divides rows across partitions (more common)
  + Vertical Partitioning: Divides columns across tables (less common)

**AZURE**

**Cloud Service Models**

* SaaS (Software-as-a-Service): A subscription-based service model which allows users to access software apps over the web
  + **Most complete** model with minimal user responsibility
  + Provider manages everything except data, user access, and device security
  + Ideal for fully developed applications like email or financial software
  + Easiest to use with minimal setup effort
* PaaS (Platform-as-a-Service): A platform to support the development and deployment of applications (e.g. Azure App Service)
  + Balances flexibility and convenience
  + Provider maintains hardware, OS, middleware, and development tools
  + **You focus** on application development without infrastructure headaches
  + Suitable for building and deploying applications efficiently
* IaaS (Infrastructure-as-a-Service): A cloud computing service model which allows to use cloud services by a provider (e.g. Azure)
  + **Most flexible** cloud model
  + Cloud provider handles hardware, network connectivity, and physical security
  + **You manage** OS, configuration, patches, and security
  + Offers maximum control but requires technical expertise

**Compute Services**

* Azure Virtual Machine (AVM): A virtualised server (computer) in the cloud with full control over the OS (Windows/Linux) and allows for software deployment and specialised hosting configurations
* Azure Virtual Desktop (AVD):
* Azure App Service: A platform to host web apps, APIs, or mobile backends without managing the underlying servers
  + Languages Supported: .NET, .NET Core, Java, Ruby, Node.js, PHP, Python
  + Platforms Supported: Windows and Linux
  + Integration: Automated deployments from GitHub, Azure DevOps, or any Git repo
  + Scaling: Automatic scaling to handle variable traffic loads
  + High Availability: Built-in load balancing and traffic management
* Azure Functions:A serverless event-driven compute service that automatically manages the underlying infrastructure, allowing to focus solely on code
  + Event-Driven Execution: Functions are triggered by specific events, such as HTTP requests, timers, or messages from Azure services
  + Serverless Architecture: No need to maintain servers or containers - resources are provisioned and deallocated automatically
  + Stateless vs Stateful: Functions can restart for every event (stateless) or implemented during *durable functions*, enabling tracking of prior activities or workflows (stateful)
  + Pay-As-You-Go: Billed only for the CPU time and memory used while the function runs, making it a cost-effective solution
  + Automatic Scaling: Functions scale dynamically based on the number of incoming events, handling variable demand with ease
  + Flexible Deployment: Functions can be deployed serverless or in managed environments like virtual networks for enhanced control and security

**Storage and Databases**

* Azure Blob Storage: A cloud-based hard drive for storing large amounts of unstructured data
* Azure SQL Database: A fully managed database service for storing and querying relational data

**Networking**

* Azure Virtual Network (VNet): A private network in the cloud where resources can securely communicate with each other
  + Secure Communication: Private traffic stays within the Microsoft network
  + Scalability: Easily scale resources while maintaining secure and efficient communication
  + Flexibility: Connect resources across VNets, regions, and hybrid setups
  + Control: Fine-grained control over traffic flow, routing, and filtering
* Azure Load Balancer: Distributes traffic across multiple VMs to ensure high availability
* Azure VPN Gateway: Securely connects your on-premises network to Azure over a VPN

**AI and Analytics**

* Azure Cognitive Services: Ready-to-use AI models for tasks like image recognition, natural language processing, or speech synthesis
* Azure Synapse Analytics: A big data analytics platform to process and analyse large datasets efficiently
* Azure Machine Learning: Tools to build, train, and deploy your machine learning models

**Identity and Security**

* Azure Active Directory (AAD): Handles user authentication and identity management across Azure and other apps
* Azure Key Vault: Securely store secrets, keys, and certificates for your applications
* Azure Security Centre: Helps monitor and protect your Azure resources from security threats

**Monitoring and Management**

* Azure Monitor: Provides insights into the performance and health of your Azure resources
* Azure Resource Manager (ARM): A management layer that lets you deploy, update, and manage your Azure resources in an organized way

**Containers**

* Azure Kubernetes Service (AKS): A managed service for running containers using Kubernetes, ideal for scaling applications
* Azure Container Instances (ACI): Quick and simple way to run containers without managing a server or orchestrator
* Management Group: A container that helps organise and manage multiple subscriptions
* Subscription: A container that linking Azure accounts and the resources created by those accounts
* Resource Group: A container that holds resources (web apps, databases, storage accounts, virtual machines, …) for an Azure solution
* Service: An offering (PaaS) such as:
  + Compute Service (Virtual Machines, Azure Kubernetes Service (AKS), Azure Functions)
  + App Service (Web Apps (with server-side logic), Static Web Apps (with static content or SPAs), Logic Apps)
  + Storage Service (Azure Blob Storage)
  + Database Service (Azure SQL Database, Azure Cosmos DB)
  + DevOps Service (Pipelines, Repos)
  + Security Service (Azure Active Directory, Azure Key Vault)
* Resource: An instance of a service, managed within a resource group – all resources can be found in `Resource Manager`.
  + Scalability: Ability to adjust resources to meet demand
    - Vertical scaling: Increasing/Decreasing the **capabilities** of resources (e.g. add more/lower CPUs or RAM to a virtual machine)
    - Horizontal scaling: Adding/Subtracting the **number** of resources
      * Scaling out: Add additional resources (e.g. VMs or containers) due to jump in demand
      * Scaling in: Remove extra resources due to drop in demand
  + Reliability: Ability to recover from failures and continue functioning
  + Predictability: Ensuring consistent performance (autoscaling, load balancing, high availability) and cost (real-time tracking and analytics or tools) expectations

* Azure Active Directory (AAD)/Microsoft Entra ID (MEID): Azure’s identity and access management service (i.e. authentication system)
  + Directory (Tenant) ID: The unique identifier for the tenant (i.e. organisation) in AAD
  + Object ID: The unique identifier for an object (user, group, service principal, app registration, enterprise applications) in AAD
  + SSO (Single Sign-On): Enables a user to access multiple applications or services with a single set of login credentials
  + Multifactor Authentication (MFA): Requires an additional verification factor beyond just a password
  + Passwordless Authentication: Replaces passwords with secure alternatives

* Azure Role-Based Access Control (RBAC): A system that helps manage access to Azure resources by assigning roles to users, groups or applications at specific scopes. RBAC follows the **principle of least privilege**, ensuring users only have the **minimum permissions required** to perform their tasks, thereby enhancing security.
  + Roles: Define a set of permissions for actions on Azure resources
    - Owner: Full access to all resources, including granting access to others
    - Contributor: Manage resources but cannot grant access.
    - Reader: View resources but cannot make changes.
    - Custom Roles: Created to meet specific access requirements.
  + Scopes: Define where the role applies
    - Management group: A collection of subscriptions
    - Subscription: An Azure subscription
    - Resource group: A logical container for related resources
    - Resource: A specific Azure resource like a storage account or VM
  + Role Assignments: Assign a role to a user, group or application at a specific scope

Permissions are **inherited** by child scoped. For example, assigning the ‘Owner’ role at the subscription level, gives **access to all resource groups and resources within that subscription**.

If multiple role assignments grant different levels of access to the same resource, the permissions are combined.

* App Registration: Registering an app in AAD, creating an identity and obtaining credentials that the app will use to authenticate itself when interacting with other resources
  + Client/Application ID: The unique identifier for the app
  + Client Secret: A password that the app uses to authenticate itself. This is manually created during registration.
  + Service Principal: An object representing the registered app (containing the client ID & client secret).  
    It is configured to access Azure resources (e.g. Azure DevOps) across tenants.
  + Managed Identity: A type of service principal but created and managed by Azure (**no app registration** is needed).  
    Used when resources need to access other resources directly (without credentials) in the same tenant.
    - System-assigned: Enabled directly on an Azure resource – tied to a **single** resource
    - User-assigned: Created as a standalone Azure resource – is assigned to **multiple** Azure resources
* Service Connection: A configuration that allows Azure DevOps (e.g. pipeline) to connect to different Azure services
  + Azure Resource Manager (ARM): Connects to Azure subscriptions and resource groups (i.e. access to multiple resources)
  + Docker Registry: Connects to Docker container registries (e.g. Azure Container Registry (ACR), Docker Hub)
  + GitHub: Connects to GitHub repositories
  + Azure Kubernetes Service (AKS): Connects to Azure Kubernetes Service clusters

To set up a service connection: Azure DevOps 🡪 (select a project) 🡪 Project settings (bottom left) 🡪 Service connections

* Enterprise Application: The representation of the app within the organisation, used to manage access and permissions within the organisation. It is automatically created when an app is registered to AAD.

**Azure Databricks**

A cloud-based platform designed for managing and analysing large datasets, built on Apache Spark

* Notebook: An interactive interface for development, collaboration and visualisation
* Clusters: Managed Spark clusters that can scale based on workload
* Standard Cluster: General-purpose for development and testing
* Interactive Cluster: Optimized for notebooks and exploratory analysis
* Job Custer: Created specifically to run jobs, terminated afterwards

Apache Spark: A distributed computing system optimized for big data processing. It is fast due to in-memory computing and scalable as it’s designed to run on clusters (multiple machines).

* + Job: A set of tasks/computations
  + Transformation: A **method on a dataframe** (e.g. filter, map, select, groupBy)
  + Action: A method which triggers the execution of transformations and **returns a dataframe** (e.g. show, collect, count)
  + Catalyst Optimizer: A Spark mechanism which **optimizes execution plans by combining transformations efficiently** (predicate pushdown, column pruning)
  + Lazy Evaluation: A Spark feature which holds off the execution of transformations until an action is called
* By not executing transformations immediately, we **avoid bringing the entire dataframe into memory immediately**, saving cluster capacity and improve computation performance.
* Due to lazy loading, we end up having a **bunch of transformations to run together** - the catalyst optimizer then **combines them in the most efficient way**.
* Spark Core: Manages distributed data processing
* Spark SQL: Handles structured data with SQL queries
* Spark Streaming: Processes real-time data streams
* Spark Cluster: A set of resources/VMs used to run the job – can be scaled up or down based on workload and consists of:
* Master node: The central coordinator of the cluster, managing resource allocation and task scheduling
* Driver node: Executes the Spark application
* Worker nodes (Executors): Perform the actual data processing
* Partition: A chunk of data
* Repartition: Increasing the number of partitions – expensive, avoids shuffling
* Coalesce: Reducing the number of partitions – cheaper, avoids shuffling
* PySpark: The Python API for Spark
  + RDD (Resilient Distributed Dataset): A data structure that represents a collection of elements, partitioned across the nodes of a clusted
    - Immutable, fault-tolerant collections of objects distributed across the cluster
    - No Schema
    - Manual optimization
  + DataFrame: A collection of data organised into named columns
    - Distributed processing across a cluster (vs Pandas’ in-memory processing)
    - Optimized for large datasets (vs Pandas’ for smaller datasets)
    - Lazy evaluation (vs Pandas' instant evaluation)

Databricks Workflow:

1. Spark cluster starts
2. Driver node starts to run the main script and splits it into stages (reading, cleansing, filter, grouping, ...)
3. A DAG (Directed Acyclic Graph) is created - this a logical representation of the order of stages
4. Data is split into partitions (chunks), enabling parallel processing - let's say it's 1000
5. For each stage, 1000 tasks (one per partition) are created to distribute data to the worker nodes
6. Executors perform tasks

**Azure Data Factory**

Used for creating ETL pipelines to move and transform data from various sources to destinations

**Azure Functions**

A serverless compute service that runs code in response to triggers without needing to manage infrastructure

* Function App: A container for the Azure Function(s)

**Setting up a Docker container**

1. Add your account as a ‘Docker user’
   * Go to ‘Computer Management’ 🡪 ‘Local Users and Groups’ 🡪 Groups
   * Click on ‘docker-users’
   * Click ‘Add’
   * Type ‘GHDNET\username’ 🡺 GHDNET\rpapadopoulos2
2. Open Command Line/Terminal as administrator
3. Open Docker Desktop as administrator.
4. If engine doesn't start:
   * Run "**wsl --unregister docker-desktop**" in terminal
   * Type "**services.msc**", find "Docker Desktop" and stop & rerun
5. Change to the directory where the app is located  
   (e.g. cd "C:\Users\rpapadopoulos2\OneDrive - GHD\Projects\Pipeline Cost Estimation Toolbox\PESTWebApp")
6. Run the command "**docker build -t X .**", where X is the app name

* docker build -t pest .

If you get an error:

* Run "**docker logout && docker login**" and login with your Docker credentials
* Rerun the 'build' command

1. Ensure you have **Contributor** access to the RG that contains the Container Registry
2. Run ‘**az login**’ and log in with the account that has Contributor access
3. Run the command "**az acr login -n Y**", where Y is the name of the Azure Container Registry

* az acr login -n ghdacrukspcet001

1. Run the command "**docker tag X Z/X**", where X is the app name and Z is the Login Server (can be found in 'Overview' tab in the ACR)

* docker tag pest ghdacrukspcet001.azurecr.io/pest

1. Run the command "**docker push Z/X**" (might have to run this a couple of times till everything is pushed)

* docker push ghdacrukspcet001.azurecr.io/pest

**GIS**

GIS (Geographic Information Systems): A system for capturing, storing, analysing and visualising geospatial (location-based) data

ArcGIS: A GIS software

* ArcGIS Pro: The desktop-based GIS software that provides advanced tools for 2D and 3D mapping, data visualization and spatial analysis

A diagram of a project

Description automatically generated

* ArcGIS Online: A cloud-based SaaS platform that allows users to create, share and manage GIS content (maps, apps, data) online
* ArcGIS Enterprise: The on-premises GIS solution that allows organisations to deploy GIS infrastructure (i.e. host maps, data and services) on their own servers
* ArcGIS Dashboard: A GIS software that allows you to create and share interactive dashboards that display data in real-time
* Information Products: Ways of conveying geospatial information
  + Static Map: A stationary graphic representation of the spatial relationships of entities within an area
  + Web Map: A 2D web-based interactive visualisation of geospatial content
  + Web Scene: A 3D web-based interactive visualisation of geospatial content
  + Time Map: A visualization of one or more temporal layers that represent changes over time
  + Animation: Transitions between keyframes (snapshot of data in a particular state)
    - Fly-through animation: Simulates a camera smoothly moving through a scene or map
    - Tour animation: Zooms into a location, zooms out, and then zooms back in to the next location
  + Chart: A graphic representation of tabular data
  + Story: An interactive web application that incorporates text, images, GIS information products, and other web content to provide a narrative context for your data

Shapefile: A collection of files representing geospatial data

* .shp (Shape format): Stores the actual **geometry of the features** (points, lines, polygons) – it is the core of the shapefile
* .shx (Shape index format): An **index** of the geometry in the .shp file, allowing for **faster access** to the features
* .dbf (Attribute format): Contains non-location (tabular) data (**attributes**) associated with each feature (e.g. name, population)
* .prj (Projection file): Defines the coordinate system and map projection used for the geospatial data

Coordinate System: Defines the position of features in a 2D or 3D space

* Geographic Coordinate System (GCS): Uses latitude and longitude (degrees)
* Projected Coordinate System (PCS): Transforms data onto a flat surface (2D maps)

Projection: Transformation from spherical coordinates to a flat (XY) coordinate system

Georeferencing: Aligning geospatial data to a known coordinate system so it can be viewed on a map

Geocoding: Transforming a description of a location (e.g. address, place name) into geographic coordinates

* Input Data: The location description provided by the user
* Locator: A tool that maps input data to geographic coordinates to place features on a map
  + Free locators: Open-sourced such as ArcGIS Online World Geocoding Service
  + Paid locators: Firewall-based such as ArcGIS World Geocoder and ArcGIS StreetMap Premium
  + Own: Built by yourself, with your own data

A **composite locator** combines two or more individual locators which are used in sequence to find a location.

* Reference Data: The data used by the locator to match the input data against such as addresses (streets, postcodes), points of interest (landmarks, natural features, businesses), populated places (cities, regions, countries)

Buffering: Creating zones around features (e.g., drawing a 1 km buffer around roads)

Intersect/Union: Spatial analysis techniques to combine or compare features of different layers

Feature: An object that stores its geographic representation (i.e. point, line, polygon)

Vector Data: Represents features with discrete boundaries like **points** (e.g. locations), **lines** (e.g. roads), and **polygons** (e.g. city boundaries)

Raster Data: Data in grid form, where each cell contains information (band) used to represent surfaces where the values change gradually across space. By default, rasters with single-band information are displayed in grayscale, whereas rasters with multi-band information are displayed in colour.

* Continuous raster: Stores continuous data such as rainfall, temperature or elevation
* Discrete data: Stores discrete data such as building types
* Imagery: Stores data produced using a camera or other sensor, such as aerial and drone imagery
* Scanned maps: Existing print maps can be scanned and stored as a raster

Geodatabase: A database (central repository) designed to store, query and manage spatial data

* Feature Class: A collection of features with the same geometry type (point/line/polygon) and a common set of attribute fields. A feature class can store both spatial and non-spatial data.
* Template Feature Class: Defines the features’ attribute fields based on the attribute table of an existing feature class
* Has M: Determines whether the features will contain linear measurement information (M values)
* Has Z: Determines whether the features will contain elevation information (Z values)
* Coordinate System: Defines the coordinate system
* Feature Dataset: A collection of related feature classes that share a common coordinate system. They are used to organise feature classes and ensure that spatial relationships between them are maintained.
* Table: Stores non-spatial data (attributes) that can be linked to spatial features
* Raster Dataset: Store raster data
* Mosaic Dataset: A collection of multiple raster datasets that are treated as a single entity, allowing to manage large collections of raster data efficiently
* Relationship Class: Defines relationships between tables or features classes (one-to-one, one-to-many, many-to-one)
* Topology: Defines the spatial relationship between feature class, ensuring data integrity by enforcing rules such as connectivity, adjacency, overlap
* Networks: Special structures that represent and manage linear features that can be used for modelling flow (e.g. water, traffic, electricity)
* Geometric Networks: For modelling systems like water or electrical networks, where connectivity and flow direction are important
* Network Datasets: Used for transportation networks to model routing, travel times, and directions

A diagram of a data center

Description automatically generated

**Visualisation**

Bookmark: A saved spatial location that allows to quickly navigate to a specific area of interest

Layer: A geographic dataset that represents a specific type of feature (e.g. roads, rivers, population density). These layers are overlayed to perform spatial analysis.

* Feature Layer: Contains vector data (points, lines, polygons) that represent geographic features
* Tile Layer: Pre-rendered images of maps for faster viewing
* Imagery Layer: Used for raster data
* Elevation Layer: Represents terrain (a stretch of land) and surface data in 3D

Map: A collection of layers that are displayed together in a 2D environment

* Static Maps: Can view but not interact with (based on basemap/static layer)
* Dynamic Maps: Can view and interact with (based on operational/dynamic layer)

Scene: Similar to a map but operates in a 3D environment

* Global Scene: Used for data visualised on a global scale (e.g. satellite data, weather patterns)
* Local Scene: Used for smaller areas where detailed terrain and elevation are important (e.g. city planning, building models)

Symbology: The visual styles (colours, shapes, line types and sized) that represent different features or values on the map

A diagram of symbols and text

Description automatically generated

Extrusion: The process of giving z-values (e.g. height, elevation, population) to features to appear in 3D space

Elevation: The absolute vertical distance above sea level

Height: The relative vertical distance from a base surface (such as ground level to the top of a building)

Vertical Coordinate System: Defines how the z-values are measured (i.e. elevation, height)

Triangulated Irregular Network (TIN): A 3D representation of terrain surfaces, created by connecting irregularly spaced points with **triangles** and is useful for showing variation in terrain

Digital Elevation Model (DEM): A raster that represents elevation of terrain

Mesh: A feature composed of triangles in various ways to create a 3-dimesional shape

* Integrated Mesh: Represents a discrete object (e.g. car, tree, building) and the ground surface as a single 3D object - it cannot have attributes
* 3D Object Scene Layer: Represents a discrete feature in 3D – can add attributes
* Multipatch Feature: A format used to represent the outer shells of a discrete feature in 3D. It can store images and textures to imitate the way that a feature looks in the real world – can add attributes