Study Definition Repository (SDR)

Reference Implementation

Solution Architecture

Version 3.0

Document History

Version No.	Date	Author	Revision Description
V0.1	Nov 27th, 2021	Nachiket Vaidya	Initial Version
V1.0	March 11th, 2022	Nachiket Vaidya	Revised
V2.0	August 19th, 2022	ACN	Updated diagrams and authentication related content
V3.0	20-Mar-2023	ACN	Updated diagrams and content as per latest architectural changes

Table of Contents

List of Tables

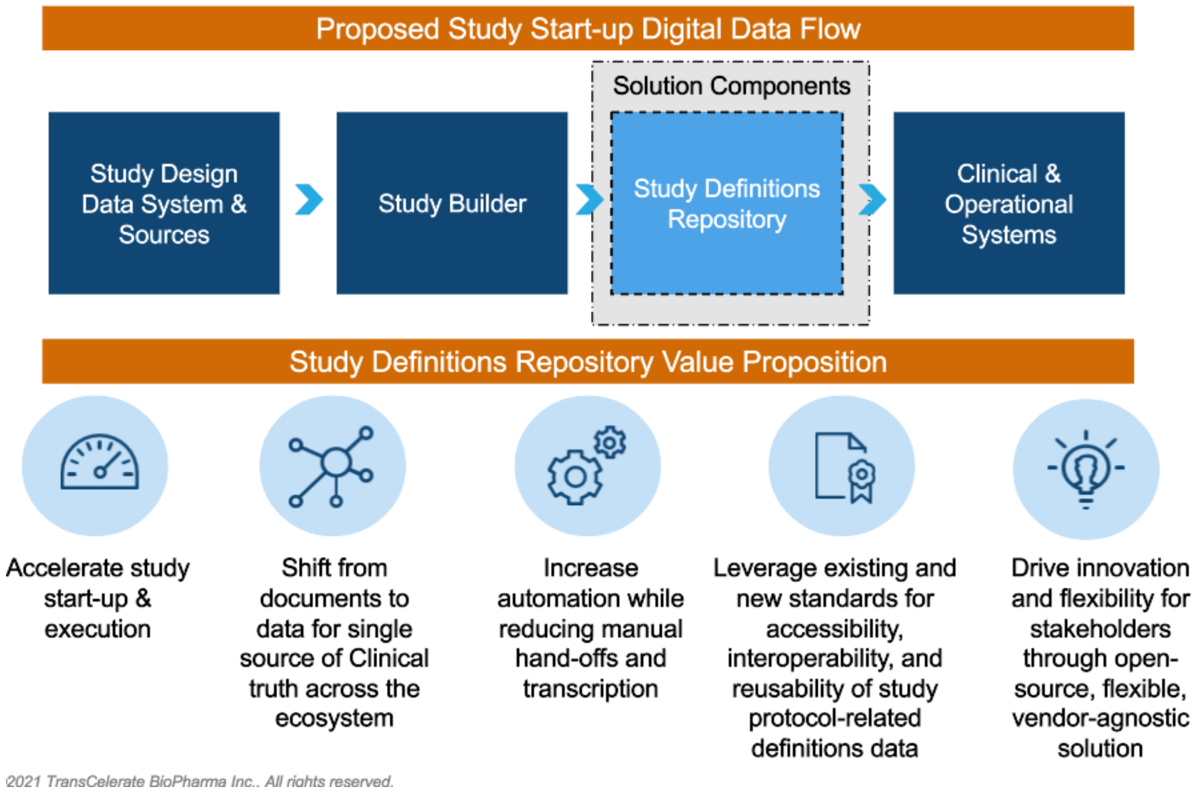
List of Figures

1. Solution Overview

Digital Data Flow (DDF) is TransCelerate's vision to catalyze an industry-level transformation, enabling digital exchange of study definitions (e.g., protocols) by collaborating with standards bodies to create a sustainable open-source Study Definition Repository (SDR) based upon a CDISC standardized model – the Unified Study Definitions Model (USDM). The SDR seeks to transform the drug development process by enabling a digital workflow to move from a current state of manual asset creation to a future state of fully automated and dynamic readiness to support clinical study execution.

The SDR Reference Implementation is an attempt to demonstrate the aforesaid ability to support this vision of the DDF initiative and is implemented in the Azure platform.

Figure 1 - Solution Overview Diagram


Proposed Study Start-up Digital Data Flow
Solution Components
Study Design Data System & Sources
Study Builder
Study Definitions Repository
Clinical & Operational Systems
Study Definitions Repository Value Proposition
Accelerate study start-up & execution
Shift from documents to data for single source of Clinical truth across the ecosystem
Increase automation while reducing manual hand-offs and transcription
Leverage existing and new standards for accessibility, interoperability, and reusability of study protocol-related definitions data
Drive innovation and flexibility for stakeholders through open-source, flexible, vendor-agnostic solution
2021 TransCelerate BioPharma Inc.. All rights reserved.

1.1. High Level Solution Architecture

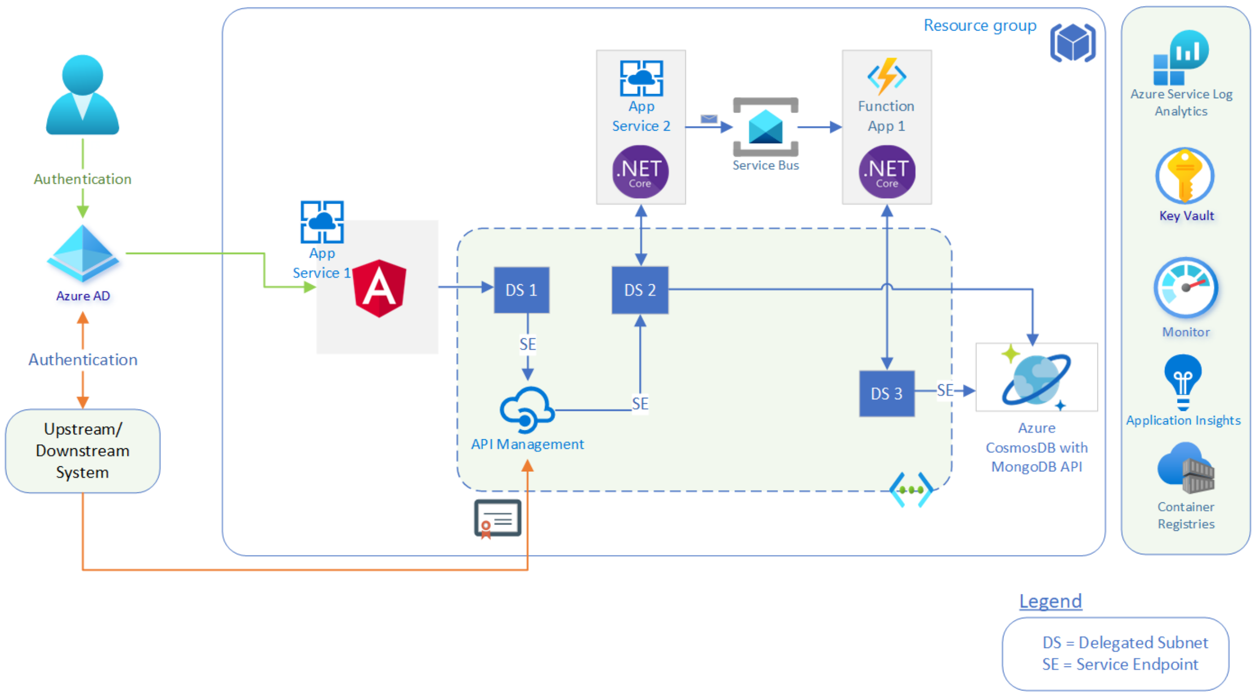
Figure 2 below depicts the high-level architecture of the SDR Reference Implementation, which is built using Angular for the frontend, .NET 6 for the backend and deployed in Microsoft

Azure Cloud[1]. The solution architecture components are chosen in a way to support TransCelerate's objectives of making the future releases cloud and vendor agnostic and support portability and deployment of the reference implementation to other cloud providers.

For the Reference Implementation, the vision is to leverage the USDM provided by CDISC by building a repository to house the USDM complying study data, establishing inbound APIs to enable **upstream systems (e.g., study builders (SB), protocol authors)** to input data, and outbound API to enable outward flow of data to **downstream systems (e.g., Electronic Data Capture systems (EDC)** to automate study start-up activities.

[1] To be clear, TransCelerate does not endorse any particular software, system, or service. And the use of specific brands of products or services by TransCelerate and its collaboration partners in developing the SDR Reference Implementation should not be viewed as any endorsement of such products or services. To the extent that the SDR Reference Implementation incorporates or relies on any specific branded products or services, this resulted out of the practical necessities associated with making a reference implementation available to demonstrate the SDR's capabilities.

Figure 2 - High Level Solution Architecture Diagram



2. Architecture Goals and Constraints

This section provides a description of goals and constraints of Solution Architecture.

2.1. Architecture Goals/Objectives

The architecture for the SDR Reference Implementation has been designed to achieve the following key objectives and architectural goals. The architectural components are chosen in a way to meet the foundational business requirements of being vendor, cloud, and system agnostic.

- **Cloud Agnostic / Open-Source** - Create an application that is relatively cloud agnostic from an implementation perspective by choosing technology stack and cloud components/services that offer extensibility and portability to the application.
- **Accelerate study start-up / execution** by enabling the automation of data flow to downstream clinical systems which reduces the need for duplication, manual input, and transcription.
- **Reduce Manual input** by creating an application that automates data flow.
- Open API Specifications with REST endpoints to maximize system interoperability and promote collaboration.

2.2. Architectural Assumptions and Decisions

The SDR Reference Implementation will leverage Microsoft Azure Cloud Components and services for development and deployment. The choice of solution components, however, has been made to achieve key architectural goals and objects listed in section 2.1 above. Following are some of the key architectural decisions made for the SDR Reference Implementation.

Table 1 - Architectural Decisions

Area	Decision
Connectivity	OAS compliant RESTful API interfaces operate on a Push / Pull model – the upstream vendor is responsible for pushing data into the SDR and the downstream vendor is responsible for pulling data from the SDR. Import and Export functionality within the SDR will be considered for future release.
Role Based Access Control	Role Based Access Controls are designed for accessing different components and PaaS services within Azure. For now, SDR RI will have two roles – SDR User and SDR Admin.
Database	No SQL Database (Cosmos DB) is used to define the Application Data Model and support data versioning, data encryption as well

	as data partitioning standards. Additionally, it also disallows the import of data that is not in USDM format. Mongo DB libraries will be used by the application layer to connect to Cosmos DB, allowing for deployment using Mongo DB in other environments.
Authentication and Authorization	Azure Active Directory (OAuth 2.0) is used to authenticate the user access to the SDR UI and API along with Certificate based authentication.
SDR Upload Version	Study Definitions stored in the repository and assigned a version generated and maintained by the SDR Application.
USDM Version Management	USDM Versions published by CDISC will be maintained by SDR application for each Study Definition stored in the repository. For a given instance of SDR, it will support 3 major versions of USDM (including all minor versions in between).
Auditing	EntryDatetime and EntrySystem are captured in the Audit Log for SDR Reference Implementation.

2.3. Solution Architecture Attributes

2.3.1. Tools and Technologies

The SDR Reference Implementation is built on Azure technology, however, the application components are designed keeping in mind portability and interoperability to other systems or cloud environments such as Amazon Webservices Cloud (AWS)[2], and Google Cloud Platform (GCP)[2].

Below is the list of Tools and Technologies adopted in designing & developing the DDF SDR Reference Implementation.

[2] To be clear, TransCelerate does not endorse any particular software, system, or service. And the use of specific brands of products or services by TransCelerate and its collaboration partners in developing the SDR Reference Implementation should not be viewed as any endorsement of such products or services. To the extent that the SDR Reference Implementation incorporates or relies on any specific branded products or services, this resulted out of the practical necessities associated with making a reference implementation available to demonstrate the SDR's capabilities.

Table 2 - Tools and Technologies

Cloud Services (Azure)	Front End UI Application	Backend API Application	DevOps	Testing
• Azure Subscription (ACP) • Azure AD (OAUTH 2.0) for Authentication/ Security • Azure API Management • Azure App Service • Azure Cosmos DB (Mongo API) • Azure Key Vault • Azure Monitor	Angular 13 Bootstrap HTML5	.NET 6 .NET Entity Framework 5	Terraform GitHub	Functional Testing: • NUnit Testing • Postman for API Testing • SDR UI - Manual Non-Functional Testing: • JMeter for performance testing

2.3.2. Patterns

Azure API Management endpoints are the only set of interfaces that are called by external apps and other services/ systems. The architecture design prevents apps from calling integration services directly, instead all web service requests are channeled through the API Management layer.

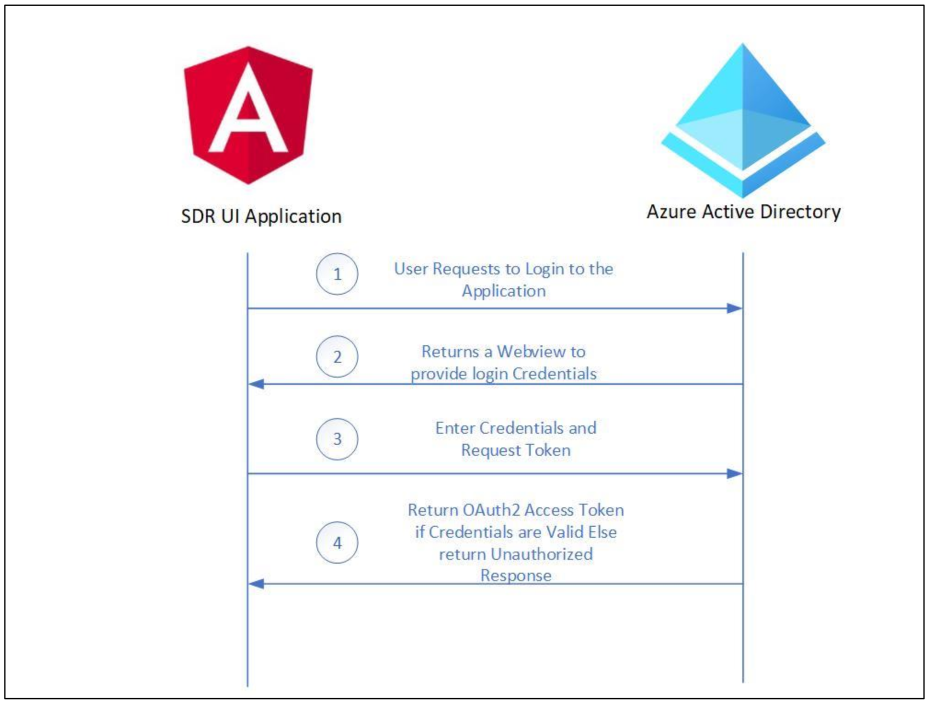
The following principles define how APIs, and the Integration services are built and applied to the wider solution:

- Access to the services layer is always via the API management layer. This is a fundamental architecture principle as it enables security, billing, and endpoint publication.
- The depth of API and Integration service chaining should be kept to a minimum.
- APIs would adhere to Open API Specifications.

2.3.3. Authentication and Authorization

The Solution Architecture for SDR Reference Implementation adopts Microsoft's cloud-based identity and access management service Azure Active Directory for authenticating the users as well as authorization. Azure AD is to be the identity provider for generating access tokens using OAuth2.0 standards which are the main method of authentication. Additionally, certificate-based authentication is enabled on APIM for externally exposed API endpoints. Managed Identities (MSI) is the primary method of authentication between APIM and backend Azure services. Authorization is implemented using the App roles on Azure App Registrations.

Figure 3 - Authentication and Authorization Workflow


SDR UI Application
Azure Active Directory
1
User Requests to Login to the Application
2
Returns a Webview to provide login Credentials
3
Enter Credentials and Request Token
4
Return OAuth2 Access Token if Credentials are Valid Else return Unauthorized Response

2.3.4. Portability

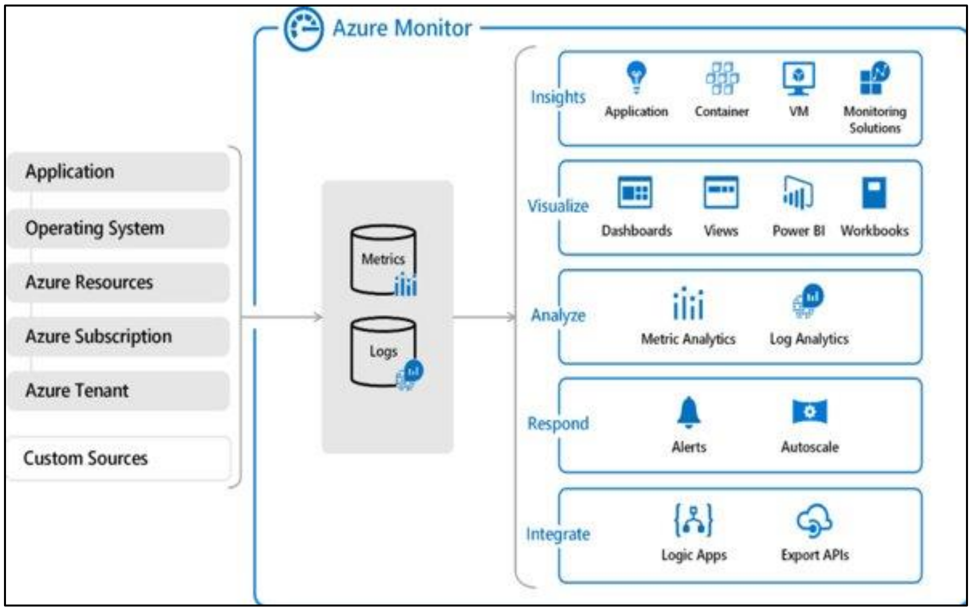
The architecture of SDR Reference Implementation has been designed, keeping in mind portability, which is defined as the ability to move and suitably adapt the applications and data between systems and cloud services (from one cloud service/system provider to different cloud providers and/or systems and services). Specifically:

Data Portability: The data model of the solution has been designed using Cosmos DB (Azure's version of MongoDB) and Mongo APIs which store data in collections in JSON format. This approach allows to easily transfer data from Cosmos DB to any other No SQL database.

Application Portability: Technology stack and PaaS services chosen for the Reference Implementation ensure ease of porting/deploying the application components to different cloud providers or target environments. The SDR RI codebase should be updated with relevant minimal changes to integrate with the target cloud platform. The DDF SDR Platform Agnostic Recommendation document lists the parallels of corresponding platform components between multiple cloud providers. The Front end of the application is built using Angular which is a popular choice for building the web application and has a build in process that is standard across any build environment. The API layer is built using .NET 6 and is deployed in Azure App Service.

2.3.5. Diagnostics and Logging

Figure 4 - Azure Monitor



Diagnostic logs provide rich information about operations and errors that are important for auditing as well as troubleshooting purposes. Diagnostic logs differ from activity logs, as activity logs mainly provide insights into the operations performed on Azure resources. Diagnostic logs provide insight into operations performed by resources. The SDR Reference architecture utilizes Application Insights for Diagnostic Logging and Application Logging.

Shared dashboards configured in Application Insights allow for troubleshooting and diagnostic tracking statistics as well as recording logs related to API Usage, Reliability, Responsiveness and Failures. Application Logging enables tracing the application calls and any exceptions that may arise because of code failures. The App Insights dashboards are accessible via the Azure portal.

2.3.6. Alerting

Metric alerts in Azure Monitor provide notifications when one of the metrics crosses a threshold value. Metric alerts work on a range of multi-dimensional pre-set platform and custom metrics, including Application Insights standard and custom metrics. Alerts proactively provide notifications when important conditions are found in the monitored data. Which allows the identification and addressing of issues before users of the system notice them. Alert rules are the actions taken when an alert fires, which captures the target and criteria for alerting. The alert rule can be either in an enabled or a disabled state and are only fired when enabled. For the SDR Reference Architecture alerts are configured to send emails and notify stakeholders when the resource utilization crosses 80% of its threshold usage.

3. Application Architecture

The application architecture section describes and defines the solution’s architecture, including the major solution components, their relationships and the technologies and tools used to build them. The Application Layer for the Solution has a frontend (UI) Application built using Angular 13 and an API Layer built using .NET 6 and follows the Open API Specifications.

3.1. Frontend Application (UI)

The frontend application for the solution is designed using Single Page Application (SPA) pattern and follows a Model View Controller (MVC) architectural style to integrate with the backend solution components. The User Interface for the application has been designed to enable the user to perform several business functions such as View Study Details, Search Study Definitions, Study Comparison and System Usage Report.

3.2. API Layer

The API layer of SDR Reference Implementation is designed to ensure grouping of endpoints by the scope of their usage – externally exposed / public APIs and internal APIs consumed by SDR UI. All the endpoints have basic authentication configured and the externally exposed endpoints have additional client certificate-based authentication as well (two-factor authentication).

3.3. API Service Specifications

The API Layer of the solution complies with the OpenAPI Specification (OAS) standards which allows systems to discover and understand the capabilities of the service without access to the source code, documentation, or through the network traffic inspection. When properly defined, a consumer can understand and interact with remote services with a minimal amount of implementation logic.

3.4. API Architectural Style

The API Layer for the SDR follows the REST architectural style that uses HTTP requests to GET, PUT, and POST data. REST standards are not linked with any technology or platform, and introduce the best practices known as constraints. They also describe how the server processes requests and responds to them. Operating within RESTful architecture constraints, the system gains desirable properties such as reliability, ease of use, improved scalability and security, low latency while enhancing the system performance and helping achieve technology independence in the process.

3.5. API Component Model

The API Layer components for the SDR Reference Implementation are developed using .NET classes written in C#.

4. Data Architecture

Data architecture is dependent on multiple factors mainly associated with how data moves across different upstream and downstream systems. Specifically, to standardize the data exchange between different systems, the SDR Reference Implementation only persists the data in a USDM conformant - JSON format in Cosmos DB (NoSQL DB PaaS in Azure). The SDR RI leverages one exclusive collection for storing study definitions while supporting features like change audit, group & user management are maintained in separate collections. The study definition collection is a common collection that stores all study definitions irrespective of the USDM version they conform to.

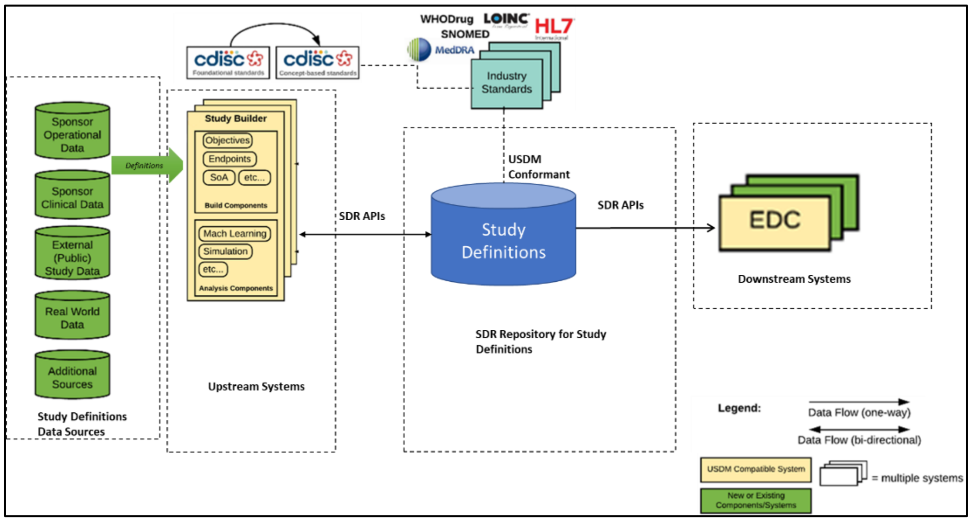
The NoSQL database design allows for persisted Study Definitions as JSON documents with built in support for tracking changes and auto indexing to ensure optimum system performance. Additionally, it also supports future SDR needs related to data partitioning.

4.1. Conceptual/Logical Data Model

Refer to the CDISC Phase 2 USDM V1.9 data model here.

4.2. Data Sources

Figure 5 - Up-stream and Down-stream systems for SDR



4.3. Data Dictionary

The Data Dictionary captures the USDM conformance rules and relationships for all the data elements of Study Definition. The simplified version of Data Dictionary used for SDR Application available in GitHub link.

For more details, refer CDISC portal here.

5. Security Architecture

5.1. Security Solution Overview

All components of the architecture are designed to communicate through secure connections to allow alternative hosting and deployment models for high availability or disaster recovery situations. Building the architectural elements with these considerations in mind ensured that the stability and security of the solution will not be compromised regardless of any situation or compromise to any of the individual component. All the architectural components are leveraging the default security configurations provided by the

Cloud Service provider which are compliant with latest SSL requirements. To prevent unauthorized access client certificates are configured and enforced via APIM policies.

6. Infrastructure

The infrastructure section provides a description of the Azure resources that are deployed as part of Azure Platform for the SDR Reference Implementation.

6.1. Azure Platform Components

The reference Architecture for SDR is being deployed in Microsoft's public cloud platform, Azure. A single tenant single subscription model has been configured to house application code and data for delivering application functionality.

Below is the list of different Azure components and services utilized in the reference implementation of SDR.

Table 3 - Azure Platform Components

Architecture Area	Configuration Items	Azure Component
Governance	Tagging	Azure Tags
	Naming Convention	https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-best-practices/resource-naming
	Resource Group	Azure Resource Groups
	Cost Management	Azure Cost Management Billing
Subscription & Regions	Subscription	Azure Subscription
	Region	Azure Region (US East)
Networking	Vnet	Azure VNET for PaaS integration
	Subnet	Azure Subnet
	DNS	Azure DNS
Identity	Identity Provider	Azure AD
	Users	Active Directory Users
	Groups	AAD Security Groups
	Service Principals	AAD SP
	Managed Identity	Azure Managed Identities

Architecture Area	Configuration Items	Azure Component
	RBAC	AAD & Subscription roles
Security	Security Monitoring	Azure Defender
	Baseline Policy	
	Key Management	Azure Key Vault
	DDOS	
	API Gateway Inbound Control	Azure APIM Inbound Policies
Operations	Logging	Application Insights
	Monitoring	Azure Monitor

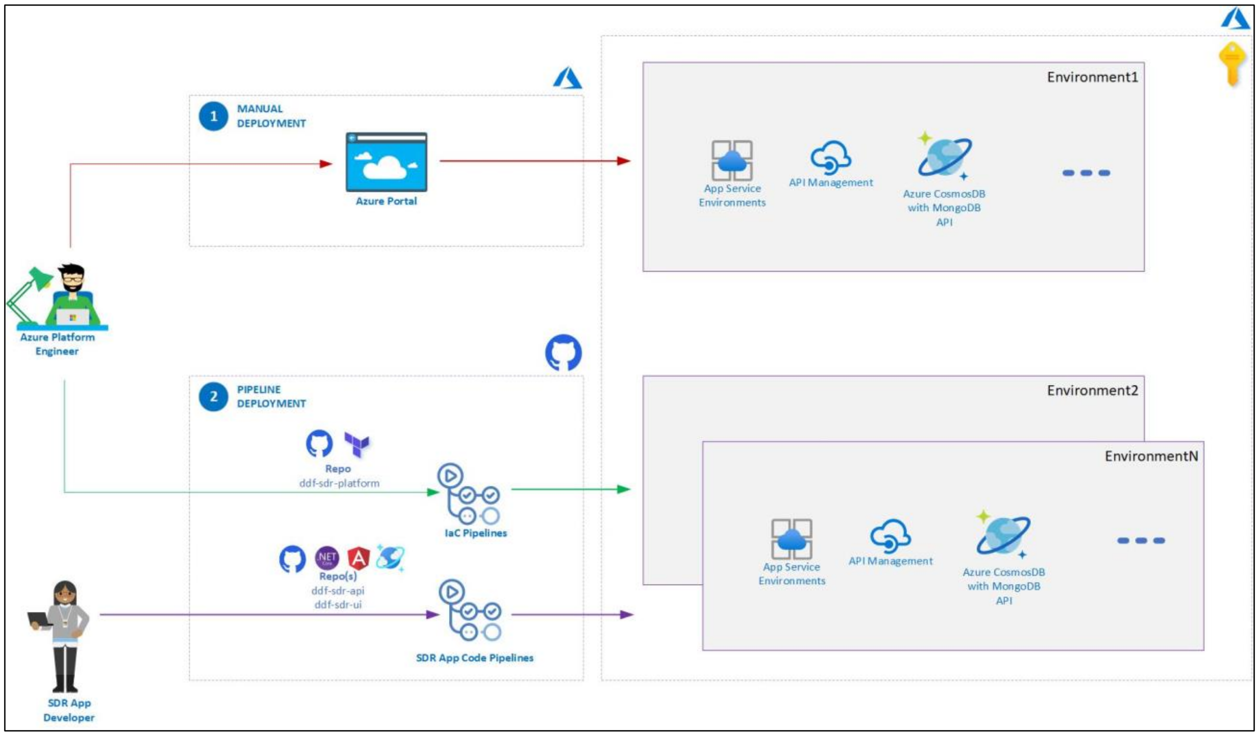
6.2. Deployment Models

The Deployment Models section describes different ways of deploying the Azure Platform Resources.

Deployment Models:

- **Manual Deployment** – All the Azure resources are manually deployed using Azure Portal.
- **Pipeline Deployment** – The deployment of Azure resources through terraform code using a YAML script.

Figure 6 - Deployment Models



Appendix A - Key Terms

Error! Reference source not found.A below provides definitions and explanations for terms and acronyms relevant to the content presented within this document.

Table 4 - Appendix A: Key Terms

Term	Definition
AAD	Azure Active Directory
API	Application Programming Interface
AWS	Amazon Web Services
DDF	Digital Data Flow
EDC	Electronic Data Capture
GCP	Google Cloud Platform
PAAS	Platform as a Service
RI	Reference Implementation
SB	Study Builder
SDR	Study Definition Repository
USDM	Unified Study Definition Model