TransCelerate BIOPHARMA INC.

# Study Definition Repository (SDR)

## Reference Implementation

## Solution Architecture

## Version 1.0

# Document History

| Version No. | Date | Author | Revision Description |
|---|---|---|---|
| V1.0 | Nov 27th, 2021 | Nachiket Vaidya | Initial Version |
| V1.0 | March 11th, 2022 | Nachiket Vaidya | Revised |
| | | | |
| | | | |
| | | | |

# Table of Contents

1. Solution Overview ..... 5
1.1. High Level Solution Architecture ..... 5
2. Architecture Goals and Constraints ..... 7
2.1. Architecture Goals/Objectives ..... 7
2.2. Architectural Assumptions and Decisions ..... 7
2.3. Solution Architecture Attributes ..... 8
2.3.1. Tools and Technologies ..... 8
2.3.2. Patterns ..... 9
2.3.3. Authentication and Authorization ..... 9
2.3.4. Portability ..... 10
2.3.5. Diagnostics and Logging ..... 11
2.3.6. Alerting ..... 11
3. Application Architecture ..... 12
3.1. Front End Application (UI) ..... 12
3.2. API Layer ..... 12
3.3. API Service Specifications ..... 12
3.4. API Architectural Style ..... 12
3.5. API Component Model ..... 13
4. Data Architecture ..... 13
4.1. Conceptual/Logical Data Model ..... 14
4.2. Data Sources ..... 14
4.3. Data Dictionary ..... 15
5. Security Architecture ..... 15
5.1. Security Solution Overview ..... 15
6. Infrastructure ..... 15
6.1. Azure Platform Components ..... 15
6.2. Environment Strategy for SDR ..... 17
6.3. Deployment Models ..... 17
Appendix A - Key Terms ..... 19

# List of Tables

Table 1 - Architectural Decisions ..... 7
Table 2 - Tools and Technologies ..... 9
Table 3 - Azure Platform Components ..... 15
Table 4 - Environment Details ..... 17
Table 5 - Appendix A: Key Terms ..... 19

# List of Figures

Figure 1 - Solution Overview Diagram ..... 5

Figure 2 - High Level Solution Architecture Diagram ..... 6

Figure 3 - Authentication and Authorization Workflow ..... 10

Figure 4 - Azure Monitor ..... 11

Figure 5 - Component Relationship Diagram ..... 13

Figure 6 - UML Diagram for Clinical Study ..... 14

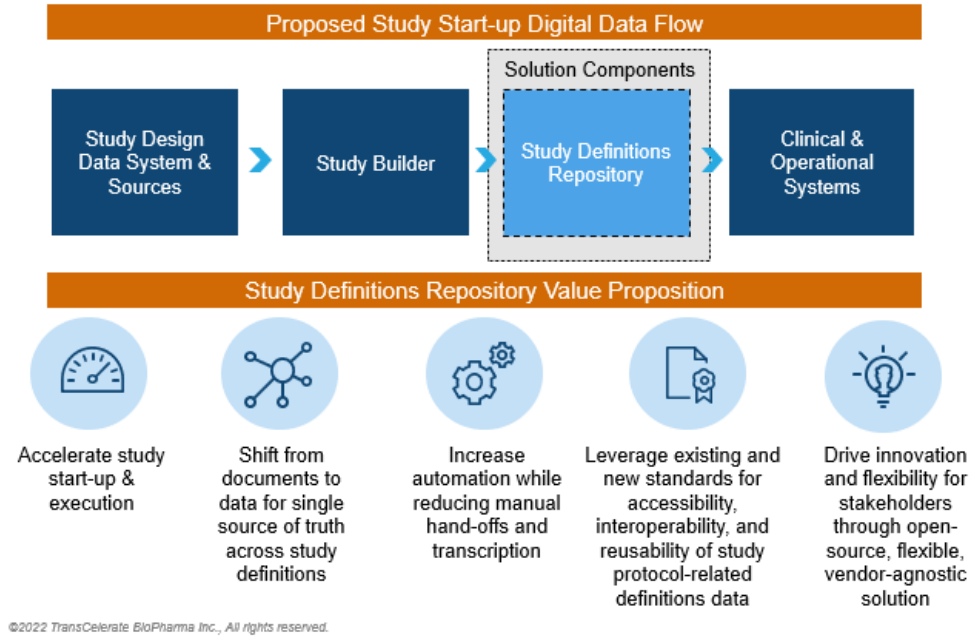
Figure 7 - Upstream and Downstream systems for SDR ..... 14

Figure 8 - Deployment Models ..... 18

# 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., protocol) by collaborating with standards bodies to create a standardized model – the Unified Study Definitions Model (USDM) – that serves as the foundation for a sustainable open-source automation and interoperability solution known as the Study Definition Repository (SDR) Reference Implementation. The SDR Reference Implementation 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.

*Figure 1 - Solution Overview Diagram*



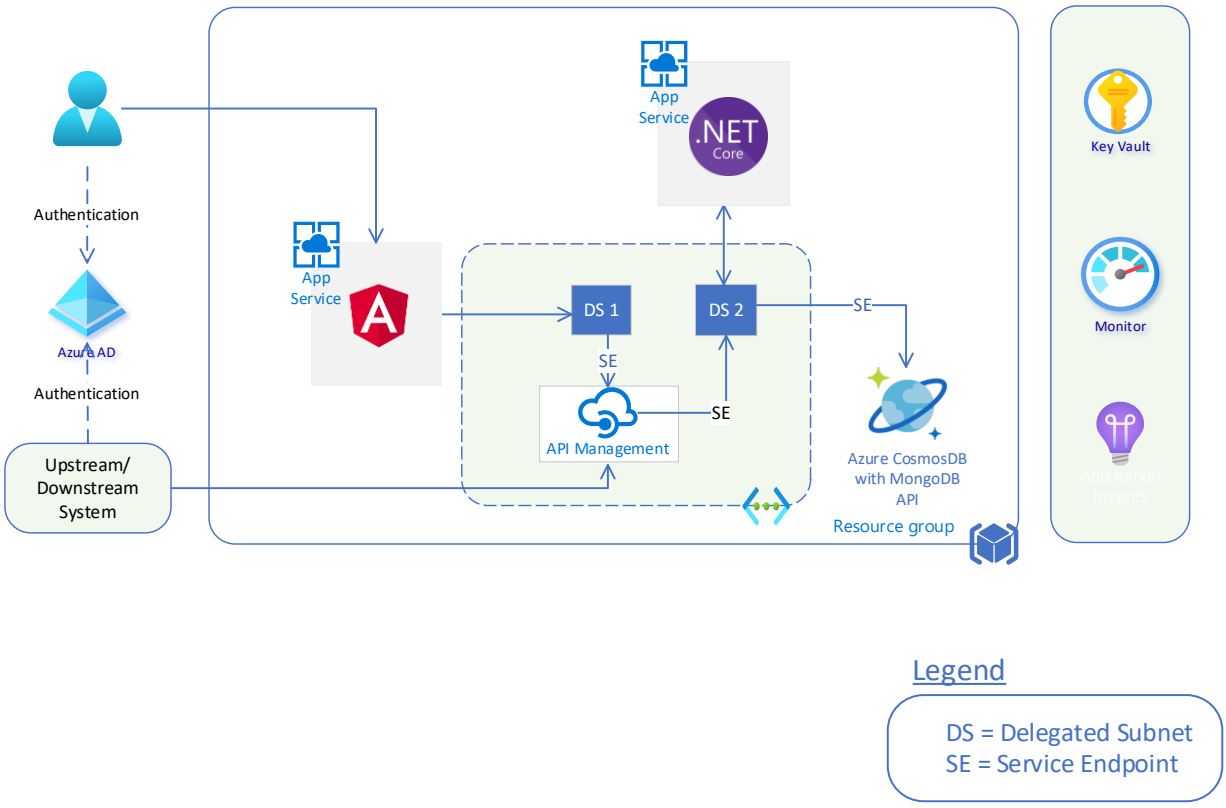

## 1.1. High Level Solution Architecture

Figure 2 below depicts high level architecture of the SDR Reference Implementation which is built using Angular Front End and.NET Core Backend and deployed in Microsoft Azure Cloud. 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. 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 environments.

For the Reference Implementation, the vision is to leverage the USDM provided by the Clinical Data Interchange Standards Consortium (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)**.

*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.

- **Technology Agnostic**- Create an application that is relatively technology 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. Event based notifications and/or support for GraphQL will be considered in future releases. |
| Role Based Access Control | Role Based Access Controls are designed for accessing different components and PAAS services within Azure. However, for the purpose of MVP, it will have single role type for the necessary users |

| | |
|---|---|
| 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 interface 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 application while Certificate based authentication is used to authenticate API calls |
| Versioning | Study Definitions stored in the repository and assigned a version generated and maintained by the SDR Application |
| 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.

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

*Table 2 - Tools and Technologies*

| Cloud Services (Azure) | Front End UI Application | Backend API Application | Dev ops | Testing |
|---|---|---|---|---|
| Azure Subscription (ACP)<br>Azure AD (OAUTH 2.0) for Authentication/Security<br>Azure API Management<br>Azure App Service<br>Azure Cosmos DB (Mongo API)<br>Azure Key Vault<br>Azure Monitor | Angular 11<br>Bootstrap<br>HTML5 | .Net 6<br>.Net Entity Framework 5 | Terraform<br>GitHub | Functional Testing:<br>• NUnit Testing<br>• Postman for API Testing<br>• SDR UI - Manual<br>Non-Functional Testing:<br>• 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:

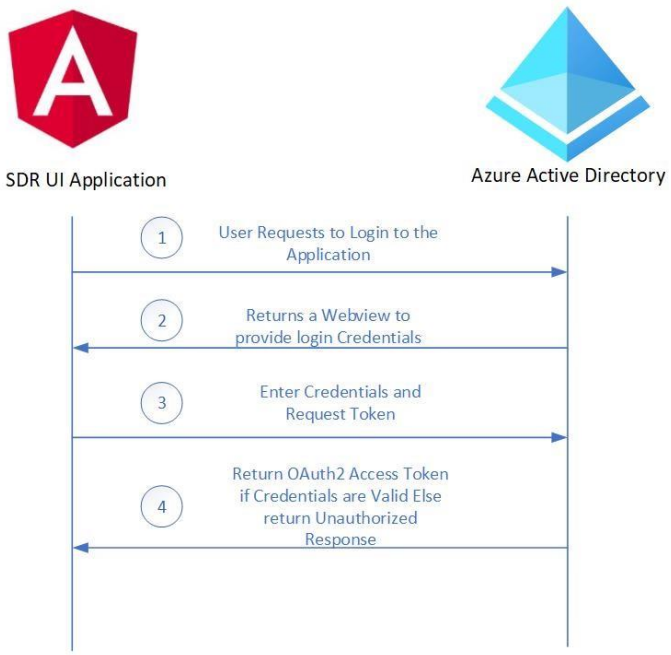
- Access to the services layer is always via the API management layer. This is a fundamental architecture principle as it enables security, billing, and endpoint publication.
- The depth of API and Integration service chaining should be kept to a minimum
- APIs will adhere to Open API Standards as defined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

### 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 authorizing the user access to different service components within the application architecture. Azure AD is to be the identity provider for generating access tokens using OAuth2.0 standards which are the main method of authentication. Managed Identities (MSI) is the primary method of authentication between APIM and backend Azure services. Shared Access Signatures

(SAS) keys will be leveraged where MSIs are not supported. Authorization is managed within each API and through Role Based Access Control (RBAC) where applies.

*Figure 3 - Authentication and Authorization Workflow*




### 2.3.4. Portability

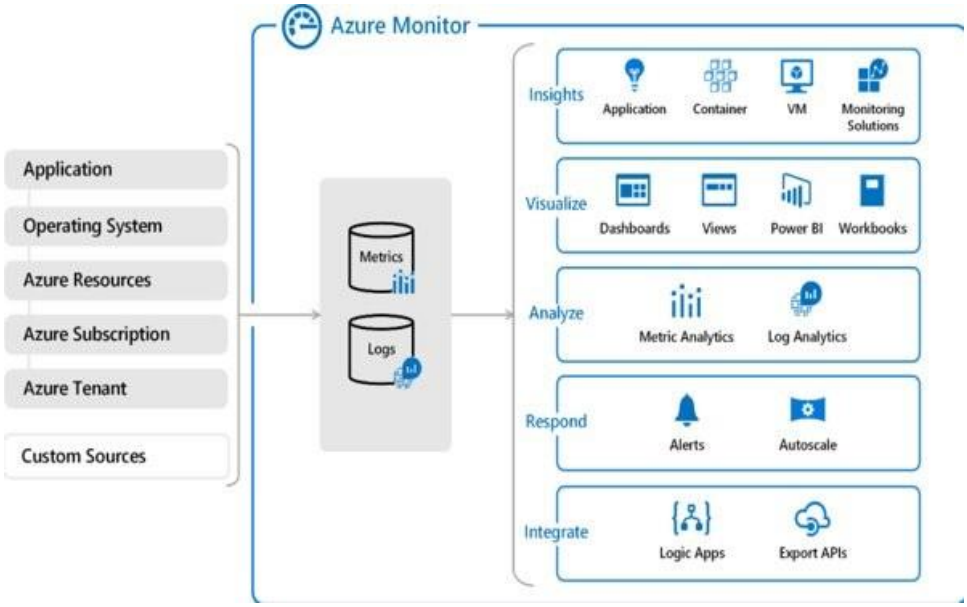
The architecture of SDR Reference Implementation has been designed, keeping in mind portability. Which is defined as the ability to move and suitably adapt the applications and data between systems and cloud services (from one cloud service/system provider to different cloud providers and/or systems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

**Data Portability**: The data model of the solution has been designed using Cosmos DB 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 or any other relational database with minimal transformation effort.

**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 by simply recompiling the codebase without significant changes. 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 Core and is deployed in Azure App Service. The current API layer built allows deployment to any containerized solutions such as Azure Kubernetes Service (AKS) or Docker.

### 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.

### 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 Front End (UI) Application built using Angular 11 and an API Layer built using .Net Core and follows the Open API Specifications.

## 3.1. Front End Application (UI)

The front-end Application for the solution is designed using Single Page Application pattern and follows a Model View Controller 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, Export Study Definitions, Protocol Assembly Dashboard and a Search and Filter functions.

## 3.2. API Layer

API elements have a well-defined scope and consumption scenarios. APIs should therefore have the minimum level of accessibility exposed to align with the intended consumption. Keeping in mind that exposing APIs, in a manner which does not align with their intended consumption, is a potential attack vector and could lead to system behavioral failure.

The correct implementation of public/private scoping allows for certain details to be hidden and therefore reduces the likelihood of over intimacy and tight coupling in the implementation. This design principle when properly followed allows changes to implementation without causing risk of significant regression and the introduction of technical debt.

The SDR Reference Implementation follows this design principle for class level interfaces, their members as well as the members exposed publicly from the compiled service interfaces. This aligns with the principles of micro-service architectural approach. A simple summary of the principle would be to publicly expose WHAT the implementation does and to retain privacy of HOW the implementation achieves it.

## 3.3. API Service Specifications

The API Layer of the solution complies with the Open API 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

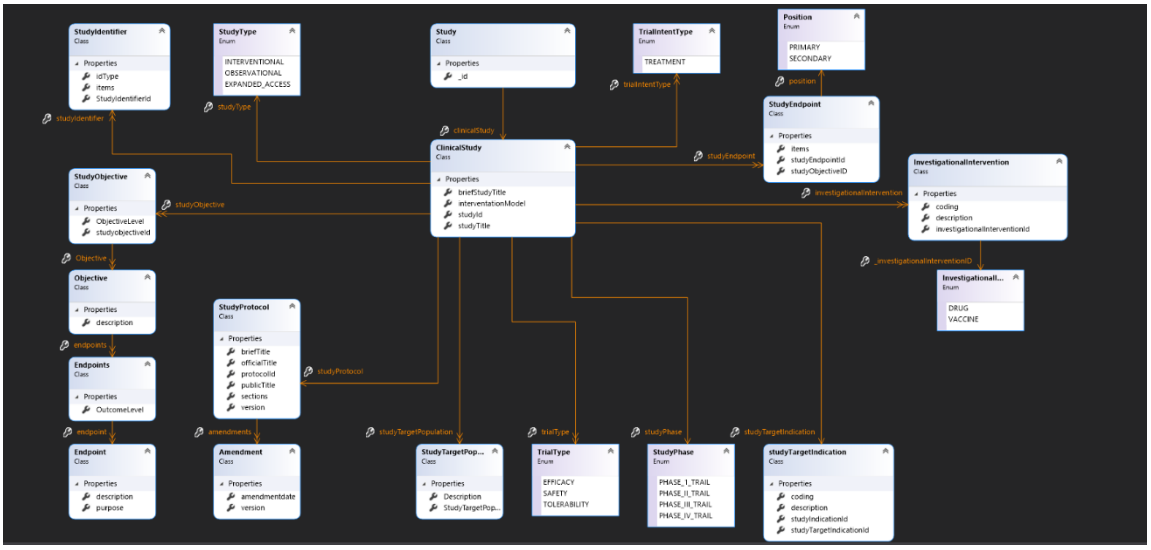
The API Layer for the SDR follows the REST architectural style that uses HTTP requests to GET, PUT, POST and DELETE data. REST standards are not linked with any technology or platform, and introduce the best practices known as constraints. They also describe how the server processes requests and responds to them. Operating within these 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 C# classes. The section below shows the relationship between the different components in API Layer.

*Figure 5 - Component Relationship Diagram*

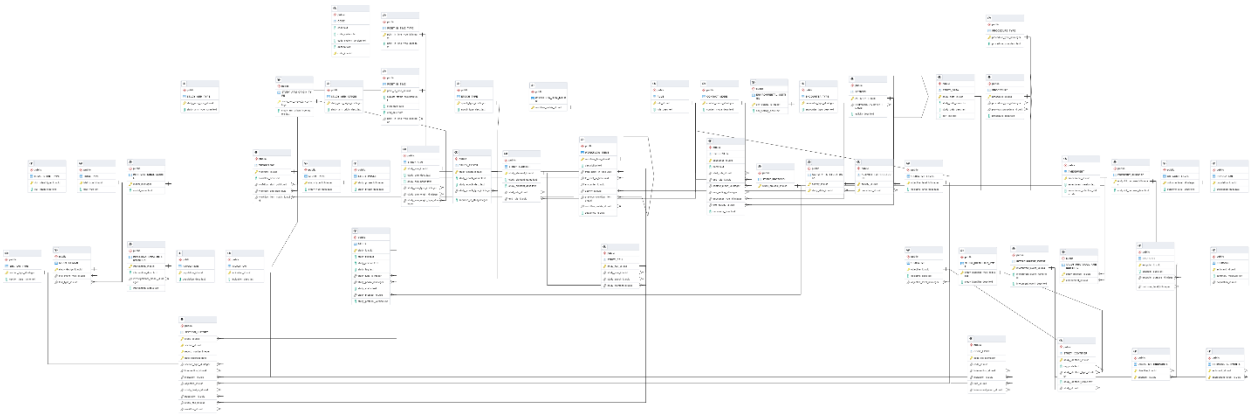



## 4. Data Architecture

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 architecture only persists the data in a USDM conformant - JSON format in Cosmos DB (NoSQL DB PAAS in Azure). 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

*Figure 6 - UML Diagram for Clinical Study*

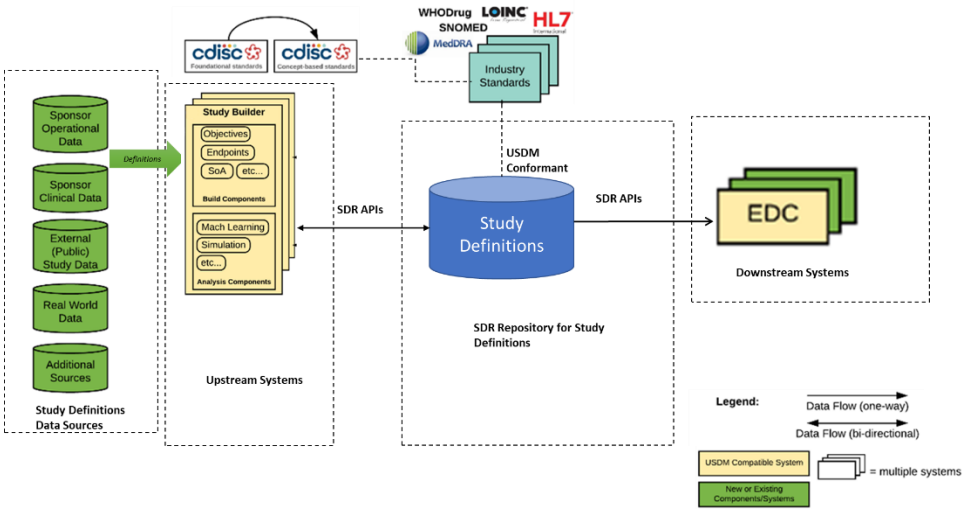


Open attached object for better viewing experience of UML Diagram.

SDR-CDISC-UML.png

## 4.2. Data Sources

*Figure 7 - Upstream and Downstream systems for SDR*



### 4.3. Data Dictionary

The Data Dictionary created by CDISC captures the USDM conformance rules and relationships for all the data elements of Study Definition. The simplified version of Data Dictionary used for SDR Reference Application attached below.

DataDictionary-SDR_ Reference_Applicatior

For more details on Data Dictionary, 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 alternate 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.

The following security measures are configured and applied via APIM Policies:

- Enforce pre-access checks - validate OAuth2.0 token policy - valid and proper claims
- Prevent unauthorized access - Using client certificates
- Prevent excessive usage - rate and quota limits policies Configured Alerts

## 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 |

| Architecture Area | Configuration Items | Azure Component |
|---|---|---|
| | 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 | Azure Vnet | Azure VNET for PaaS integration |
| | Subnet | Azure Subnet |
| | DNS | Azure DNS |
| Connectivity | Remote Access to internet facing resources | Direct connect from Internet |
| Identity | Identity Provider | Azure AD |
| | Users | Active Directory Users |
| | Groups | AAD Security Groups |
| | Service Principals | AAD SP |
| | Managed Identity | Azure Managed Identities |
| | RBAC | AAD & Subscription roles |
| Security | Security Monitoring | Azure Defender |
| | Baseline Policy | |
| | Other Policy | |
| | Key Management | |
| | DDOS | |
| | APIM Inbound Control | |
| | Cosmos DB Inbound Control | |

| Architecture Area | Configuration Items | Azure Component |
|---|---|---|
| Operations | Logging | Application Insights |
| | Monitoring | Azure Monitor |

### 6.2. Environment Strategy for SDR

For SDR Reference Architecture, 3-tier environment model is recommended as mentioned in below.

*Table 4 - Environment Details*

| Environment | Details |
|---|---|
| Dev | Development of SDR (This env is only accessible to SDR development/testing teams) |
| QA/UAT | For handling UAT, tickets, bugs, fixes. (This env is only accessible to SDR development/testing teams) |
| Pre-Prod | With QA cleared code applied on with almost production ready data (this environment is accessible to SDR users) |

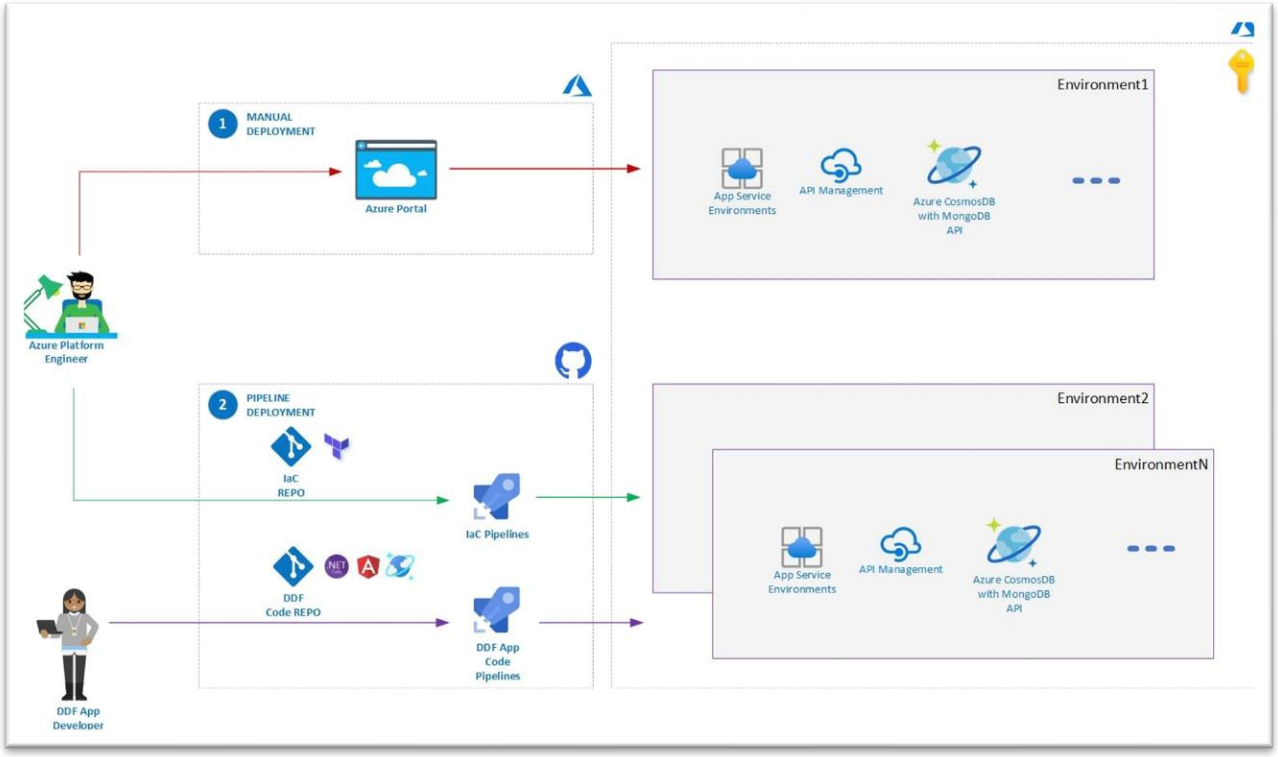
### 6.3. Deployment Models

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 Pipeline.

*Figure 8 - Deployment Models*



# Appendix A - Key Terms

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

*Table 5 - Appendix A: Key Terms*

| Term | Definition |
|---|---|
| DDF | Digital Data Flow |
| MVP | Minimum Viable Product |
| SDR | Study Definition Repository |
| API | Application Programming Interface |
| EDC | Electronic Data Capture |
| SB | Study Builder |
| AAD | Azure Active Directory |
| USDM | Unified Study Definition Model |
| PAAS | Platform as a Service |