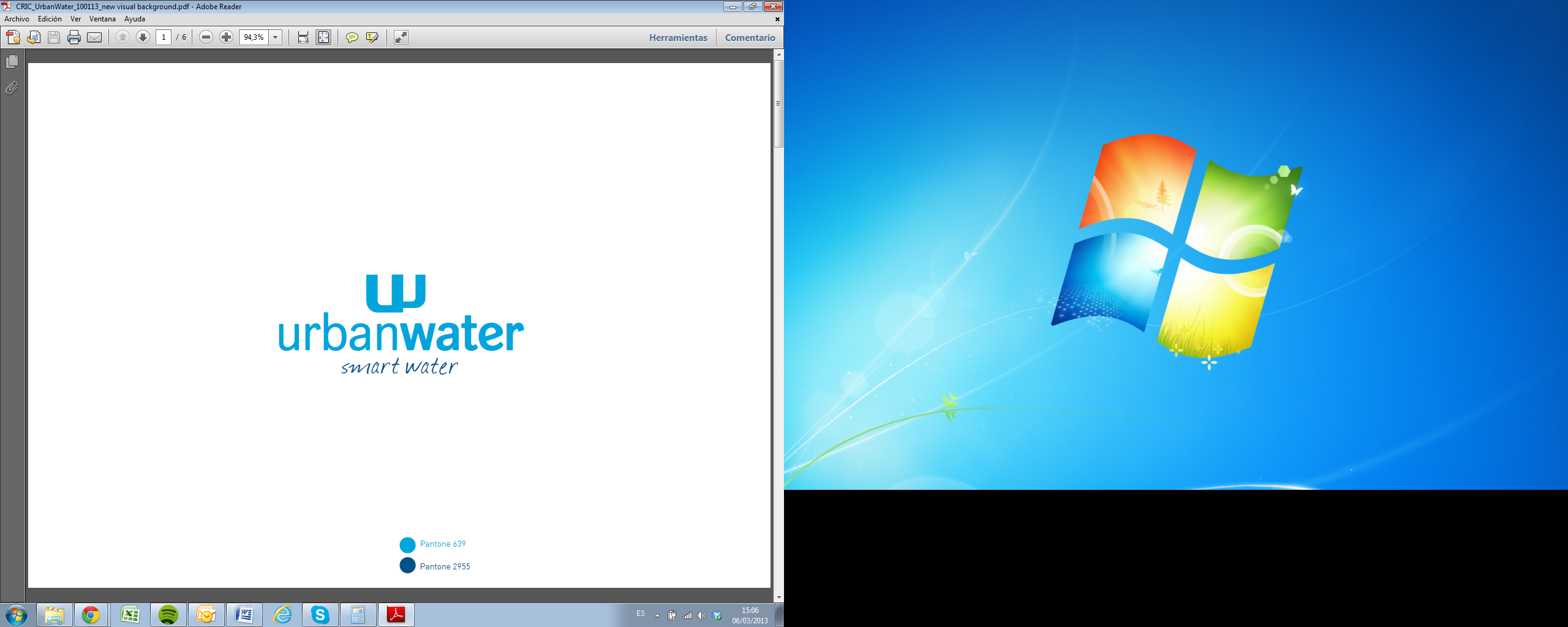
Project Title:

Intelligent Urban Water Management System

Project Acronym:

URBANWATER



**Seventh Framework Programme**

Collaborative Project

Grant Agreement Number 318602

Subject:

**D3.2 – Data Management Cloud Platform – WP3**

Dissemination Level: PUBLIC

Lead beneficiary: Red Skies Limited

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision | Preparation date | Period covered | Project start date | Project Duration |
| FINAL/v1.2 | Month 06 | Months 06 to 09 | December 2012 | 30 Months |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under the UrbanWater project, grant agreement no. 318602.  This publication reflects only the author's views and the European Union is not liable for any use that may he made of the information contained therein. | | |

# History of Changes

|  |  |  |
| --- | --- | --- |
| Version | Author, Institution | Changes |
| 0.1 | Martin Kenirons | First Draft |
| 1.0 | Martin Kenirons | Updated with reviewer comments |
| 1.2 | Martin Kenirons | Complete rewrite |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Abstract

This document describes the proposed design for the data management cloud platform system in the UrbanWater research project. Therefore, it contains information about the technologies considered and flow diagrams of how different component will interact.

# List of Acronyms

|  |  |
| --- | --- |
| AES | Advanced Encryption Standard |
| SWT | Simple Web Token |
| ACS | Access Control Services |
| WASABi | Windows Azure AutoScaling Application Block |
| SaaS | Software As A Service |
| PaaS | Platform As A Service |
| IaaS | Infrastructure As A Service |
| OData | Open Data |
| WCF | Windows Communication Framework |
| WIF | Windows Identification Foundation |
| DNS | Domain Name Server |
| CDN | Content Delivery Network |
| VPC | Virtual Private Cloud |
| VPN | Virtual Private Network |
| EC2 | Amazon Elastic Compute Cloud |
| EMR | Amazon Elastic MapReduce |
| EBS | Amazon Elastic Block Storage |
| RDS | Amazon Relational Database Service |
| SQS | Amazon Simple Queue Service |
| SWF | Amazon Simple Workflow |
|  |  |
|  |  |
|  |  |

**Table of Contents**

[History of Changes 2](#_Toc382572082)

[Abstract 3](#_Toc382572083)

[List of Acronyms 4](#_Toc382572084)

[List of Figures 8](#_Toc382572085)

[List of Tables 8](#_Toc382572086)

[1 Introduction 9](#_Toc382572087)

[1.1 Objectives of UrbanWater 9](#_Toc382572088)

[1.2 Objectives of WP 3 (Data management cloud platform) 9](#_Toc382572089)

[1.3 Objectives of this document 10](#_Toc382572090)

[1.4 Structure of this document 10](#_Toc382572091)

[2 Cloud Computing 12](#_Toc382572092)

[2.1 Cloud computing & the UrbanWater Project 12](#_Toc382572094)

[2.2 Service Models 12](#_Toc382572095)

[2.3 Deployment Models 13](#_Toc382572096)

[2.4 Cloud Computing Technologies 13](#_Toc382572097)

[2.5 Amazon Web Services 13](#_Toc382572098)

[2.5.1 Compute 13](#_Toc382572101)

[2.5.2 Networking 14](#_Toc382572103)

[2.5.3 Content Delivery 14](#_Toc382572104)

[2.5.4 Storage & Content Delivery 14](#_Toc382572105)

[2.5.5 Database 14](#_Toc382572106)

[3.5.6 Management 15](#_Toc382572107)

[3.5.7 App Services 15](#_Toc382572108)

[2.6 Windows Azure 15](#_Toc382572109)

[2.6.1 Windows Azure 15](#_Toc382572110)

[2.6.2 Windows Azure platform AppFabric 17](#_Toc382572111)

[2.6.3 SQL Azure 18](#_Toc382572112)

[2.6.3 Scalability & Redundancy in Azure 18](#_Toc382572113)

[2.7 Selection of Cloud computing technology 19](#_Toc382572114)

[3 Security & Open Data 21](#_Toc382572115)

[3.0 Open Data Protocol 21](#_Toc382572116)

[3.1 What is a WCF Data Service 22](#_Toc382572117)

[3.2 Claims based Authentication 23](#_Toc382572118)

[3.3 Oauth & SWT 24](#_Toc382572119)

[3.4 Certificate Authority 24](#_Toc382572120)

[3.5 Encryption (AES) 24](#_Toc382572121)

[4 UrbanWater Data 25](#_Toc382572122)

[4.1 Water Network Meters Data 25](#_Toc382572127)

[4.2 Topology Data 25](#_Toc382572128)

[5.2.1 Node Geometry 25](#_Toc382572129)

[5.2.1 Line Geometry 26](#_Toc382572130)

[5.2.3 Junctions 26](#_Toc382572131)

[5.2.4 Reservoirs 27](#_Toc382572132)

[5.2.5 Tanks 27](#_Toc382572133)

[5.2.6 Valves 28](#_Toc382572134)

[5.2.7 Pumps 29](#_Toc382572135)

[5.2.8 Pipes 31](#_Toc382572136)

[4.3 Meteorological Data (Non Raster Field Data) 33](#_Toc382572137)

[5.3.1 Sensor Data Table 33](#_Toc382572138)

[5.3.2 Product Data Table 33](#_Toc382572139)

[5.3.3 Assignment Data Table 34](#_Toc382572140)

[5.3.4 Record Data Table 34](#_Toc382572141)

[5.3.5 Example 35](#_Toc382572142)

[4.4 Raster Field Data 37](#_Toc382572143)

[5.4.1 Raster fields Data Table 37](#_Toc382572144)

[5.4.2 Raster Record Data Table 38](#_Toc382572145)

[4.5 Calendar Data 38](#_Toc382572146)

[4.6 Private Model Results Data 39](#_Toc382572147)

[4.7 Customer & Tariff Data 39](#_Toc382572148)

[5 Data management Cloud Platform Design 40](#_Toc382572149)

[5.0 Overview 40](#_Toc382572150)

[5.1 Azure Table Storage – NO-SQL Database 41](#_Toc382572151)

[5.2 Integrating with Data Providers(Utilities and Head End System) 42](#_Toc382572152)

[5.3 Data management “public” cloud platform data flow cycle 44](#_Toc382572153)

[5.4 Worker Role for Storage Monitoring 46](#_Toc382572154)

[5.5 Worker Role for Data Processing 46](#_Toc382572155)

[5.6 Web Role responsible for Data Access 47](#_Toc382572156)

[5.7 Open Data Web Service 48](#_Toc382572157)

[5.8 Examples of using the service 49](#_Toc382572158)

[5.9 Protecting OData End Point 49](#_Toc382572159)

[5.10 Integrating wuth the UrbanWater Platform 50](#_Toc382572160)

[The OData web service will have to integrate with the UrbanWater platform being developed in WP6. The Urban Water platform which is based on RabbitMQ and wil TODO need to write this section with input form Andres. 50](#_Toc382572161)

[5.10 The Private Cloud – Key Management 51](#_Toc382572162)

[5.11 Elasticity in Data Management Platform 51](#_Toc382572163)

[5.12 Platform notifcations 52](#_Toc382572164)

[6 Data management Cloud Platform UML 53](#_Toc382572165)

[6.1 Storage Monitor Worker Role UML 53](#_Toc382572166)

[6.2 Data Processing Worker Role & Queue UML 53](#_Toc382572167)

[6.3 OData Web Role UML 54](#_Toc382572168)

[6.4 Data Management Platform Scalability UML 54](#_Toc382572169)

[6.5 Data Management Platform Security UML 56](#_Toc382572170)

[7 Conclusion and Outlook 57](#_Toc382572171)

[8 References 58](#_Toc382572172)

[9 Annex A 59](#_Toc382572173)

[A.1 PutBlock() 59](#_Toc382572174)

[A.2 PutBlockList() 62](#_Toc382572175)

[10 Annex B 68](#_Toc382572176)

[B.1 Reference Implementation to obtain a SWT from ACS 68](#_Toc382572177)

[B.2 Reference Implementation to upload data using PutBlock() 69](#_Toc382572178)

[11 Annex C 72](#_Toc382572179)

[C.1 File structure of Meter Readings 72](#_Toc382572180)

[12 Annex D 74](#_Toc382572181)

[D.1 Obtaining a SWT from ACS 74](#_Toc382572182)

[D.2 Token issuing endpoint 75](#_Toc382572183)

[D.3 Password token requests 75](#_Toc382572184)

[D.4 SWT token requests 77](#_Toc382572185)

[D.5 Claims assertion via the OAuth WRAP protocol 79](#_Toc382572186)

[D.6 Unwrapping and sending the token to a web application or service. 79](#_Toc382572187)

[D.7 ACS Error Codes and Descriptions 80](#_Toc382572188)

[13 Annex E 81](#_Toc382572189)

# List of Figures

[Figure 1: All currently supported platforms that built SDKs for OData based Development 28](#_Toc381178535)

[Figure 2: Full list of all supported options and operators on an OData WCF Data Service 29](#_Toc381178536)

[Figure 3: Claims based Authentication cycle 29](file:///C:\Users\Martin\Desktop\D3.2_Resubmit.docx#_Toc381178537)

[Figure 4: An overview of the data management cloud platform. 46](#_Toc381178538)

[Figure 5: This diagram shows the Azure Table Storage, a NO-SQL Database. 47](#_Toc381178539)

[Figure 6: This figure show Azure Blob Storage internal structure, highlighting containers in the storage account. 48](#_Toc381178540)

[Figure 7: This fugure shows Block nature of Block Blob Storage. 49](#_Toc381178541)

[Figure 8: Federated claims based authentication using ACS to obtain a SWT for authentication 56](#_Toc381178542)

# List of Tables

[Table 1 : Calendar, Hydrological and Water Distribution Network Data requirements for each partner in the UrbanWater project 12](#_Toc381190833)

[Table 2: Meteorological data requirements for each partner in the UrbanWater project 13](#_Toc381190834)

[Table 3: Comparison of both cloud platform providers 26](#_Toc381190835)

[Table 4: Meter Data Table 31](#_Toc381190836)

[Table MET 5: Node Geometries Azure Table 32](#_Toc381190837)

[Table MET 6: Line Geometries Azure Table 32](#_Toc381190838)

[Table MET 7: Junctions Azure Table 32](#_Toc381190839)

[Table MET 8: Reservoirs Azure Table 33](#_Toc381190840)

[Table MET 9: Tanks Azure Table 33](#_Toc381190841)

[Table MET 10: Types of Valves Azure Table 34](#_Toc381190842)

[Table MET 11: Valves Azure Table Azure Table 35](#_Toc381190843)

[Table MET 12: Pumps Azure Table 35](#_Toc381190844)

[Table MET 13: Constant Power Azure Table 36](#_Toc381190845)

[Table MET 14: Design Point Curve Azure Table 36](#_Toc381190846)

[Table MET 15: Exponential 3-Point Curve Azure Table 36](#_Toc381190847)

[Table MET 16: Pipes Azure Table 37](#_Toc381190848)

[Table 17: Sensor Azure Table 39](#_Toc381190849)

[Table 18: Product Azure Table 39](#_Toc381190850)

[Table 19: Assignment Data Azure Table 40](#_Toc381190851)

[Table 20: Record Data Azure Table 40](#_Toc381190852)

[Table 21: Raster Fields Azure Table 43](#_Toc381190853)

[Table 22: Raster Record Azure Table 44](#_Toc381190854)

[Table 23: Calendar Azure Table 44](#_Toc381190855)

[Table 24: Scalability in data management platform 57](#_Toc381190856)

# Introduction

## Objectives of UrbanWater

Improving the efficiency of water management in Europe is recognized as essential for overcoming the growing exposure of European countries to increasing populations and water scarcity and droughts.

The objective of the UrbanWater Project (<http://urbanwater-ict.eu/>) is to enable better end-to-end water management in developed regions, which accounts for 17% of freshwater demand in the European Union.

The project will develop an innovative Information & Communication Technology (ICT) based platform for the efficient, integrated management of water resources. The system will benefit consumers, water utilities, public authorities, the environment and the general public in terms of:

* providing consumers with comprehensive tools enabling them to use water more efficiently, thereby reducing overall consumption
* helping water utilities to meet demand at reduced costs
* fostering new partnerships between stakeholders so as to ensure the successful development of the system and the evolution of the European Water Sector as a global leader.

The UrbanWater system will likely incorporate;

* advanced metering solutions
* real-time communication of supply / demand data
* new data management technologies with real-time predictive capability
* supply / demand forecasting
* demand pattern interpretation
* decision support systems
* adaptive pricing
* user empowerment solutions

The UrbanWater consortium includes ICT companies, research organizations, water utilities and authorities with complementary capacities and know-how. Two water utilities included in the group will undertake large-scale validations on their water supply systems, thus promoting a final outcome that is close to the market and to the consumers. The final outcome of the project will remain open and interoperable with energy and water management schemes, thus positively impacting not only water consumption, but overall usage of natural resources throughout Europe.

## Objectives of WP 3 (Data management cloud platform)

This deliverable is part of the work in work package 3 – Data management cloud platform. The objectives of work package 3 are:

* Deploy a robust single cloud platform able to store and process large amounts of data meeting the needs for operating water smart meters.
* Develop the architecture and components required to be able to securely store and manage large volumes of data on behalf of the water and energy companies using the system, as well as to guarantee customers’ data privacy.

## Objectives of this document

This deliverable D3.2 “Data management cloud platform design” gives an overview of the proposed data management cloud platform design as part of the UrbanWater project. The UrbanWater data management platform will provide a cloud based data management platform that will store encrypted data. The platform will scale to meet demand during busy period and will provide secure data access. While every effort has been made to ensure accurate and through design all designs are subject to change as the project evolves.

## Structure of this document

This document is structured to give a brief introduction to the technology used in the cloud data management platform design and the design itself. Following this introduction, the report includes the following sections:

* Section 2 gives a brief review of the requirements gathered in task D3.1, which are used for the bed rock of this design document.
* Section 3 will give an introduction to the cloud platform technology chosen Microsoft’s Windows Azure and compare and contrast this technology with one of its leading competitors Amazons EC2. It also highlights of reason for choosing Microsoft Azure as our cloud platform provider.
* Section 4 introduces a lot of the technology that is used in securing the open data web service that will be provided and securing that web service. It also gives a brief introduction into Advanced Encryption standard (AES).
* Section 5 will give a brief overview of the data the platform is required to store, including the table structure that the data will be stored in Azure Table storage.
* Section 6 presents the data management cloud platform design. The design highlights the database technology, internal processing components and the OData Web Service.
* Section 7 present the data management cloud platform design UML.
* Appendix A presents a detailed overview of the API for uploading data to the cloud platform.
* Appendix B provides a reference implementation for uploading data and retrieving information from the cloud platform.
* Appendix C presents the structure of file uploaded to the platform form WP2.
* Appendix D provides detailed information for adhering to the platform security information including different methods for obtaining a Simple Web Token (SWT).

# Cloud Computing



## Cloud computing & the UrbanWater Project

The UrbanWater data management platform proposed under the UrbanWater project is based on harnessing the power of emerging cloud computing technologies and strategies for processing large amounts of data. This is evident in requirements RS-1, RS-2 “Storing all data requires for the UrbanWater project” and “Guarantee Secure data storage and access”.

Cloud computing is a very ambiguous term that has been moulded mainly by marketing department s for selling the what has been termed the next evolution of computing, however on further analysis the ability to perform complex distributed computing tasks can be traced back to the 1950s. Cloud computing can be defined as a general term that describes an application or service hosted over the internet or network.

Presently we are experiencing a huge uptake for in cloud computing from the enterprise sector as business what to off-set the cost of physical resources and their management. Companies can adopt a pay as you go approach where they only pay for the time their applications are hosted in the cloud environment and don’t have to support hardware outside of deployment.

## 2.2 Service Models

Cloud computing providers offer their services through three main service models:

* Infrastructure as a Service
* Platform as a Service
* Software as a Service

**Infrastructure as a Service (IaaS):** is the term given where software provides access to physical computers and storage or most commonly to virtual machines. In this model the consumer can install their own operating system and are responsible for its maintenance, deploys their application on top of it. This model varies for different Cloud service providers which may include load balancing and other software service to assist the customer.Examples of IaaS providers are: Microsoft Azure, Amazon EC2, RackSpace, HP Cloud and Google Compute Engine.

**Platform as a Service (PaaS):** in this service model the cloud computing provider provides the computing platform; this model can vary across different providers but always includes the operating system and can include database technologies, storage technologies and web servers. Examples of PaaS are similar to IaaS due to the fact the both methodologies are supported by both Microsoft azure and Amazon EC2.

**Software as a Service (SaaS**): this is sometimes referred to as software on demand, all underlying concerns are taken care of by the service provider from hardware to operating system. In this service model the user’s gets access to application via a client. SaaS can allow an Enterprise to reduce IT operational costs by outsourcing hardware and software maintenance and support to the cloud provider. This enables an Enterprise to reallocate IT operations costs away from hardware/software spending and personnel expenses, towards meeting other goals. There are many examples of SaaSsuch as Microsoft Office 365, Salesforce, and Google Apps etc.

## 2.3 Deployment Models

There are three main models by which a cloud based deployment can be deployed:

**Public cloud:**

The public cloud is where services are not contained in house and are accessed over a public network. This is by far the most common type of deployment where an organization can access applications/resources that are hosted and maintained on a remote site. These are generally considered less secure than private clouds because a third part provider would have access to the physical resource to where the application is hosted. However, the advantages are of using this approach is a complete pay as you go service.

**Private cloud:**

The private cloud uses the same technology and software as the public cloud but all applications and resources are hosted and managed locally(in house). This deployment model is considered very secure as all resources being used in the deployment are controlled locally and a third party is not involved.

**Hybrid cloud:**

A hybrid cloud is a combination of a public and a private cloud, the hybrid models can used for exposing existing service to wider community of user to strategies for protecting sensitive customer information as part of a wider system. Application can use a technique call cloud bursting when scaling to use public resources in times of high demand and scale back when demand is reduced.

## 2.4 Cloud Computing Technologies

There are a number of different cloud computing providers available but for the purposes of this design document we are only going to review the two main providers, Amazon Web Service and Microsoft Windows Azure.

## 2.5 Amazon Web Services

Amazon Web Services (AWS) is the name given to a collection of web services developed by amazon for developing cloud computing applications, the best known of these services are Amazon EC2 and Amazon S3 a complete reference can be found at [2].

A list of services provided by amazon is a follows:



### 2.5.1 Compute

The compute section of Amazon cloud platform can be broken into two main sections, Amazon Elastic Compute Cloud (EC2) and Amazon Elastic MapReduce (EMR). EC2 provides access to virtual machine which can scale are the most commonly used compute infrastructure offered by Amazon However Amazon also offers access to the Apache Hadoop to help in data mining large data sets that contain unstructured data-sets though EMR. EMR uses the Web Scale ability offered by EC2.



### 2.5.2 Networking

Networking or connection to the cloud environment is especially important when working with sensitive data in a cloud environment. Amazon offers what it calls a Virtual Private Cloud (VPC) where an end user can connect to an existing private cloud infrastructure using Virtual Private Network (VPN). Amazons also supports high through put of data to Amazon data centres using AWS Direct connect. Amazon also provides its own Domain Name Service (DNS) web service, it supports scalability and redundancy.

### 2.5.3 Content Delivery

Amazon CloudFront is a content delivery network (CDN) web service, it facilitates businesses and end users high data transfer speed and low data latency.

### 2.5.4 Storage & Content Delivery

AWS provides support for different types of data storage which are tailored to most storage scenarios encounters. Amazon Glacier provides low cost data storage that is ideal for data archive but would quickly become an expensive option when extensively queried for the data responses. The main storage used in AWS is Amazon Simple Storage Service (S3) it is simply web service based storage. Amazon also offers Elastic Block Storage (EBS) provides a BLOB storage account for EC2.

AWS Storage Gateway is a virtual application that is use for cloud based back up Storage; it is extensively used in a hybrid cloud deployment model. AWS uses an Import/Export feature to accelerate the movement of data in and out of AWS.

### 2.5.5 Database

As with other aspects of functionality, AWS provides support for multiple database technologies:

* Amazon Relational Database Service (RDS) provides a scalable virtual machine that is optimised for relational database such as SQL Server, Oracle, Informix and MySQL.
* Amazon ElastiCache is Amazons implementation of Mememcached, Mememcached is a generic distributed memory caching system, it is extensive used to speed database driven web sites.
* Amzon DynamoDB is a highly optimized NOSQL database it is fully managed and is designed for application where fast database response is critical.
* Amazon SimpleDB is design to facilitate end users in running queries on non-relational data, it is main design goal is offset the work of database Administrators.
* AWS Data Pipeline is a web service that enables user to develop complex fault tolerant data processing pipe lines. It enables the easy transport of data between the compute and storage in AWS.

### 3.5.6 Management

* AWS Identity and Access Management (IAM) is an authentication platform used in AWS to control access to services and resources. It allows administrators to impose permissions and rules on the access of data.
* Amazon CloudWatch: monitors resources and is used to collect metrics, which can be used scale the application.
* AWS Management Console (AWS Console): This is the main console of AWS where user can log into AWS and manage all aspect of their user account. From the console user can run application start roles or databases and control spending on their account.

### 3.5.7 App Services

AWS has a number of existing applications that can be used to add extra functionality application:

* Amazon CloudSearch: Enables users to quickly develop their own scalable search solution for application or web sites.
* Amazon DevPay: this is a billing service which provides application developers with the ability to manage order pipe lines.
* Amazon Elastic Transcoder:is a service that provides media transcoding ( converting media files from one format into another).
* Amazon Simple Email Service (SES) provides an email engine that can used to run and manage outbound email campaigns.
* Amazon Simple Queue Service (SQS) provides a queuing infrastructure that can help pull application together and provided scalability for services.
* Amazon Simple Workflow (SWF) is a workflow service that allows you to manage and coordinate task in your application, it provide a framework that allows easy development and scalability for applications.

## 2.6 Windows Azure

Windows Azure [1] is Microsoft’s cloud computing platform; it can be subdivided into three main components:

* Windows Azure
* Windows Azure platform AppFabric
* SQL Azure

### 2.6.1 Windows Azure

Windows Azure or windows Azure server is subdivided into two environments the compute and storage environment. The **compute environment** is responsible for processing request, its core components are subdivided into three categories:

* Web Roles
* Worker Roles
* Virtual Machine Roles

**Web Roles:**

Web Roles in Microsoft Azure run on Internet Information Service (IIS) web server that can be used to host front-end web applications. It can support both http and https.

**Worker Roles:**

Worker roles are generally used for asynchronous long running tasks that require no user input unlike the web roles. A common application deployment is to contain both web and worker roles, where all the processing work would be performed by the worker role making the application highly responsive to the user.

**Virtual Machine Roles:**

This role has been currently retired from Windows Azure, it used to allow a custom version of an operating system to be installed. However, it has been superseded by Windows Azure Virtual Machines which have extended the functionality of this former role but are not part of the compute environment.

The **storage environment** of the Windows Azure Server is again spilt into three categories of storage:

* BLOB Storage
* Table Storage
* Queue Storage

**Blob storage:**

BLOB Storage is a storage service that facilitates storing any arbitrary data such as video to binary data. There are two types of Blob storage, Page and Block Blob storage. When initializing a BLOB storage account the designer must specify which type of blob is being used and a container to which the blob is assigned. A container can contain one or more blobs are used a part of the namespace when addressing a specific BLOB.

**Block blobs:**

Block blobs are exactly what their name sounds like, they consist of blocks each with own unique block ID, uploading information to a series of block and them upload the list of block IDs where data has been written/uploaded. Blocks can be of different sizes but cannot exceed a limit of 4 MB. The maximum size of a Block Blob Storage account can be 200Gb but can only contain 50,000 Blocks, therefore each block would have to be at it max capacity of 4 MB.

Block Blobs with their block consistency are designed specifically design for parallel file upload which is a major advantage when dealing with uploading large files, however, data uploaded to each blob is not part of the blob until a list of the blobs IDs (where data has been uploaded) is passed up to the BLOB. Blocks can include an MD5 hash to verify successful completion of an upload; blocks can be uploaded in any order and can be overwritten by specifying the block ID.

**Page Blobs:**

Page Blob is a collection of 512 byte pages for random read and write operations. Uploads/Writes to page blobs are instantly part of the blob unlike Block Blobs, the maximum size of a page blob is 1TB.

**Azure Table Storage:**

Azure table storage is designed for storing information in tabular fashion with rows and columns. Table storage can be described as non-relational table storage. Tables store data as collections of entities. Entities are similar to rows. An entity has a primary key and a set of properties. A property is a name, typed-value pair, similar to a column. The Table service does not enforce any schema for tables, so two entities in the same table may have different sets of properties. Developers may choose to enforce a schema on the client side. A table may contain any number of entities.

An entity always has the following system properties: PartitionKey property, RowKey property and a Timestamp property. These system properties are automatically included for every entity in a table. The names of these properties are reserved and cannot be changed. The developer is responsible for inserting and updating the values of PartitionKey and RowKey. The server manages the value of Timestamp, which cannot be modified.

Tables are partitioned to support load balancing across storage nodes. A table's entities are organized by partition. A partition is a consecutive range of entities possessing the same partition key value. The partition key is a unique identifier for the partition within a given table, specified by the PartitionKey property. The partition key forms the first part of an entity's primary key. The partition key may be a string value up to 1 KB in size. You must include the PartitionKey property in every insert, update, and delete operation.

**Queue Storage**

Windows Azure Queue storage is a service for storing large numbers of messages that can be accessed from geographically different locations via authenticated calls using HTTP or HTTPS. A single queue message can be up to 64KB in size, a queue can contain millions of messages, up to the 100TB total capacity limit of a storage account. Common uses of Queue storage include:

* Creating a backlog of work to process asynchronously
* Passing messages from a Windows Azure Web role to a Windows Azure Worker role

### 2.6.2 Windows Azure platform AppFabric

Windows Azure platform AppFabric allows developers to integrate on-premises services to their cloud services, to secure cloud & on-premises services with new or existing security frameworks (identity based, active directory, etc.), cache Internet or other content. The AppFabric can be broken into two main categories:

* Service Bus
* Access Control

**Service Bus:**

Service Bus provides a hosted, secure and widely available infrastructure for widespread communication and large scale event distribution and event distribution. Service bus provides connectivity options for Windows Communication Foundation (WCF) and other service endpoints, which would otherwise be very difficult to reach. The service bus also provide a queuing mechanism which supports first in first out (FIFO), this supports the decoupling of cloud based applications.

**Access Control Service:**

Access Control Service (ACS) feature of AppFabric that enables an easy way to authenticate and authorize users who want to interact with your applications or services. It provides and identity service which can be used to create application support federated third party claims based authentication.

ACS has the following features

* Integration with Windows Identity Foundation (WIF)
* Support for popular web identity providers including Windows Live ID, Google, Yahoo, and Facebook
* Support for Active Directory Federation Services (AD FS) 2.0
* Support for OAuth 2.0 (draft 10), WS-Trust, and WS-Federation protocols
* Support for the SAML 1.1, SAML 2.0, and Simple Web Token (SWT) token formats
* Integrated and customizable Home Realm Discovery that allows users to choose their identity provider
* An Open Data Protocol (OData)-based management service that provides programmatic access to the ACS configuration
* A browser-based management portal that allows administrative access to the ACS configuration

**Windows Identity Foundation:**

Windows Identity Foundation (WIF) is a sub set of the .Net framework used for implementing claims based authentication in .Net Applications see section 4.3. WIF is just one part of Microsoft’s Federated Identity software family that implements the shared industry vision based on open standards. Federated Identity comprises three components: Active Directory® Federation Services (AD FS) 2.0, ACS, and WIF. These three components form the core of Microsoft’s new claims-based cloud identity and access platform. Claims-based identity can greatly simplify the authentication process for the user because he or she doesn't have to sign in multiple times to multiple applications. A single sign in creates the token which is then used to authenticate against multiple applications, or web sites. As well, because certain facts (claims) are packaged with the token, the user does not have to tell each individual application those facts repeatedly.

### 2.6.3 SQL Azure

SQL Azure is a specialized version of Microsoft SQL server relational database management system (RDBMS). It is specifically design to support relational queries against stored data which can be structured or unstructured documents.

### 2.6.3 Scalability & Redundancy in Azure

Windows azure provides the ability to scale both web and worker roles using both fixed and dynamic scalability. In general, queue-based data exchange represents the foundation of a reliable and highly scalable messaging architecture capable of supporting a range of powerful scenarios in the distributed computing environment. Whether it’s high-volume work dispatch or durable messaging, a message queuing technology can step in and provide first-class capabilities to address the different requirements for asynchronous communication at scale.

Windows Azure AutoScaling Application Block (WASABi) governs scalability in Azure, it driven by rules which are generated by the developer, reactive rules and constraint rules. Reactive rules fire based on metric or variables defined by the developer, while constraint rules set a value for the maximum and minimum number of instances for each role.

**Redundancy in Azure:**

Windows Azure can provided geographic redundancy for data, also data stored at Microsoft Data Centre will be replicated at least three times to prevent against any loss form hardware failures. It is important to consider redundancy when designing application in Windows Azure. When developing SaaS application and running an application in Azure data centres all of the underlying complexity associated with redundancy is managed by Microsoft.

## 2.7 Selection of Cloud computing technology

To give a brief overview and comparison between the two cloud platform providers we have compiled a table that illustrates the different technologies on offer from the cloud platform providers.

|  |  |  |
| --- | --- | --- |
|  | **Microsoft Windows Azure** | **Amazon Web Services** |
| Logo |  |  |
| Compute Instance | Extra Small, Small, Medium, Large, Extra Large | Micro, Small, Large, Extra Large |
| Compute VM | VM Role Instance | EC2(Elastic Compute Cloud) Instance |
| Scaling | Auto Scaling Blocks | Auto Scaling |
| Identity Management | AppFabric ACS (Access Control System) | IAM |
| Messaging | Service Bus | SNS(Simple Notification Services) |
| Caching | AppFabric Cache | Elastic Cache |
| CND | Azure CDN | Cloud Front |
| Storage Tables | Azure Table Storage | Simple DB |
| Storage Drive | Azure Drives | EBS(Elastic Block Storage) |
| Database | SQL Azure | MySQL, Oracle |
| Queues | Azure Queues | SQS(Simple Queue Services) |
| Connect | Azure Connect | VPC(Virtual Private Cloud) |
| Load Balancing | Azure Load Balance | Elastic Load Balance |
| Traffic Manager | Azure Traffic Manager | N/A |
| Monitoring Tools | AzureWatch(3rd Party) | CloudWatch |

Table 3: Comparison of both cloud platform providers

Amazon EC2 and Microsoft Azure are closely matched when it comes to services offered, with a slight lead held by EC2. However, it is generally recognized that Azure has a far superior **price performance** than amazon and aids **quick development** of applications. Azure also boasts a higher read speed over EC2 which will be a vital factor for dealing with the large data sets being produced at the Utilities. Scalability in Azure is seamless it’s a simple as changing a value in a configuration file, while EC2 is a little more involved. **Scalability** is a core requirement of the system as it will be required to grow through-out the project and meet the demand of more ICT providers.

From the requirements generated in task 3.1 of Work Package 3 we have selected Microsoft’s Windows Azure as our cloud platform provider. In requirement RS-2, Guarantee Secure data storage and access **Microsoft data centres** were best placed to facilitate this requirement. Microsoft has two data centres that can be used for UrbanWater Project validation in Scotland and Portugal, they are both located in North-western Europe. The primary Data centre selected for the UrbanWater project is located in Dublin while geo-graphic redundancy can be carried out together with the Noord-Holland, Netherlands data centre. Microsoft’s data centres are compliant with EU Data Protection Directive 95/46/EC and with ISO/IEC 27001:2005.

To sum up our reasons for choosing Microsoft Azure as our cloud platform provider are Price performance, quick development, scalability and access to Microsoft’s very secure data centres.

# Security & Open Data

## 3.0 Open Data Protocol

The requirements to store all data required by all partners in the UrbanWater platform and allow secure access to the data have driven use to consider using Web Services. Given that we do not want to confine our partners into one particular technology we require technology that can be used to query multiple different data sets, we have selected open data to meet these requirements.

The open data (OData) Protocol as described in [3] is an application-level protocol for interacting with data via Restful web services. It is pronounced “Oh data”. The protocol supports the description of data models and the editing and querying of data according to those models. It provides facilities for:

* Metadata: a machine-readable description of the data model exposed by a particular data provider.
* Data: sets of data entities and the relationships between them.
* Querying: requesting that the service perform a set of filtering and other transformations to its data then return the results.
* Editing: creating, editing, and deleting data.
* Operations: invoking custom logic
* Vocabularies: attaching custom semantics

The OData Protocol is different from other REST-based web service approaches in that it provides a uniform way to describe both the data and the data model. This improves semantic interoperability between systems and allows an ecosystem to emerge. The image below is a graphic of current SDKs available using OData protocol over http/https:



Figure : All currently supported platforms that built SDKs for OData based Development

All communication will be sent over https, (Hypertext Transfer Protocol Secure (HTTPS) is a communications protocol for secure communication over a computer network, with especially wide deployment on the Internet. Technically, it is not a protocol in and of itself; rather, it is the result of simply layering the Hypertext Transfer Protocol (HTTP) on top of the SSL/TLS protocol, thus adding the security capabilities of SSL/TLS to standard HTTP communications).

The Secure Socket Layer (SSL) is one of the most commonly used protocols to secure data in transit on an open network or the internet. Developed by Netscape, SSL uses a cryptographic system that uses public and private keys to encrypt data.

## 3.1 What is a WCF Data Service

WCF data service are ready made templates available with visual studio that gives us the possibility for creating ODATA complaint WCF services easily. It is built on top of WCF REST Services.

WCF Data services were earlier known as ADO.NET data service. This templates gives us the possibility of exposing any kind of data from the service using ODATA protocol. It is implemented on top of WCF REST services to provide ODATA complaint data access and it has most of the boilerplate functionality already in place. The developer just need to configure the data source and let the service template know what data needs to exposed and with what permissions. The figure below shows the OData query options available for a WCF Data Service.

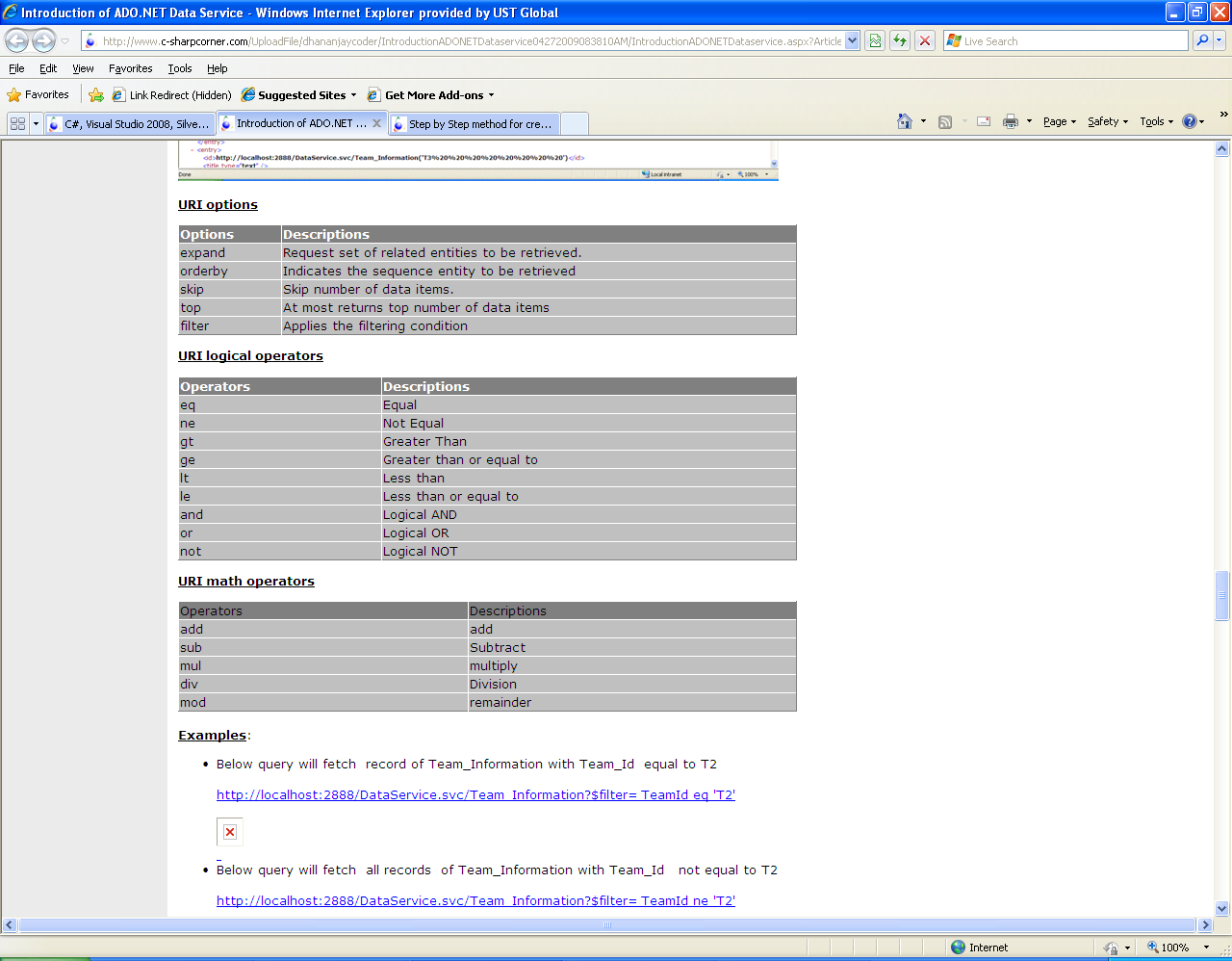


Figure : Full list of all supported options and operators on an OData WCF Data Service

## 3.2 Claims based Authentication

A “claim” can be described as a piece of information that describes a given identity on some aspect. Claims are held in an authentication token that can be signed to prevent against tampering. Claims based authentication is a mechanism where an external system operates in full trust with the system making claims about a user. The external system will receive an authentication token and will verify that this token is valid by contacting the claims system.

C:\Program Files (x86)\Microsoft Office\MEDIA\CAGCAT10\j0205582.wmf

Client

Service

Authentication

Request

Validate Token

Authentication Broker

Security Request with token

Figure : Claims based Authentication cycle

## 3.3 Oauth & SWT

OAuth is an open standard for authorization, it currently at version 2.0. ACS as discussed in section 3.6 issues tokens via the OAuth WRAP protocol. All token requests via the OAuth WRAP protocol are transmitted over SSL. ACS always issues a Simple Web Token (SWT) via the OAuth WRAP protocol, in response to a correctly formatted token request. All token requests via the OAuth WRAP protocol are sent to ACS in an HTTP POST. You can request an ACS token via the OAuth WRAP protocol from any platform that can make an HTTPS FORM POST: .NET Framework, Windows Communication Foundation (WCF), Silverlight, ASP.NET, Java, Python, Ruby, PHP, Flash, and other platforms. Simple Web Token (SWT) defines a format for transmitting a simple assertion that is compact and formatted to be easily included in a header for protocols such as HTTPS.

## 3.4 Certificate Authority

A Certificate Authority (CA) is a trusted entity or service that issue digital certificates. CAs are generally used in Public Key Infrastructure (PKI) schemes. These consist of a private and public key, this allows any third parties to trust any assertion made by the private key that corresponds to the public key that is certified.

## 3.5 Encryption (AES)

The Advanced Encryption Standard (AES) is a National Institute of Standards and Technology specification for the encryption of electronic data [4]. It is expected to become the accepted means of encrypting digital information, including financial, telecommunications, and government data. AES is a cryptographic algorithm that can be used to protect electronic data. Specifically, AES is an iterative, symmetric-key block cipher that can use keys of 128, 192, and 256 bits, and encrypts and decrypts data in blocks of 128 bits (16 bytes). Unlike public-key ciphers, which use a pair of keys, symmetric-key ciphers use the same key to encrypt and decrypt data. Encrypted data returned by block ciphers have the same number of bits that the input data had. Iterative ciphers use a loop structure that repeatedly performs permutations and substitutions of the input data.

AES is the successor to the older Data Encryption Standard (DES). DES was approved as a Federal standard in 1977 and remained viable until 1998 when a combination of advances in hardware, software, and cryptanalysis theory allowed a DES-encrypted message to be decrypted in 56 hours. Since that time numerous other successful attacks on DES-encrypted data have been made and DES is now considered past its useful lifetime.

In 1999, the Rijndael algorithm, created by researchers Joan Daemen and Vincent Rijmen, was selected by the NIST as the proposal that best met the design criteria of security, implementation efficiency, versatility, and simplicity. Although the terms AES and Rijndael are sometimes used interchangeably, they are distinct. AES is widely expected to become the de facto standard for encrypting all forms of electronic data including data used in commercial applications such as banking and financial transactions, telecommunications, and private and Federal information.

# UrbanWater Data

The UrbanWater project is made up of multiple partners performing value added work for the Utilities. A complete list of the partners and the technology/services they are offering Utilities are available in D3.1 Data management cloud platform requirements. In order to achieve their goals they require data from a variety of different sources again a complete list can be found D3.1. The Utility can provide most of the data in addition to this they require data from local Meteorological offices.

The data management cloud platform will have to receive data form Utilities and other data providers and then has to process this data and deliver data in a format that the project partners can use in their applications. This chapter will mainly focus on the how Azure tables described in the design chapter 5 will host data required for the UrbanWater project.The data is broken into seven Water Network Meter Data, Topology Data, Meteorological Data, Ratser Field Data, Calendar Data, Data storage for models and Customer & Tariff data (customer and tariff information will not be stored in the cloud )



## Water Network Meters Data

As described in chapter 5, data coming from the water meter network will be stored in the following format: Meter ID, Time Stamp, Encrypted Readings and stored in Azure table storage. This data will be updated every four hours with new information coming from the Utility and the Smart Gateway being developed in WP2.

Table 4: Meter Data Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| Meter ID | String | ID of the Water Meter |
| Meter Time | DateTime (UTC) | Time of reading taken |
| Encrypted Reading | String | Encrypted reading of the meter |

## Topology Data

Topology can be considered a static data-set in that if may change slowly over time with the expansion of the Utilities network but is not subject to daily updates or changes. This data will also be stored in Azure table storage. Water supply networks geometry can be derived from basic geometries like points and lines. The attributes of these basic geometries are described next. In the implementation of UrbanWater’s data model, these geometries should be based on already existing open source libraries in order to take advantage of already existing spatial operations. For the implementation already done, the open source library NetTopologySuite has been used.

### 5.2.1 Node Geometry

Nodes are the first basic element of a water distribution network. EPA NET based models describe junctions, tanks, reservoirs, pumps and valves as nodes. Next table shows the attributes for nodes.

Table MET 5: Node Geometries Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| ID | String | ID of the Node |
| X | Double | X Coordinate of the Node |
| Y | Double | Y Coordinate of the Node |

### 5.2.1 Line Geometry

Lines are the second basic element of a water distribution network. EPA NET based models describe pipes as lines, or more accurately as LineStrings. Next table shows the attributes for lines.

Table MET 6: Line Geometries Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| ID | String | ID of the Line |
| UpStream Node | Node | Upstream Node of the Line |
| DownStream Node | Node | Downstream Node of the Line |
| LineString | Vertex Array | Array of ordered vertex which describe the line geometry. |

**Model Elements**

The next sections describe the basic modelling elements which are usually found in a water supply network and appear in EPA Net based models.

### 5.2.3 Junctions

Junctions are node elements. Next table shows the properties for junctions.

Table MET 7: Junctions Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| Description | String | Description of the junction (optional) |
| Elevation | Double | Physical Elevation of the junction |
| Zone | String | Pressure zone associated to the reservoir |
| Demand | Time Series | Demand at the node (Please see Time Series for more details) |

### 5.2.4 Reservoirs

Reservoirs are node elements. In EPA Net based models they are used as boundary condition having unlimited volume and are generally used to represent a lake or other inexhaustible supply source. Next table shows the properties for reservoirs.

Table MET 8: Reservoirs Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| Description | String | Description of the Reservoir |
| Elevation | Double | Physical Elevation of the Reservoir (NOTE: If variable in time, this can be a Time Series) |
| Zone | String | Pressure zone associated to the reservoir |

### 5.2.5 Tanks

Tanks are node elements. In EPA Net based models distinguished from reservoirs as having a known finite volume and water surface levels change with time (EPS only) as water flows into or out of them.

Table MET 9: Tanks Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| Description | String | Description of the Tank |
| Elevation | Double | Physical Elevation of the base of the tank |
| Minimum Level | Double | Minimum Storage Level |
| Maximum Level | Double | Maximum Storage Level |
| Tank Section Type | Enum | Cylindrical or Variable Area |
| Tank Diameter | Double | Diameter of the Tank (if “Cylindrical”) |
| Tank Curve | Curve | Relation Volume / Level (if “Variable Area”) |
| Zone | String | Pressure zone associated to the tank |

### 5.2.6 Valves

In EPA NET based models a set of valves can be considered. Next table describe the most common valve types and their required attributed.

Table MET 10: Types of Valves Azure Table

|  |  |
| --- | --- |
| Valve Type | Required Fields |
| Pressure Reducing Valves (PRV) | Elevation - Specify the elevation of the valve.  Diameter - Diameter of valve  Setting - Downstream pressure setting |
| Pressure Sustaining Valves (PSV) | Elevation - Specify the elevation of the valve.  Diameter - Diameter of valve  Setting - Downstream pressure setting |
| Pressure Breaker Valves (PBV) | Elevation - Specify the elevation of the valve.  Diameter - Diameter of valve  Setting - Downstream pressure setting |
| Flow Control Valves (FCV) | Elevation - Specify the elevation of the valve.  Diameter - Diameter of valve  Setting - Design flow (through valve) |
| Throttle Control Valves (TCV) | Setting - %Open (with curve identified), K value (no curve)  Diameter - Diameter of valve  Curve (Optional) - Curve ID |
| General Purpose Valves (GPV) | Elevation - Specify the elevation of the valve.  Diameter - Diameter of the valve  Curve - Curve that represents the GPV. |

Valves are also node elements. Next table shows attributes for valves.

Table MET 11: Valves Azure Table Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| Description | String | Description of the Valve |
| Elevation | Double | Physical elevation of the valve |
| Diameter | Double | Diameter of the valve |
| Type | Enum | Type of the valve. See table above |
| Setting | Double | Setting of the valve. Meaning depends on the valve type |
| Minor Loss Coefficient | Double | Hydraulic Loss Coefficient |
| Zone | String | Pressure zone associated with the Valve |
| Valve Curve | Curve | Applicable to GPV (Head loss vs. Flow) |

### 5.2.7 Pumps

Pumps are also node elements. Different pumps are characterized of their specific pump curve (see below). Next table shows the attributes for pumps.

Table MET 12: Pumps Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| Description | String | Description of the Pump |
| Elevation | Double | Physical elevation of the pump |
| Pump Definition | Pump Definition | Characteristics of the Pump |
| Zone | String | Pressure zone associated with the Pump |

**Pump definition**

There are different types of pump definitions which can be found in EPA NET based models. Next tables show the attributes for those definitions.

**Constant Power Input**

Table MET 13: Constant Power Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| Elevation | Double | Water Elevation of the pump |
| Diameter | Double | Diameter of the pump |
| Power | Double | Power rating of the pump. |

**Design Point Curve**

Table MET 14: Design Point Curve Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| Elevation | Double | Water Elevation of the pump |
| Diameter | Double | Diameter of the pump |
| Design Head | Double | The design operating head |
| Design Head | Double | The design flow |

**Exponential 3- Point Curve**

Table MET 15: Exponential 3-Point Curve Azure Table

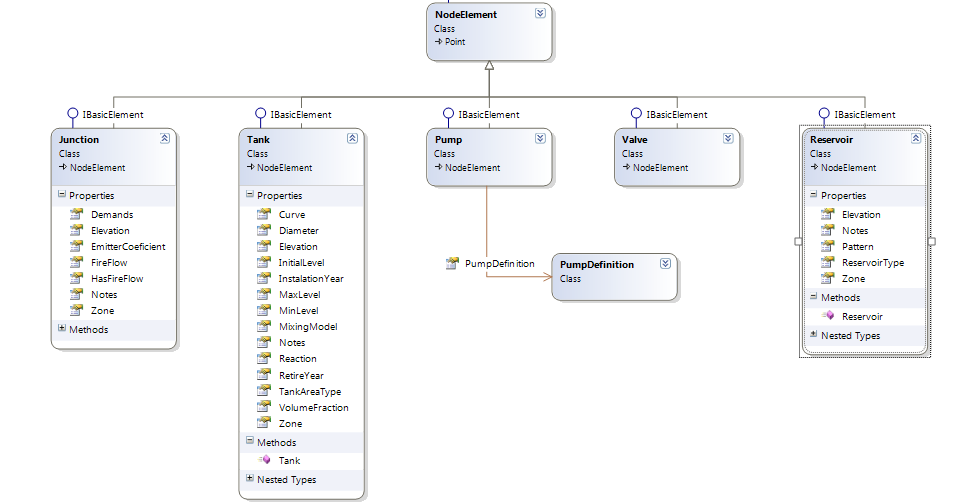
|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| Elevation | Double | Water Elevation of the pump |
| Diameter | Double | Diameter of the pump |
| Design Head | Double | The design operating head |
| Design Head | Double | The design flow |
| Shutoff Head | Double | The shutoff head at zero flow |
| High Head | Double | The head at the upper end of normal operating flow range |
| High Flow | Double | The flow at the upper end of normal operating flow range |

### 5.2.8 Pipes

Pipes are line string elements. Characteristics of pipes are presented in the next table.

Table MET 16: Pipes Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| Description | String | Description of the pipe |
| Material | String | Material of the pipe |
| Length | Double | Length of the pipe |
| Rugosity | Double | Rugosity coefficient of the pipe |

**Class Diagram**

## Meteorological Data (Non Raster Field Data)

Meteorological data will be divided into four Azure tables Sensor Data, Product Data, Assignment Data Table and Record Data Table.

### 5.3.1 Sensor Data Table

Table 17: Sensor Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| ID | String | The identifier used within UrbanWater to define this sensor |
| X | Double | X Coordinate of the Node |
| Y  Supplier  Code  Type | Double  String  String  String | Y Coordinate of the Node  The code that SW or TAVE use to identify this sensor  Utility code used to identify device  Type of the sensor (rain gage, flow meter, water level meter ...). Note that with the defined structure we could store not only meteorological information but also smart-meter readings, water consumptions, ... |

### 5.3.2 Product Data Table

This table stores the information of which variables could measure a sensor.

Table 18: Product Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| ID | String | The identifier of each variable |
| Name | String | Name of the Product |
| Unit | Double | The unit of the Product |
|  |  |  |

Examples of products are:

* Temperature / ºC
* Flow / m³/s
* Consumption / l

### 5.3.3 Assignment Data Table

This table relates products and sensors. With these assignments the system is able to define which variables are measured by which sensor.

Table 19: Assignment Data Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| ID | String | The field that identifies one assignment |
| Sensor ID | Double | The ID of the sensor assigned to a product |
| Product ID | Double | The ID of the product assigned to a sensor |

### 5.3.4 Record Data Table

This table stores not only measured information but predicted. Thus it could even store predicted demands in one point of the distribution network for instance.

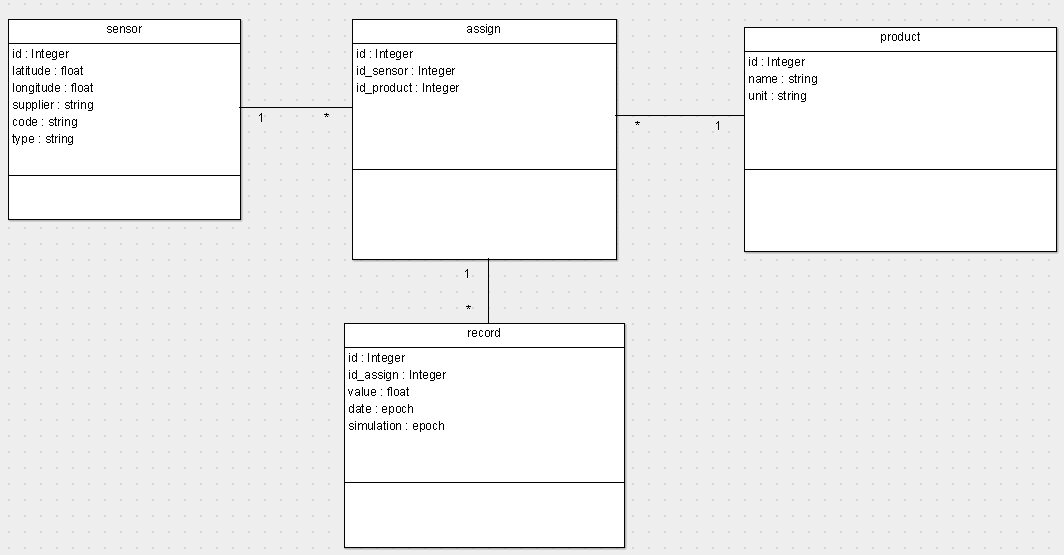
Table 20: Record Data Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| ID | String | An identifier of each record |
| ID\_Assign | Double | The ID of the sensor assigned to a product |
| Date  Simulation  Value | DateTime(UTC)  String  Float | The instant of this measurement. In the case the value is from a prediction, it means the instant of the simulation.  The instant from the prediction is calculated. In case of measurements, this value should be null  The value itself |

### 5.3.5 Example

* Sensor:
  + ID: 1432
  + Latitude: 54.32
  + Longitude: -5.24
  + Supplier: SW
  + Code: MO12220
  + Type: rain gage
* Product:
  + ID: 243
  + Name: 1h accumulation
  + Unit: mm
* Assign:
  + ID: 65
  + ID\_Sensor: 1432
  + ID\_Product: 243
* Record:
  + ID: 342423
  + ID\_Assign: 65
  + Value: 12.3
  + Date: 1392811200 (2014/02/19 12:00:00 UTC)
  + Simulation: null

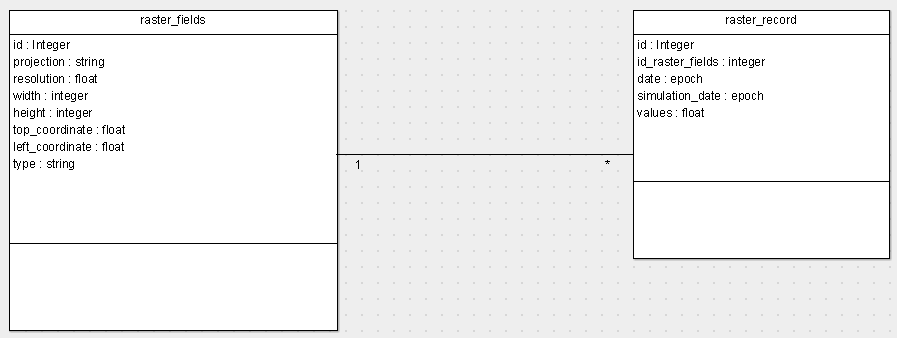
Class Diagram



## Raster Field Data

This section describes how raster fields could be stored in the database. This raster fields can be in UrbanWater:

* Rasters containing radar reflectivity
* Rasters containing prediction data fields (from numerical models)



This two tables are used for describe how data is generated (raster\_fields), and the information record itself at every timestep (raster\_record).

### 5.4.1 Raster fields Data Table

Table 21: Raster Fields Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| ID | Int | The integer of one raster field type. |
| Width | Double | Width of the field stored in the matrix |
| Height | Double | Height of the field stored in the matrix |
| Top\_Coordinates | String | Top coordinate of the field |
| Left\_Coordinates | String | Left Coordinate of the field |
| Projection  Resolution  Type | String  String  String | Definition of the projection used by the field (e.g. SRID: 4326 -latlon- , 23030 -UTM-, 900913 -Mercator/Google-)  Resolution of the field in degrees or meters per pixel, depending on the projection  Type of information (temperature, 1h accumulation, …) |
|  |  |  |

### 5.4.2 Raster Record Data Table

Table 22: Raster Record Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| ID | String | The identifier of one field |
| ID\_raster\_fields | String | The ID of the raster\_field type |
| Date | Double | The instant when this field was measured |
| Simulation | DateTime(UTC) | The instant from when simulation was calculated. In case of measurements this field is null |
| Values | Array | A matrix containing the cell values. |

In the case of radar reflectivity fields, simulation field will be null. In case of prediction models data, simulation will indicate when the simulation begins (e.g. today at 00:00), and date indicates of which instant of the simulation the field is (e.g. accumulation field between 12:00 and 13:00).

## Calendar Data

Calendar data can with notice of special events will be stored in Azure table storage under one table.

Table 23: Calendar Azure Table

|  |  |  |
| --- | --- | --- |
| Attribute | Data Type | Description |
| Date  Description | String  String | Date of the event  Special Event Description |
| Start Time  End Time | DateTime(UTC)  DateTime(UTC) | Start time of the event  End Time of the event |

## Private Model Results Data

These will bestored on the Azure Blob Storage see section 6.3, where these private results will not be shared with the rest of the consortium. Connection will be secured with a connection.

## Customer & Tariff Data

All sensitive customer data will not be stored in the cloud it will be acquired and stored by the billing and adaptive pricing systems.

# Data management Cloud Platform Design

## 5.0 Overview

This chapter will introduce the design of data management cloud platfrom, which main objective is store data from the utilities and from the data gathered by the head end system in WP2. It will make this data available to all partners through the urbanWater Platform being developed in WP6, the platform will also privide a faciltate for private storage for Modules or applications that require such functionality.

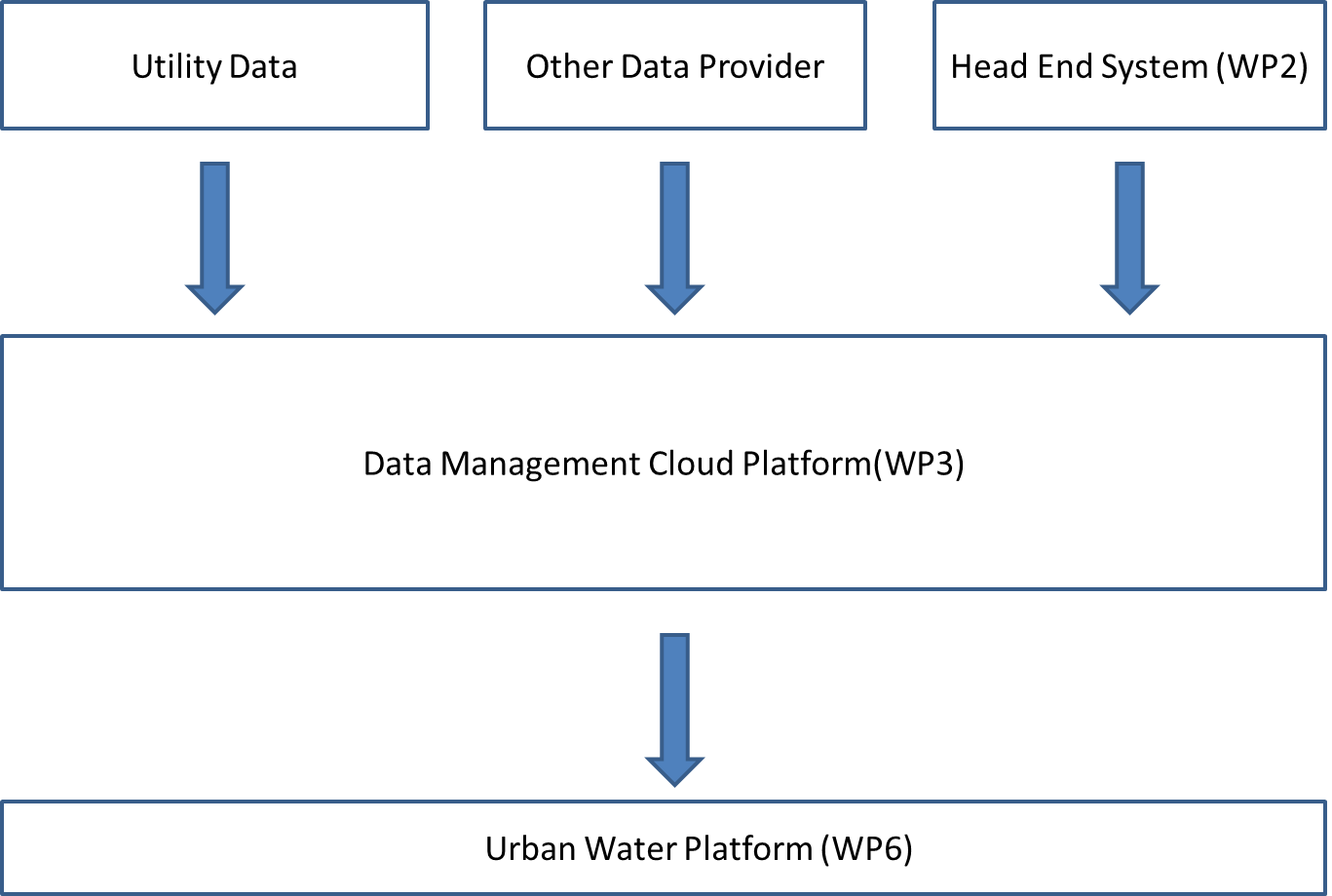


Figure : This figure illustrates where the cloud platform fits into the overall UrbanWater design.

The design of the UrbanWater data management cloud platform is based on a hybrid cloud (discussed in 3.3 Deployment models). As we have requirements to guarantee data security, all data stored on the public cloud of sensitive nature must be encrypted as we are using a public cloud provider Microsoft’s Windows Azure (discussed in chapter 2 of this document).

A hybrid cloud consist of both a public and a private cloud, for the data management platform we are using Microsoft’s SQL Server for key management of the encryption keys used to encrypt the meter readings that will be stored on the public cloud. All data will be encrypted at the utilities before uploading to Microsoft Azure data centre located in Dublin, Ireland. Data encryption will be performed using AES 128 bit encryption and the encryption keys will be stored at each utility in a private cloud (please see section 4.5 for information on AES). In order for the data to be decrypted by the UrbanWater platform being developed in WP6, data will have to be retrieved from the Azure platform via Web Service and the Key needed to decrypt this value will have to be obtained from the private cloud located at each utility. We able to implement this approach by using a very simplistic data model where readings are encrypted and meta-data is not encrypted e.g. Meter ID, Time Stamp and Encrypted Reading.

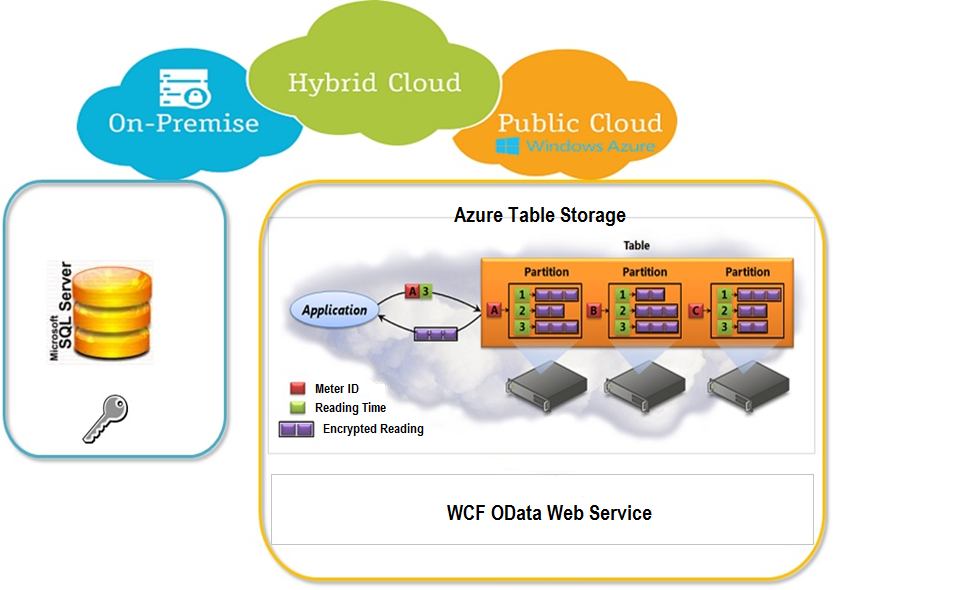


Figure 5: An overview of the UrbanWater data management cloud platform.

Data will be stored on the Windows Azure public cloud platform in Azure table storage database as described in the next section. Data will be made available the end user (UrbanWater platform) WP6, by an Open Data (OData) Web Service (see section 4.1).

Our private cloud which consists of a key management database (Microsoft SQL Server) will be stored at each utility, where physical access to the machine will be restricted. Connection strings will be made available to the UrbanWater platform (WP6) and to interested third parties that are approved by the utility, to facilitate the decryption of the encrypted data stored on the public cloud. This deployment model will allow Utilities to exert more control on their data-sets, where they can physically make all data stored on public cloud redundant by switching off the key management database stored at the Utility.

## 5.1 Azure Table Storage – NO-SQL Database

For storing Meter Readings needed by the partners in the UrbanWater project from the Water Distribution Network we are using Microsoft’s Windows Azure Table storage (see section 3.6.1) which is a NO-SQL database. Azure table storage is optimized to store time series data like the data needed for the UrbanWater Project.

Windows Azure table Storage contains two main properties the PartitionKey and the RowKey, the PartitionKey and RowKey properties work together as an index for your table, so when defining them, you must consider how your data is queried. Together, the properties also provide for uniqueness, acting as a primary key for the row. Each entity in a table must have a unique PartitionKey/RowKey combination.

In the cloud data management platform the Partition Key will be the meter ID, the RowKey will be the time of the meter reading and a third column will feature the encrypted meter reading.

Meter ID(PartitionKey), Meter Reading Time(RowKey), Encrypted Meter Reading(Entity)

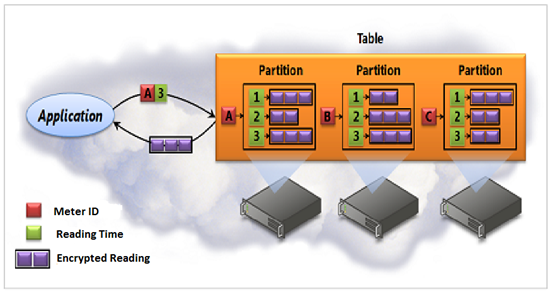


Figure : This diagram shows the Azure Table Storage, a NO-SQL Database.

By selecting Meter ID as the PartitionKey this will allow our database to scale/partition across multiple disks in the data centre and to respond quickly to queries on the dataset.

Geographic Redundancy of the data-sets can easily be achieved between Dublin, Ireland and Noord-Holland, Netherlands, keeping all copies of the data sets stored in the EU by using region locking. Region locking is a feature in Microsoft Windows Azure that allows companies or cloud application developers to specific if they want their data-sets to be confined to specified regions.

Two independent Azure storage accounts will be used for validation in both Scotland and Portugal, data from both sites will never be stored in the same storage accounts. The following is a break-down of how the rest of the data required by the partners will be stored in the data management cloud based platform not just information form Water Meter Network.

## 5.2 Integrating with Data Providers(Utilities and Head End System)

The cloud data management platform will have to support upload of data not just from the Multiprotocol Gateway system being developed in WP2 but from the utilities partners and other partners wishing to share data with the platform. As part of the internal design of the public cloud platform it will have to accept encrypted readings from the utilities containing meter readings. The cloud data management platform facilitates this by using containers in Azure BLOB storage described in section 3.6.1. In Azure BLOB storage, the storage account can be partition into sections known as containers; the data management platform uses this feature to separate out different required data-sets while keeping a copy of the original uploaded file.

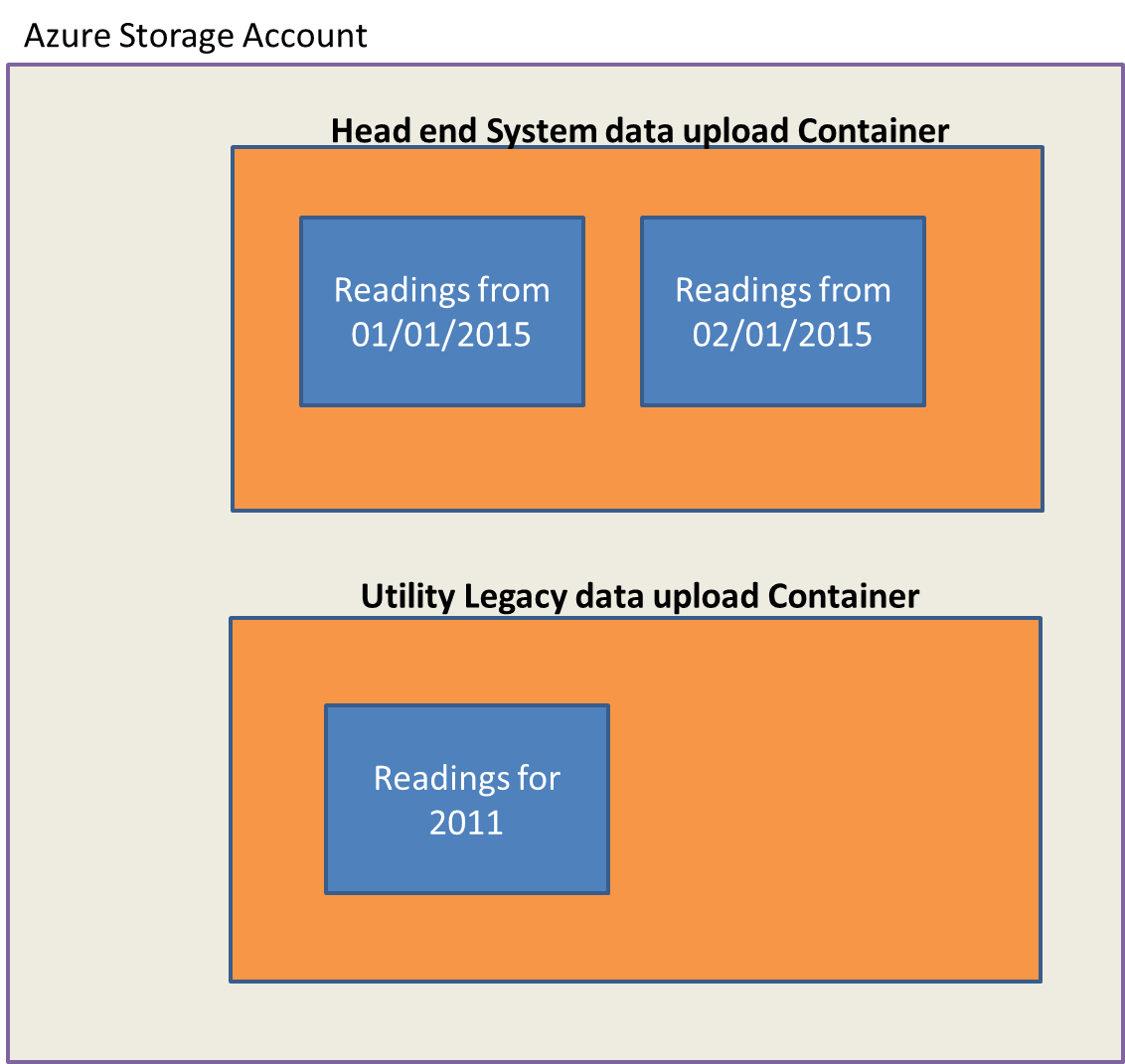
**

Figure : This figure show Azure Blob Storage internal structure, highlighting containers in the storage account.

The above figure shows the internal structure of a Blob Storage account in Microsoft Azure. The platform will have multiple Containers in its storage account to facilitate each partner uploading data to Azure Storage. These containers will be mainly used for uploading meter Readings to the platform from the Multi-Protocol Gateway system being developed in WP2 and from the Utilities for legacy meter data and meters not connected to the Gateway system. Using containers allow the platform to support data uploads of different format each container has own address and will allow the platform (which will constantly monitor these containers for file uploads) **to support uploads of different formats and different types of data**.

Container will also allow the data management platform to provide private storage to the project partners where they will have a connection string for connecting to the container please see Appendix B for sample code to integrate. This will allow private model results to be stored by partners and not shared with the rest of the consortium. We use a Block BLOB Storage account in order to facilitate parallel and partial file upload. Block Blobs are organized in small 4MBytes blocks that can be written to simultaneously making this an ideal storage account for uploading files simultaneously. The REST based API for uploading files to Block Blob Storage along with a reference implementation is documented in Appendix A&B respectively using PutBlock and PutBlockList Methods.



Figure : This figure shows Block nature of Block Blob Storage.

## 5.3 Data management “public” cloud platform data flow cycle

The cloud based data mangaemnt system flow diagram detailing the processing components, that will facilitate the processing of large number of file uploads to different continers in Block Blob Storage:

****

**Azure Blob**

**Storage**

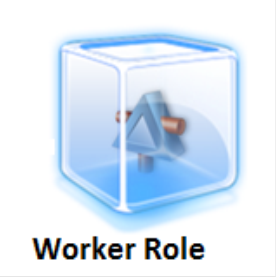
**Data Provider**

****

****

**UrbanWater Data Management**

**Platform Flow Cycle**

****

**Azure Table**

**Storage**

****

**Step 1:**

Data providers can upload files to the cloud data management platforms storage through a Rest API optimized for partial/parallel file streaming over SSL via https. This is done by calling PutBlock() and PutBlocklist() API methods a sample implementation and detail documentation on this API can be found in **Appendix B & A respectively**. All data being uploaded using the HeadEnd system will be encrypted only Meta data will be not be encrypted, therefore, there will no requirement on the system to encrypt this data.

**Step 2:**

A worker role A in the UrbanWater platform will monitor this storage account in Azure. The worker role will detect when a new file has finished uploading based on an ETAG property associated with each BLOB. This worker role will also acts as an inventory of each file uploaded to the Block BLOB.

**Step 3:**

Worker Role A will then add each new file entry to the UrbanWater Queue for processing, Queue storage is replicated at least three times in an Azure data Centre. All messages added to the queue will be on a first in first out (FIFO) basis.

**Step 4:**

Worker role B monitors the queue and acts when a message is added.

**Step 5:**

Worker role B based on the received queue message will process each file added to BLOB Storage and add all data to Azure data tables after doing some validation on each reading and store it in the correct table. The Azure table storage account will be partitioned on the basic of n number of meters per table; this will allow the platform to scale to meet the demand of adding millions of meters and each meter contains a large number of readings. Meter Reading data will be stored in the format Meter ID, Time of Reading and Encrypted Readings.

**NB:** All data stored in the “cloud” will be encrypted.

## 5.4 Worker Role for Storage Monitoring

As shown in section 6.2 the data management platform will monitor several storage containers in BLOB storage, in order to check when new files have been uploaded. This role will keep an internal list of all files uploaded for the project written to another storage account containing Azure Table Storage database. Each file once it has been detected will have it ETag property value recorded, the ETag property is updated on write operations to the blob. When the file is fully uploaded the ETag property will remain constant. The data management platform uses this feature of Blob Storage to determine when a file have been fully uploaded and adds a message to the data management queue that a new file is available for processing.

## 5.5 Worker Role for Data Processing

As the data management platform will have the ability to receive different files contain different data and different file formats, it will gave to process these files in order to add the individual readings contained in each file to Azure Table Storage database. These worker roles for file processing will act on messages from queue storage shown in section 6.3, each message in the queue contains the following information: file owner, file format and file location (storage container where the file has been uploaded). The processing role(s) will read the file line by line in order to be able to process the large files, adding each entry to Azure Table Storage database.

## 5.6 Web Role responsible for Data Access

Providing data access to other partners in the UrbanWater project is one of the primary requirements of the data management platform as multiple partners require different and distributed data sets. The data management platform will allow Utilities and third parties such as regional Met offices to upload data. We make data available using Open Data protocol to allow partners and other agreed third parties to write and run flexible queries on the data sets. OData WCF Data services are discussed in more detail in section 4.1.

****

**Azure Table**

**Storage**



**UrbanWater**

**Platform (WP6)**

**Step 1:**

Azure table storage uses a web role host in Microsoft’s Windows Azure to provide an OData interphase the web role will allow queries to run against a table context object. The table will return a type of IQuerable.

**Step 2:**

The web role will be accessible from the Urban Water Platform being developed in Work Package 6. The platform will be able to use OData protocol to query the WCF web service.

## 5.7 Open Data Web Service

The following is an EDMX document of the data management platform OData web service; an .edmx file is an XML file that defines a conceptual model, a storage model, and the mapping between these models. This will generate much more helpful information about the web service than a WADL because we are developing OData Web Service.

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx">

<edmx:DataServices xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" m:DataServiceVersion="1.0">

<Schema Namespace="Microsoft.WindowsAzure.StorageClient" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://schemas.microsoft.com/ado/2006/04/edm">

<EntityType Name="TableServiceEntity" Abstract="true">

<Key>

<PropertyRef Name="PartitionKey" />

<PropertyRef Name="RowKey" />

</Key>

<Property Name="Timestamp" Type="Edm.DateTime" Nullable="false" />

<Property Name="PartitionKey" Type="Edm.String" Nullable="false" />

<Property Name="RowKey" Type="Edm.String" Nullable="false" />

</EntityType>

</Schema>

<Schema Namespace="UrbanWaterData" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://schemas.microsoft.com/ado/2006/04/edm">

<EntityType Name="MeterReadingEntity" BaseType="Microsoft.WindowsAzure.StorageClient.TableServiceEntity">

<Property Name="MeterID" Type="Edm.String" Nullable="true" />

<Property Name="CreatedOn" Type="Edm.DateTime" Nullable="false" />

<Property Name="EncryptedReadings" Type="Edm.String" Nullable="true" />

</EntityType>

<EntityContainer Name="MeterReadingTableStorageContext" m:IsDefaultEntityContainer="true">

<EntitySet Name="MeterReadings" EntityType="UrbanWaterData.MeterReadingEntity" />

</EntityContainer>

</Schema>

</edmx:DataServices>

</edmx:Edmx>

## 5.8 Examples of using the service

**Get all Meter readings from a specific meter with id “2100723”:**

http://urbanwater.cloudapp.net/WaterMeterData.svc/MeterReadings?$filter=PartitionKey eq '2100723'

**Get all Meter readings from a specific meter with id “2100723” and meter time is '2004-05-02T01:00:00Z'**

http://urbanwater.cloudapp.net/WaterMeterData.svc/MeterReadings?$filter=PartitionKey eq '2100723' and RowKey eq '2004-05-02T01:00:00Z'

**Get all Meter readings from a specific meter with id “2100723” and meter time is greater than (after) '2004-05-02T01:00:00Z'**

http://urbanwater.cloudapp.net/WaterMeterData.svc/MeterReadings?$filter=PartitionKey eq '2100723' and RowKey gt '2004-05-02T01:00:00Z'

**Get all Meter readings from a specific meter with id “2100723” and meter time is less than (before) '2004-05-02T01:00:00Z'**

http://urbanwater.cloudapp.net/WaterMeterData.svc/MeterReadings?$filter=PartitionKey eq '2100723' and RowKey lt '2004-05-02T01:00:00Z'

**Get all Meter readings from a specific meter with id “2100723” and meter times between '2004-05-02T01:00:00Z' and '2004-05-02T02:00:00Z'**

http://urbanwater.cloudapp.net/WaterMeterData.svc/MeterReadings?$filter=PartitionKey eq '2100723' and RowKey gt '2004-05-02T01:00:00Z' and RowKey lt '2004-05-02T01:00:Z’

**A demonstration of the web service and data management platform is available from** <http://urbanwater.cloudapp.net/WaterMeterData.svc/>.

However, in order to gain access to the encrypted data readings one can use the sample code provided in Appendix B to adhere to the security model described in the next section.

## 5.9 Protecting OData End Point

To protect out OData Web Service end point we use Windows Azure Access Control Service (ACS) for federated claims based authentication described in section 4.6.2. Credentials are supplied using https to obtain a Simple Web Token (SWT) as described in section 4.4 using OAuth 2.0 protocol. A detailed example of how to obtain a SWT from ACS is available in Appendix D.

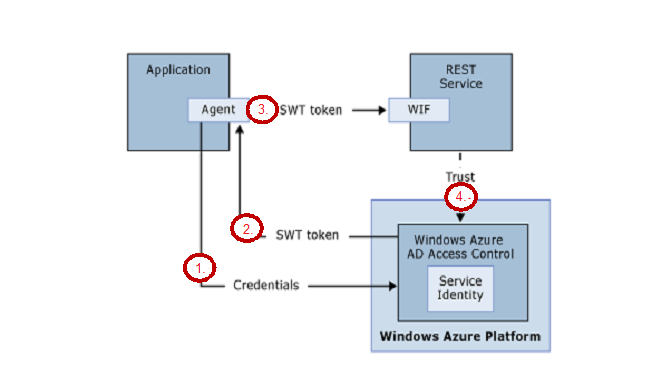
****

Figure : Federated claims based authentication using ACS to obtain a SWT for authentication

**Step 1:**

The UrbanWater platform being developed in WP6 will authenticate using credentials against the Windows Azure Access Control (ACS). This security model is based on federated claims based authentication described in section 4.3.

**Step 2:**

The UrbanWater platform will receive a limited life-span Simple Web Token (SWT) via Oauth (2.0) WRAP protocol described in section 4.4.

**Step 3:**

This token will passed in the web service headers call to the UrbanWater data management platform which is integrated with Windows Identification Foundation (WIF).

**Step 4:**

The data management platform will pass the SWT to ACS to verify the token. Once verified tokens can be used for duration of its lifetime in headers of data requests. The life time set of this project is approximately four hours. The token can be passed in the header of the OData web service.

## 5.10 Integrating wuth the UrbanWater Platform

The OData web service will have to integrate with the UrbanWater platform being developed in WP6. The Urban Water platform which is based on RabbitMQ and wil TODO need to write this section with input form Andres.

## 5.10 The Private Cloud – Key Management

In order to garauntee data security its is neccessary all data in the public cloud (Windows Azures Dublin data center) will be encrypted.In order to achieve this design objective it will be necessary to encrypt all data at the utility or on the domestic meter itself. While the multiprotocol gateway system delivered in wok package 2 will only deliver encrypted reading to the data managemnet platform. Legacy data will have to me encrypted at the Utility using AES 128 encryption from libararies provided by Microsoft as part of the .Net Framework. Each meter wil have it own unique encryption key and unique Initialization vector, this will add another layer of security; if one key is comprimised it will not effect the security of the rest of the data-set. However, using multiple encryption keys to encrpt the data-set will result in additional compexity as we will have to manage the keys at the Utility or in another secure location where physical access is restricted.

The key management data base will have to accomondate a very simplicitic data model where we will match each meter identification with the AES encryption key and the intialization vector. We are using Microsoft SQL Server database for key management, SQL Server is a relational database management system developed by Microsoft. SQL Server is ideal choice as facilitates a hybrid cloud deployemnt quite easily considering we are using Microsoft Azure as our clould platform provider.

## 5.11 Elasticity in Data Management Platform

During times of high load the data management platform can scale up to meet demand it can also scale down when demand has dropped off. The platform scales according to percentage CPU usage on the storage monitor worker roles and according to queue size on the on the data processing worker roles. All scale up and scale down effort will only happen in the last 20 minutes of each hour in order to avoid a “yoyo” effect. The “yoyo” effect of processing nodes powering up and powering down when a threshold is marginally achieved or under achieved in high undesirable as a billing unit in a minimum of one hour in Windows Azure. Scalability in the platform works on the principle once we have purchased an hour of processing time additional resources, it will not attempt to scale down until forty-five minutes of that hour has passed. The below table shows the range each role can scale up and down in the platform:

Table 24: Scalability in data management platform

|  |  |  |
| --- | --- | --- |
| Component | Min. Scale Range | Max. Scale Range |
| Worker Role A (monitoring) | 1 | 2 |
| Worker Role B (processing) | 1 | 10 |
| Web Role (OData) | 1 | 5 |

## 5.12 Platform notifcations

The data management platform (WP3) will inform the UrbanWater platform (WP6) when it has received a new batch of readings from the new Gateway system being developed in WP2. This will be done by a web Service call, a web service will be provided by the platform by which the data management platform will send up a new date of the last meter reading. This date will be passed by the Gateway system as part of the file structure, please see sample file in Appendix C.

# Data management Cloud Platform UML

## 6.1 Storage Monitor Worker Role UML

****

## 6.2 Data Processing Worker Role & Queue UML

****

****

## 6.3 OData Web Role UML

****

## 6.4 Data Management Platform Scalability UML







## 6.5 Data Management Platform Security UML





# Conclusion and Outlook

This design document gives a detailed description of the design proposed for the data management cloud platform. The main objective until the integration phase of WP3 is to deliver the design as they are outlined in this document.

# References

[1] "Microsoft Azure Documentation", Microsofts Online Azure documentation Archive. Retrieved 24 May 2013. <http://www.windowsazure.com/en-us/documentation/>

[2] "Amazon Web Services Documentation", Amazons Online AWS documentation Archive. Retrieved 24 May 2013. <http://aws.amazon.com/>

[3] "The Open Data Organisation", The open data protocol organisation. Retrieved 24 May 2013. <http://www.odata.org/>

[4] Joan Daemen, Vincent Rijmen, "The Design of Rijndael: AES – The Advanced Encryption Standard." Springer, 2002. [ISBN 3-540-42580-2](http://en.wikipedia.org/wiki/Special:BookSources/3540425802).

[5]"Blob Storage Web Services", Complete documentation of put block and put block list for upload of data to BLOB Storage. Retrieved 30 May 2013.

<http://msdn.microsoft.com/en-us/library/windowsazure/hh674475.aspx>

[6]"Using OAuth to obtain a SWT", Secuirty guide using ACS to third party claims based authentication. Retrieved 30 May 2013. <http://msdn.microsoft.com/enus/library/windowsazure/ee691974.aspx>

# Annex A

The following is a complete guide to PutBlock and Putblock Web Services, this appendix has been ripped form Microosft Azure documentation [5] and is included as a complete reference.

## A.1 PutBlock()

The **Put Block** operation creates a new block to be committed as part of a blob.

[Request](javascript:void(0))

The **Put Block** request may be constructed as follows. HTTPS is recommended. Replace myaccount with the name of your storage account:

|  |  |
| --- | --- |
| **PUT Method Request URI** | **HTTP Version** |
| <https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=block&blockid=id> | HTTP/1.1 |

#### Emulated Storage Service URI

When making a request against the emulated storage service, specify the emulator hostname and Blob service port as 127.0.0.1:10000, followed by the emulated storage account name:

|  |  |
| --- | --- |
| **PUT Method Request URI** | **HTTP Version** |
| <http://127.0.0.1:10000/devstoreaccount1/mycontainer/myblob?comp=block&blockid=id> | HTTP/1.1 |

#### URI Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| Blockid | Required. A valid Base64 string value that identifies the block. Prior to encoding, the string must be less than or equal to 64 bytes in size.  For a given blob, the length of the value specified for theblockid parameter must be the same size for each block.  Note that the Base64 string must be URL-encoded. |
| Timeout | Optional. The timeout parameter is expressed in seconds. |

#### Request Headers

The following table describes required and optional request headers.

|  |  |
| --- | --- |
| **Request Header** | **Description** |
| Authorization | Required. Specifies the authentication scheme, account name, and signature. |
| Date or x-ms-date | Required. Specifies the Coordinated Universal Time (UTC) for the request |
| x-ms-version | Required for all authenticated requests. Specifies the version of the operation to use for this request. |
| Content-Length | Required. The length of the block content in bytes. The block must be less than or equal to 4 MB in size.  When the length is not provided, the operation will fail with the status code 411 (Length Required). |
| Content-MD5 | Optional. An MD5 hash of the block content. This hash is used to verify the integrity of the block during transport. When this header is specified, the storage service compares the hash of the content that has arrived with this header value.  Note that this MD5 hash is not stored with the blob.  If the two hashes do not match, the operation will fail with error code 400 (Bad Request). |
| x-ms-lease-id:<ID> | Required if the blob has an active lease. To perform this operation on a blob with an active lease, specify the valid lease ID for this header. |

#### Request Body

The request body contains the content of the block.

#### Sample Request

Request Syntax:

PUT https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=block&blockid=AAAAAA%3D%3D HTTP/1.1

Request Headers:

x-ms-version: 2011-08-18

x-ms-date: Sun, 25 Sep 2011 14:37:35 GMT

Authorization: SharedKey myaccount:J4ma1VuFnlJ7yfk/Gu1GxzbfdJloYmBPWlfhZ/xn7GI=

Content-Length: 1048576

[Response](javascript:void(0))

The response includes an HTTP status code and a set of response headers.

#### Status Code

A successful operation returns status code 201 (Created).

#### Response Headers

The response for this operation includes the following headers. The response may also include additional standard HTTP headers. All standard headers conform to the HTTP/1.1 protocol specification.

|  |  |
| --- | --- |
| **Response header** | **Description** |
| Content-MD5 | This header is returned so that the client can check for message content integrity. The value of this header is computed by the Blob service; it is not necessarily the same value specified in the request headers. |
| x-ms-request-id | This header uniquely identifies the request that was made and can be used for troubleshooting the request. |
| x-ms-version | Indicates the version of the Blob service used to execute the request. This header is returned for requests made against version 2009-09-19 and later. |
| Date | A UTC date/time value generated by the service that indicates the time at which the response was initiated. |

#### Sample Response

Response Status:

HTTP/1.1 201 Created

Response Headers:

Transfer-Encoding: chunked

Content-MD5: BN3lsXf+t19nMGs+vYakPA==

Date: Sun, 25 Sep 2011 23:47:09 GMT

Server: Windows-Azure-Blob/1.0 Microsoft-HTTPAPI/2.0

[Authorization](javascript:void(0))

This operation can be called by the account owner and by anyone with a Shared Access Signature that has permission to write to this blob or its container.

## A.2 PutBlockList()

[Request](javascript:void(0))

The **Put Block List** request may be constructed as follows. HTTPS is recommended. Replace myaccount with the name of your storage account:

|  |  |
| --- | --- |
| **PUT Method Request URI** | **HTTP Version** |
| <https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=blocklist> | HTTP/1.1 |

#### Emulated Storage Service URI

When making a request against the emulated storage service, specify the emulator hostname and Blob service port as 127.0.0.1:10000, followed by the emulated storage account name:

|  |  |
| --- | --- |
| **PUT Method Request URI** | **HTTP Version** |
| <http://127.0.0.1:10000/devstoreaccount1/mycontainer/myblob?comp=blocklist> | HTTP/1.1 |

Note that the storage emulator only supports blob sizes up to 2 GB.

#### URI Parameters

The following additional parameters may be specified on the request URI.

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| Timeout | Optional. The timeout parameter is expressed in seconds. |

#### Request Headers

The following table describes required and optional request headers.

|  |  |
| --- | --- |
| **Request Header** | **Description** |
| Authorization | Required. Specifies the authentication scheme, account name, and signature. |
| Date or x-ms-date | Required. Specifies the Coordinated Universal Time (UTC) for the request. |
| x-ms-version | Required for all authenticated requests. Specifies the version of the operation to use for this request. |
| Content-Length | Required. The length of the request content in bytes. Note that this header refers to the content length of the list of blocks, not of the blob itself. |
| Content-MD5 | Optional. An MD5 hash of the request content. This hash is used to verify the integrity of the request content during transport. If the two hashes do not match, the operation will fail with error code 400 (Bad Request).  Note that this header is associated with the request content, and not with the content of the blob itself. |
| x-ms-blob-cache-control | Optional. Sets the blob’s cache control. If specified, this property is stored with the blob and returned with a read request.  If this property is not specified with the request, then it is cleared for the blob if the request is successful. |
| x-ms-blob-content-type | Optional. Sets the blob’s content type. If specified, this property is stored with the blob and returned with a read request.  If the content type is not specified, then it is set to the default type, which is **application/octet-stream**. |
| x-ms-blob-content-encoding | Optional. Sets the blob’s content encoding. If specified, this property is stored with the blob and returned with a read request.  If this property is not specified with the request, then it is cleared for the blob if the request is successful. |
| x-ms-blob-content-language | Optional. Set the blob’s content language. If specified, this property is stored with the blob and returned with a read request.  this property is not specified with the request, then it is cleared for the blob if the request is successful. |
| x-ms-blob-content-md5 | Optional. An MD5 hash of the blob content. Note that this hash is not validated, as the hashes for the individual blocks were validated when each was uploaded.  The Get Blob (REST API) operation returns the value of this header in the Content-MD5 response header.  If this property is not specified with the request, then it is cleared for the blob if the request is successful. |
| x-ms-meta-name:value | Optional. User-defined name-value pairs associated with the blob.  Note that beginning with version 2009-09-19, metadata names must adhere to the naming rules for C# identifiers. |
| x-ms-lease-id:<ID> | Required if the blob has an active lease. To perform this operation on a blob with an active lease, specify the valid lease ID for this header. |

This operation also supports the use of conditional headers to commit the blob only if a specified condition is met.

#### Request Body

In the request body, you can specify which block list the Blob service should check for the requested block. In this way you can update an existing blob by inserting, replacing, or deleting individual blocks, rather than re-uploading the entire blob. Once you've uploaded the block or blocks that have changed, you can commit a new version of the blob by committing the new blocks together with the existing blocks that you wish to keep.

To update a blob, you can specify that the service should look for a block ID in the committed block list, in the uncommitted block list, or in the uncommitted block list first and then in the committed block list. To indicate which approach to use, specify the block ID within the appropriate XML element within the request body, as follows:

* Specify the block ID within the **committed** element to indicate that the Blob service should search only the committed block list for the named block. If the block is not found in the committed block list, it will not be written as part of the blob, and the Blob service will return status code 400 (Bad Request).
* Specify the block ID within the **uncommitted** element to indicate that the Blob service should search only the uncommitted block list for the named block. If the block is not found in the uncommitted block list, it will not be written as part of the blob, and the Blob service will return status code 400 (Bad Request).
* Specify the block ID within the **Latest** element to indicate that the Blob service should first search the uncommitted block list. If the block is found in the uncommitted list, that version of the block is the latest and should be written to the blob. If the block is not found in the uncommitted list, then the service should search the committed block list for the named block and write that block to the blob if it is found.

The request body for this version of **Put Block List** uses following XML format:

<?xml version="1.0" encoding="utf-8"?>

<BlockList>

<Committed>first-base64-encoded-block-id</Committed>

<Uncommitted>second-base64-encoded-block-id</Uncommitted>

<Latest>third-base64-encoded-block-id</Latest>

...

</BlockList>

#### Sample Request

To demonstrate **Put Block List**, assume you have uploaded three blocks that you now wish to commit. The following example commits a new blob by indicating that the latest version of each block listed should be used. It's not necessary to know whether these blocks have already been committed.

Request Syntax:

PUT https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=blocklist HTTP/1.1

Request Headers:

x-ms-date: Wed, 31 Aug 2011 00:17:43 GMT

x-ms-version: 2011-08-18

Content-Type: text/plain; charset=UTF-8

Authorization: SharedKey myaccount:DJ5QZSVONZ64vAhnN/wxcU+Pt5HQSLAiLITlAU76Lx8=

Content-Length: 133

Request Body:

<?xml version="1.0" encoding="utf-8"?>

<BlockList>

<Latest>AAAAAA==</Latest>

<Latest>AQAAAA==</Latest>

<Latest>AZAAAA==</Latest>

</BlockList>

Next, assume that you wish to update the blob. The new blob will have the following changes:

* A new block with ID ANAAAA==. This block must first be uploaded with a call to Put Block (REST API) and will appear in the uncommitted block list until the call to **Put Block List**.
* An updated version of the block with ID AZAAAA==. This block must first be uploaded with a call to Put Block (REST API) and will appear in the uncommitted block list until the call to **Put Block List**.
* Removal of the block with the ID AAAAAA==. Given that this block is not included in the next call to **Put Block List**, the block will effectively be removed from the blob.

The following example shows the call to **Put Block List** that updates the blob:

Request Syntax:

PUT https://myaccount.blob.core.windows.net/mycontainer/myblob?comp=blocklist HTTP/1.1

Request Headers:

x-ms-date: Wed, 31 Aug 2009 00:17:43 GMT

x-ms-version: 2011-08-18

Content-Type: text/plain; charset=UTF-8

Authorization: SharedKey myaccount:DJ5QZSVONZ64vAhnN/wxcU+Pt5HQSLAiLITlAU76Lx8=

Content-Length: 133

Request Body:

<?xml version="1.0" encoding="utf-8"?>

<BlockList>

<Uncommitted>ANAAAA==</Uncommitted>

<Committed>AQAAAA==</Committed>

<Uncommitted>AZAAAA==</Uncommitted>

</BlockList>

[Response](javascript:void(0))

The response includes an HTTP status code and a set of response headers.

#### Status Code

A successful operation returns status code 201 (Created).

#### Response Headers

The response for this operation includes the following headers. The response may also include additional standard HTTP headers. All standard headers conform to the HTTP/1.1 protocol specification.

|  |  |
| --- | --- |
| **Response** | **Descriptions** |
| Etag | The entity tag contains a value that the client can use to perform conditional **GET/PUT** operations by using the If-Matchrequest header. If the request version is 2011-08-18 or newer, the ETag value will be in quotes. |
| Last-Modified | The date/time that the blob was last modified. The date format follows RFC 1123.  Any operation that modifies the blob, including an update of the blob's  metadata or properties, changes the last modified time of the blob. |
| Content-MD5 | This header is returned so that the client can check for message content integrity. This header refers to the content of the request, meaning, in this case, the list of blocks, and not the content of the blob itself. |
| x-ms-request-id | This header uniquely identifies the request that was made and can be used for troubleshooting the request. |
| x-ms-version | Indicates the version of the Blob service used to execute the request. This header is returned for requests made against version 2009-09-19 and later. |
| Date | A UTC date/time value generated by the service that indicates the time at which the response was initiated. |

#### Sample Response

Response Status:

HTTP/1.1 201 Created

Response Headers:

Transfer-Encoding: chunked

Content-MD5: oafL1+HS78x65+e39PGIIg==

Date: Sun, 25 Sep 2011 00:17:44 GMT

ETag: “0x8CB172A360EC34B”

Last-Modified: Sun, 25 Sep 2011 00:17:43 GMT

x-ms-version: 2011-08-18

Server: Windows-Azure-Blob/1.0 Microsoft-HTTPAPI/2.0

[Authorization](javascript:void(0))

This operation can be called by the account owner and by anyone with a Shared Access Signature that has permission to write to this blob or its container.

# Annex B

## B.1 Reference Implementation to obtain a SWT from ACS

The following code is a reference implemnetation in c# of how coonect with the platform by obtaining SWT.

using System.Collections.Generic;

using System.Collections.Specialized;

using System.IO;

using System.Net;

using System.Text;

using System.Threading.Tasks;

using System.Web;

//Decalrations please note all namespace and credentials are simulated

static string serviceNamespace = "UrbanWaterOData";

static string acsHostUrl = "accesscontrol.windows.net";

static string realm = "http://127.0.0.1:8080/WaterMeterData.svc";

static string uid = "urbanwater";

static string pwd = "dfghTdfg#ddfgO";

//Button click event to get a new Simple Web Token and call OData Web Service

private void GetTokenButton\_Click(object sender, RoutedEventArgs e)

{

string token = GetTokenFromACS(realm);

WebClient client = new WebClient();

string headerValue = string.Format("WRAP access\_token=\"{0}\"", token);

//Pass SWT as part of the header when calling the OData Web Service

client.Headers.Add("Authorization", headerValue);

Stream stream = client.OpenRead(@"http://127.0.0.1:8080/WaterMeterData.svc/MeterReadings");

StreamReader reader = new StreamReader(stream);

String response = reader.ReadToEnd();

outputtxtbox.Text = response;

}

private static string GetTokenFromACS(string scope)

{

string wrapPassword = pwd;

string wrapUsername = uid;

// request a token from ACS

WebClient client = new WebClient();

client.BaseAddress = string.Format("https://{0}.{1}", serviceNamespace,

acsHostUrl);

//Use NameValueCollection to pass multiple values to WS

NameValueCollection values = new NameValueCollection();

values.Add("wrap\_name", wrapUsername);

values.Add("wrap\_password", wrapPassword);

values.Add("wrap\_scope", scope);

byte[] responseBytes = client.UploadValues("WRAPv0.9/", "POST", values);

string response = Encoding.UTF8.GetString(responseBytes);

Console.WriteLine("\nreceived token from ACS: {0}\n", response);

return HttpUtility.UrlDecode(

response

.Split('&')

.Single(value => value.StartsWith("wrap\_access\_token=",

StringComparison.OrdinalIgnoreCase))

.Split('=')[1]);

}

## B.2 Reference Implementation to upload data using PutBlock()

The following code is a reference implemnetation in c# of how to upload data files to the data management platform.

private void UploadDataButton\_Click(object sender, RoutedEventArgs e)

{

CloudStorageAccount cs = CloudStorageAccount.Parse("DefaultEndpointsProtocol=https;AccountName=urbanwaterData;AccountKey=AVeryLongKeyPDZo2QEU21NLc7dbdfgdfgdgJHGJGJHfgL5R8RlRm17P/Zzqgee2oHC6k3O6rB0EqtlpF7rtdfgdGGHJKRRbEw==");

Uri blobUri = UploadBlob(strFullPath,

cs,

"SAGEM/REDContainer");

}

// Approx. 4MB chunk size which is max upload size

private const int MaxBlockSize = 4000000;

public Uri UploadBlob(string filePath, CloudStorageAccount account, string containerName)

{

byte[] fileContent = File.ReadAllBytes(filePath);

string blobName = System.IO.Path.GetFileName(filePath);

return UploadBlob(fileContent, account, containerName, blobName);

}

public Uri UploadBlob(byte[] fileContent, CloudStorageAccount account, string containerName, string blobName)

{

CloudBlobClient blobclient = account.CreateCloudBlobClient();

CloudBlobContainer container =

blobclient.GetContainerReference(containerName);

container.CreateIfNotExist();

CloudBlockBlob blob = container.GetBlockBlobReference(blobName);

HashSet<string> blocklist = new HashSet<string>();

foreach (FileBlock block in GetFileBlocks(fileContent))

{

blob.PutBlock(

block.Id,

new MemoryStream(block.Content, true),

null

);

blocklist.Add(block.Id);

}

blob.PutBlockList(blocklist);

return blob.Uri;

}

private IEnumerable<FileBlock> GetFileBlocks(byte[] fileContent)

{

HashSet<FileBlock> hashSet = new HashSet<FileBlock>();

if (fileContent.Length == 0)

return new HashSet<FileBlock>();

int blockId = 0;

int ix = 0;

int currentBlockSize = MaxBlockSize;

while (currentBlockSize == MaxBlockSize)

{

if ((ix + currentBlockSize) > fileContent.Length)

currentBlockSize = fileContent.Length - ix;

byte[] chunk = new byte[currentBlockSize];

Array.Copy(fileContent, ix, chunk, 0, currentBlockSize);

hashSet.Add(

new FileBlock()

{

Content = chunk,

Id = Convert.ToBase64String(System.BitConverter.GetBytes(blockId))

});

ix += currentBlockSize;

blockId++;

}

return hashSet;

}

# Annex C

## C.1 File structure of Meter Readings

The following shows the internal structure of files uploaded to the data management platform by WP2 Gateway system the file will have to be processed by the system before it can be added to Azure table storage.

<?xml version="1.0" encoding="UTF-8"?>

<root xmlns="http://www.sagemcom.com/MeterscapeMeterDataExport/0.1">

<file-version>0.1</file-version>

<generation-date>2012-12-01T01:00:00Z</generation-date>

<!-- everytime MDM requests information about meters settings/status -->

<!-- information are cached and made available to MDM in a file like this one -->

<!-- "repository" element holds a list of "meter" elements -->

<!-- "fromDate" attribute is optional, only meters having data changed since

this date are reported -->

<!-- "toDate" attribute indicates the date at which the data has been reported

-->

<repository fromDate="2011-12-01T01:00:00Z" toDate="2012-12-02T01:00:00Z">

<!-- "meter" element provides meter's cached/stored information -->

<!-- "id" = meter serial number -->

<!-- "vendor" = vendor name or identifier -->

<!-- "model" = meter commercial name -->

<!-- "version" = meter software version -->

<meter id="SAG3432443232432" vendor="SAGEMCOM" model="CX1000" version="2.01" >

<!-- "data" elements provide last retrieved values of a meter's attribute

-->

<!-- this can be used for reporting load profile or log book configuration

to the MDM -->

<!-- example : class 7's cosem objects capture\_objects,capture\_period -->

<!-- this can also be used to report status -->

<!-- example : breaker output\_state/control\_state/control\_mode -->

<!-- "obis","attribute","class" identify the requested cosem object

attribute -->

<!-- "date" is the date at which the data concentrator gets the data from

the meter -->

<!-- "task" attribute provides the task identifier thu which the data was

retrieved (for tracking) -->

<!-- data is reported in the xml element value as a string of hexadecimal

digits -->

<data obis="1;2;99;1;0;255" attribute="3" class="7" date="2012-12-02T01:00:00Z"task="201210221452330001">0104020412000809060000010000FF0F02120000020412000109060000600A07FF0F02120000020412000309060100011D00FF0F02120000020412000309060100201D00FF0F02120000</data>

<data obis="1;1;99;1;0;255" attribute="3" class="7" date="2012-12-02T01:00:00Z"task="201210221452330001">0104020412000809060000010000FF0F02120000020412000109060000600A07FF0F02120000020412000309060100011D00FF0F02120000020412000309060100201D00FF0F02120000</data>

</meter>

<!-- another meter -->

<meter id="SAG3432654287633" vendor="SAGEMCOM" model="CX1000" version="2.01" >

<data obis="1;1;99;1;0;255" attribute="3" class="7" date="2012-12-02T01:02:00Z"task="201210221452330001">0104020412000809060000010000FF0F02120000020412000109060000600A07FF0F02120000020412000309060100011D00FF0F02120000020412000309060100201D00FF0F02120000</data>

</meter>

</repository>

</root>

# Annex D

This appendix contains a detailed guide in obtaining a SWT from ACS, it has been taken verbatim as a complete reference implementation from [6]

## D.1 Obtaining a SWT from ACS

When your web applications and services handle authentication using ACS, the client must obtain a security token issued by ACS to log in to your application or service. In order to obtain this ACS-issued token (output token), the client must either authenticate directly with ACS or send ACS a security token issued by its identity provider (input token). ACS validates this input security token, processes the identity claims in this token through the ACS rules engine, calculates the output identity claims, and issues an output security token.

This topic describes the methods of requesting a token from ACS via the OAuth WRAP protocol. All token requests via the OAuth WRAP protocol are transmitted over SSL. ACS always issues a Simple Web Token (SWT) via the OAuth WRAP protocol, in response to a correctly formatted token request. All token requests via the OAuth WRAP protocol are sent to ACS in an HTTP POST. You can request an ACS token via the OAuth WRAP protocol from any platform that can make an HTTPS FORM POST: .NET Framework, Windows Communication Foundation (WCF), Silverlight, ASP.NET, Java, Python, Ruby, PHP, Flash, and other platforms.

The following table lists three supported methods of requesting an ACS-issued SWT token via the OAuth WRAP protocol.

**Three methods of requesting a token from ACS via the OAuth WRAP protocol:**

|  |  |
| --- | --- |
| **Token request method** | **Description** |
| Password token requests | This simplest method requires the client to send a user name and password from a service identity directly to ACS via the OAuth WRAP protocol for authentication. |
| SWT token requests | This method requires the client to send a SWT token which can be signed with a service identity symmetric key or an identity provider symmetric key to ACS via the OAuth WRAP protocol for authentication. |
| SAML token requests | Intended primarily for Active Directory Federation Service (AD FS) 2.0 integration, the Security Assertion Markup Language (SAML) method requires the client to send a signed SAML token to ACS via the OAuth WRAP protocol for authentication. This approach allows the client to use an enterprise identity to authenticate with ACS. |

## D.2 Token issuing endpoint

All ACS token requests via the OAuth WRAP protocol are directed at an ACS token-issuing endpoint. The URI of this endpoint depends on the Access Control namespace. The namespace appears as a DNS name prefix in a token request URI. The rest of the DNS name is fixed, as is the path. For example, if you want to request a token from the Access Control namespace called "mysnservice", you can direct a token request to the following URI: <https://mysnservice.accesscontrol.windows.net/WRAPv0.9>.

## D.3 Password token requests

With a password token request, a client can send a user name and password from a service identity directly to ACS via the OAuth WRAP protocol for authentication. This is the easiest way to request a token from ACS using the OAuth WRAP protocol. Other than establishing an SSL connection, this approach requires no cryptographic capability. In practice, it is similar to the username/password model that is prevalent in REST Web services. This type of token request is actually an HTTPS form POST. The parameters in a password token request are form-encoded.

The following is an example of a wire trace of a plaintext request to a namespace called “mysnservice”.

POST /WRAPv0.9/ HTTP/1.1

Host: mysnservice.accesscontrol.windows.net

Content-Type: application/x-www-form-urlencoded

wrap\_scope=http%3A%2F%2Fmysnservice.com%2Fservices%2F&

wrap\_name=mysncustomer1&

wrap\_password=5znwNTZDYC39dqhFOTDtnaikd1hiuRa4XaAj3Y9kJhQ%3D

The table below provides the names, descriptions, and value requirements of the parameters that are required to be present in a password token request:

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Description** | **Value requirements** |
| **wrap\_scope** | Matches the token request against a set of rules. Set the value of this parameter to the value of the relying party application realm. You can obtain this value (in the **Realm** field) through the ACS Management Portal by selecting the appropriate relying party application from the **Relying Party Applications** page. | * HTTP or HTTP(s) URI * No query parameters or anchors. * Path segments <= 32. * Maximum length: 256 characters. * Must be URL-encoded. |
| **wrap\_name** | Validates the key of the next parameter. Set the value of this parameter to the name of a service identity within your Access Control namespace. You can obtain this value (in the **Name** field) through the ACS Management Portal by selecting the appropriate service identity from the **Service Identities** page. | * Minimum length: 1 character. * Maximum length: 128 characters. * Must be URL-encoded. |
| **wrap\_password** | Authenticates the incoming request. Set the value of this parameter to the password of a service identity within your Access Control namespace. You can obtain this value (in the **Password** field on the **Edit Credential** page), through the ACS Management Portal, by first selecting the appropriate service identity on the **Service Identities** page, and then selecting the appropriate password in the **Credentials** table on the **Edit Service Identity** page. | * String, minimum of 1 and maximum of 64 characters in length. * Must be URL-encoded. |

The values of these parameters must be URL-encoded before you send the request to ACS. Your web application or service can provide the value of the wrap\_scope to the client, or the client can decide to set the value of the wrap\_scope parameter to the URI of the web application or service resource target.

Password token requests via the OAuth WRAP protocol can also contain additional parameters which ACS can use during the output claim calculation process. These additional parameter names and values must be URL-encoded and values must not be quoted. The password token request method is fairly straightforward using .NET Framework.

WebClient client = new WebClient();

client.BaseAddress = string.Format("https://mysnservice.accesscontrol.windows.net");

NameValueCollection values = new NameValueCollection();

values.Add("wrap\_name", "mysncustomer1");

values.Add("wrap\_password", "5znwNTZDYC39dqhFOTDtnaikd1hiuRa4XaAj3Y9kJhQ=");

values.Add("wrap\_scope", "http://mysnservice.com/services");

// WebClient takes care of the URL Encoding

byte[] responseBytes = client.UploadValues("WRAPv0.9", "POST", values);

// the raw response from ACS

string response = Encoding.UTF8.GetString(responseBytes);

## D.4 SWT token requests

You can also request a token from ACS via the OAuth WRAP protocol using a SWT token signed by a symmetric key. All SWT token requests are made through an HTTPS form POST. The parameter values in this token request method are form-encoded.

The following is an example of a wire trace of a SWT token request to the "mysnservice" namespace.

POST /WRAPv0.9/ HTTP/1.1

Host: mysnservice.accesscontrol.windows.net

Content-Type: application/x-www-form-urlencoded

wrap\_scope=http%3A%2F%2Fmysnservice.com%2Fservices%2F&

wrap\_assertion\_format=SWT&

wrap\_assertion=Issuer%3dmysncustomer1%26HMACSHA256%3db%252f%252bJFwbngGdufECFjQb8qhb9YH0e32Cf9ABMDZFiPPA%253d

A SWT token request must have the following parameters and values:

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Description** | **Value requirements** |
| **wrap\_scope** | Matches the token request against a set of rules. | * Set the value of this parameter to the value of the relying party application realm. You can obtain this value (in the**Realm** field) through the ACS Management Portal by selecting the appropriate relying party application from the **Relying Party Applications** page. * HTTP or HTTP(s) URI. * No query parameters or anchors. * Path segments <= 32. * Maximum length: 256 characters. * Must be URL encoded. |
| **wrap\_assertion** | This is the input token that is being sent to ACS. | * A signed SWT token with input claims that include issuer and HMACSHA256 parameters. * Maximum length: 2048 characters. * Must be URL encoded. |
| **wrap\_assertion\_format** | This is the format of the input token that is being sent to ACS. | SWT |

As shown in the following example, the code that is required to make a SWT token request resembles the code that is required to make a password token request.

WebClient client = new WebClient();

client.BaseAddress = string.Format("https://mysnservice.accesscontrol.windows.net");

NameValueCollection values = new NameValueCollection();

// add the wrap\_scope

values.Add("wrap\_scope", "http://mysnservice.com/services");

// add the format

values.Add("wrap\_assertion\_format", "SWT");

// add the SWT

values.Add("wrap\_assertion", "Issuer=mysncustomer1&HMACSHA256=b%2f%2bJFwbngGdufECFjQb8qhb9YH0e32Cf9ABMDZFiPPA%3d");

// WebClient takes care of the remaining URL Encoding

byte[] responseBytes = client.UploadValues("WRAPv0.9", "POST", values);

// the raw response from ACS

string response = Encoding.UTF8.GetString(responseBytes);

## D.5 Claims assertion via the OAuth WRAP protocol

To enable backwards compatibility with ACS 1.0 token request behavior, ACS supports the ability to assert claims as part of token requests. The recommended way to do this is to register the asserting application or service as an ACS identity provider. Then the application or service requests a token from ACS by presenting a SAML or SWT token that contains the claims that it wants to assert, and this token is signed using an Identity Provider Key stored in ACS. For example, you can send a SAML token request with asserted claims to ACS via the OAuth WRAP protocol from AD FS 2.0 or any custom Security Token Service (STS) that is built using Windows Identity Foundation (WIF) and registered in ACS as a WS-Federation identity provider. You can use the ACS Management Portal to register an identity provider using WS-Federation metadata, or you can use the ACS Management Service to individually set identity provider properties, addresses, and keys. No service identities are required in this method of asserting claims in a token request. This method is functional via all ACS-supported protocols.

## D.6 Unwrapping and sending the token to a web application or service.

If the token request is successfully authenticated, ACS returns two form-encoded parameters: **wrap\_token** and **wrap\_token\_expires\_in**. The values of these parameters are the actual SWT token that the client can use to gain access to your web application or service and the approximate remaining lifetime of this token (in seconds), respectively. Before sending the SWT token to the web application or service, the client must extract and URL-decode it from the ACS response. If the web application or service requires the token to be presented in the HTTP Authorization header, the token must be preceded by the scheme WRAPv0.9.

The following code example demonstrates how to unpack a token and format the Authorization header.

WebClient client = new WebClient();

client.BaseAddress = string.Format("https://mysnservice.accesscontrol.windows.net");

NameValueCollection values = new NameValueCollection();

values.Add("wrap\_name", "mysncustomer1");

values.Add("wrap\_password", "5znwNTZDYC39dqhFOTDtnaikd1hiuRa4XaAj3Y9kJhQ=");

values.Add("wrap\_scope", "http://mysnservice.com/services");

// WebClient takes care of the URL Encoding

byte[] responseBytes = client.UploadValues("WRAPv0.9", "POST", values);

// the raw response from ACS

string response = Encoding.UTF8.GetString(responseBytes);

string token = response

.Split('&')

.Single(value => value.StartsWith("wrap\_token=", StringComparison.OrdinalIgnoreCase))

.Split('=')[1];

string.Format("WRAP access\_token=\"{0}\"", HttpUtility.UrlDecode(token));

## D.7 ACS Error Codes and Descriptions

ACS returns errors when it cannot satisfy a token request. In keeping with the REST design, the error contains an HTTP response code. In many cases, ACS errors also contain a SubCodeand Detail that provide context about what failed. The error format is: Error:Code:<httpStatus>:SubCode:<code>:Detail:<message>.

The Content-Type of an error is always text/plain.

HTTP/1.1 401 Access Forbidden

Content-Type: text/plain; charset=us-ascii

Error:Code:401:SubCode:T0:Detail:ACS50009: SWT token is invalid. :TraceID:<trace id value>:TimeStamp:<timestamp value>

# Annex E

***Reviewers Comments:***

*This deliverable is expected to provide the design of the data management cloud platform. Nevertheless, it lacks substantial information: no architecture diagram is provided, no storage technology to be used in the Azure cloud also. The WSDL file (WADL or any similar structure) has to be provided for the functionality to be served. This is information that has to be defined. The document does not correspond to the title and content that we are waiting for, a real system design document*

*Also reviewers comments in at Brussels meeting about the lack of explanation as to why we chose Windows Azure as cloud platform provider.*

**Solution - Amendment**

* A document organisation section has been added which will be included in chapter one to outline the structure of the document in order to cover the number of technologies used.
* The design section of the document has been completely re-written to include specific product design and overview including Architectural Diagram, WADL(EDMX) and No-SQL database.
* A new chapter is introduced to talk about the data the data management platform will have to store and how it will achieve this goal.
* More detail has been given as to why we have chosen Microsoft Azure as our cloud platform provider.
* A new appendix D has been added to give more detail as how to integrate with the platform security model.
* The document template has been changed to adhere to the new template provided by Ateknea.