DigitalRoute

Software Description - Usage Engine Private Edition 3

Table of Contents

Introduction 4
- Usage Data-driven Applications 5
- Product Themes 6
- Process Modelling and Configuration 6

Deployment Options 7
Cloud Native Design Principles 8
- 12 Factor App 8
- Micro Services 10
- Working with Micro Services in Usage Engine Private Edition 10
  - Orchestrated Container Architecture with Kubernetes 10
  - Dynamic API Driven Architecture 11
  - Test Framework 11
  - Solutions as Workflow Packages 11
  - EC Deployments 11
  - OpenAPI and HTTP/2 12
  - CI/CD 12
  - Automatic Scaling 12

Architecture 13
- PODs 14
- Storage 15
- Operational APIs 15
- Networking 16
- Third Party Components 18
- Security 19

Operational Framework 19
- Logging 19
- Monitoring (System Insight) 20
- Notifications and Alerts 20
- System Statistics Application 22
- Tracing 23
- Audit 24
- Trouble Shooting 24

Continuous Integration/Continuous Delivery (CI/CD) 25
- CI/CD Example Pipeline 27
- CI/CD Test Framework 27
- CI/CD Workflow Packages 27

Usability 28
Workflow 29
- Workflow Types Description 29
- Workflow Configuration 30
  - Workflow Group Configuration Description 30
  - Runtime Modification of the Workflow Configuration 31
  - Profiles in Workflow Table 32
- Workflow Management 32
  - Table-driven Workflow Instancing 32
  - Workflow Execution Scheduling 33
- Workflow Execution 34
  - Batch and Task Workflow Execution 35
  - Real-time Workflow Execution 35
  - The Execution Manager 36
- Workflow Execution Suspension 36
- Configuration Import and Export 37
- External References (Parametrization) 38
- Error Handling 39

Agents Description 40
Sub Systems 41
- Handling of Data Formats (UFDL) 41
  - Proprietary Formats 42
  - ASN.1 Formats 42
  - XML Formats 43
- Processing Data (APL) 43
  - Function Blocks Description 43
- Workflow Bridge Description 45
- Aggregation, Consolidation and Correlation 46
- Detection of Duplicates 46
- Archiving Description 47
- Error Correction System Description 49
- Data Veracity Description 52
  - Data Veracity Profile Description 52
  - Data Veracity Agents 52
  - Data Veracity Operator UI 53

Policy and Charging Control 54
- Data Model 55
- Data Persistency 56
- PCC Workflow Configurations 56
- User Interface 56

Appendix A – Interfaces 57

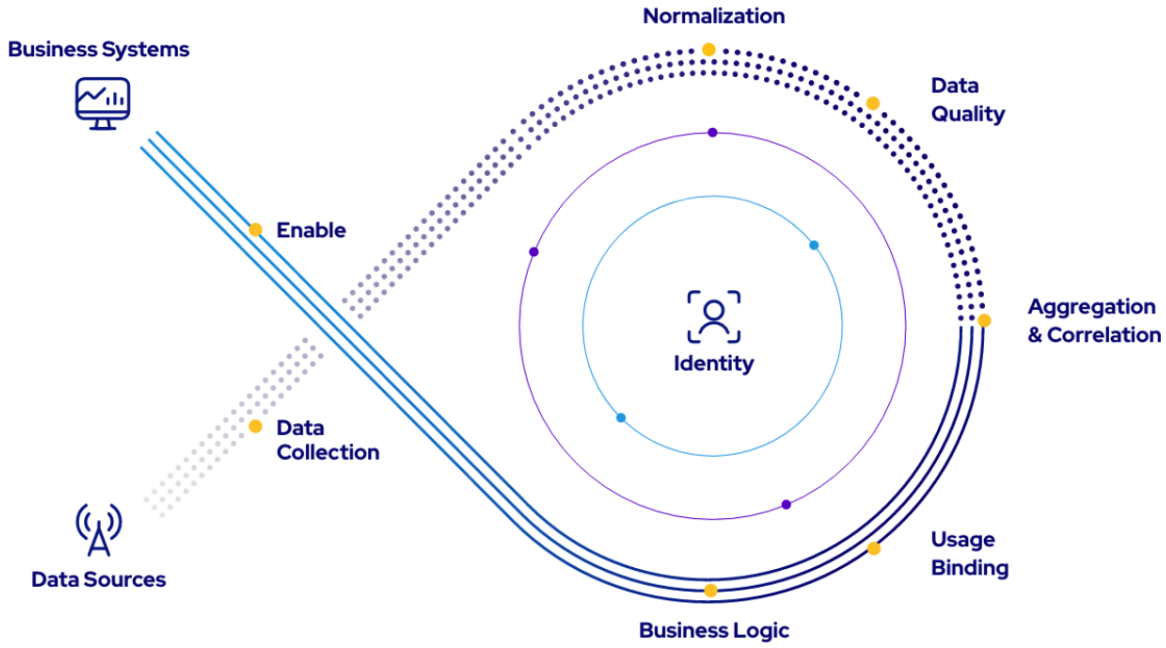
Software Description - Usage Engine Private Edition

Usage Engine - Private Edition is the latest generation of DigitalRoute's world-class usage data processing software. It represents a generational change in terms of architecture and modernized operations through the adoption of cloud-native technologies for deploying, running, and operating the software.

Search this documentation

Introduction

Usage Engine is an enterprise-grade data processing platform, combining strong integration capabilities with class-leading data processing functionality to derive useful information and drive real-time decision-making across many applications such as: billing, operations, revenue assurance, service assurance, entitlement enforcement, service control and business intelligence. A typical application of the software features and capabilities is outlined below:



Data Sources

The transaction chain starts with usage generated based on the consumption of a service or product. The service delivery platform(s) will generate, and log usage records based on actual consumption. Virtually any type of service delivery platform and associated method of generating usage information is supported, and as such the types of systems Usage Engine interfaces various significantly. Common examples include telecommunications network elements, databases, payment processors, SaaS platforms, APIs, IoT devices, and messaging queues.

Data Collection

The first step performed by the Usage Engine is to connect to the service delivery platforms to extract the relevant usage information. Either via a file-based interface, or over a real-time interface (bi- or uni-directional).

After the Usage Engine has collected the usage information, it needs to decode the data according to the way it was encoded by the usage delivery platform. There are many ways to encode and serialize data, and the Usage Engine supports most data encoding schemas through its data formatting subsystem. Examples include text-based formats such as ASCII, CSV, XML, and JSON. Many binary formats such as Google Protocol Buffer, Avro, ASN.1, AMA, and other complex fixed and variable length formats are also supported.

The decoder breaks down input data into transactions that are processed individually in subsequent steps, allowing for full control over every single usage transaction.

Normalization

After usage data has been collected and decoded, a common processing step is to normalize the transactions. This makes it easier to work with the data, as many systems will represent similar information in different ways

Data Quality

Ensuring the correct data quality is an essential step to leveraging usage data in a business process. Examples of data quality issues range from corrupted files, duplicate transactions, transactions missing key information, or badly formatted data that cannot be used as is. To avoid data and revenue loss Usage Engine has capabilities to correct and repair data detected quality issues, but transactions can also be discarded at this point.

Aggregation and Correlation

Aggregation and Correlation are two very powerful capabilities of the Usage Engine, where aggregation is the process of combining multiple transactions from a single source system, and correlation being the same but across multiple sources. There are many reasons why aggregation and correlation of usage information is often necessary, such as: Multiple usage records (from one or more sources) are needed to correctly describe the service consumption, more usage records are being generated than downstream business systems can handle and the data volumes must be reduced, separate data sets must be combined to derive useful information, etc.

Page Break

Usage Binding and Identity

Usage delivery platforms almost always lack the ability to put the service consumption into perspective of how the service or product is being delivered. Binding the usage information to a meaningful identity is a crucial step to derive useful actions and insights into the usage data. Often this requires enrichment of the usage data through referencing external systems and data sets, such as: subscriber information, product definitions, organizational hierarchies, or network topology information.

Business Logic

Any custom business rule that needs to run on the usage data can be automated as part of the transactional data processing flow.

Enable

Once clean, actionable, and usable information has been derived from the raw usage information it can be used to enable a range of downstream business processes such as: Billing, Analytics, Revenue Assurance, CRM, CPQ, ERP, etc.

The same set of integration and data formatting capabilities as used for the data ingestion can be leveraged.

Uni- and Bi-directional Flows

This entire process can be setup as a one-way data flow, or a bi-directional data flow where actions are derived and triggered by sending appropriate information and actions back to the originating service delivery platforms.

Governance

The process is also surrounded by a governance layer, ensuring auditability and traceability of the usage data processing that has taken place.

Usage Data-driven Applications

The capabilities provided by the Usage Engine, can be combined to create a multitude of online and offline data processing applications.

Typical use-cases and applications built on the Usage Engine include:

- Billing Mediation for Telecom networks
- Automated usage processing applications to support Subscription and Pay-as-you-go business models
- Online Charging and Metering applications
- Invoice and Payment reconciliation applications
- Partner settlement applications
- Policy and Entitlement enforcement applications
- Service Assurance and Performance Management applications for Telecom networks

- Leveraging usage data as part of Quote-to-cash automation

There are also additional sub-systems and functionality that can be leveraged to create high-value business outcomes, described throughout this document.

Product Themes

DigitalRoute has been building technology for leveraging and managing different types of usage data related to business and operational processes for over 20 years. The drivers behind the creation of the Usage Engine, and the need for a new product, come from the shifting IT and deployment landscape born in the cloud native era. A fundamental change in product architecture and deployment mechanisms was required to bring a product to market that natively fits in private clouds and hyperscaler environments based on Kubernetes infrastructure. The product themes and underlying technology drivers that have resulted in this product are the following:

Adapting Cloud-native Design Principles

- Clear separation between application and infrastructure
- Elasticity: utilize the right resources at any given time
- Automated Life Cycle Management: Zero touch for all operations of a product's life cycle (onboard, deploy, update, scale).

Operational Framework

- Using best of breed industry standards for non-core functions like logging and monitoring. We have used a "plug-and-play" approach to avoid being dependent on specific third parties.
- Replacing legacy functions with a modern generation of operational tools

CI/CD Support to Ensure Quality and Repeatability

- Introducing improved packaging (to make it fast and easy to move use cases between development, test, and production systems).
- New Test Framework to automate test suite execution

Usability: Clear Separation Between Developer and Operation Personas

- Workflow creation and configuration use a dedicated Developer UI
- Operational tasks are performed in a brand-new operations Web-UI

Process Modelling and Configuration

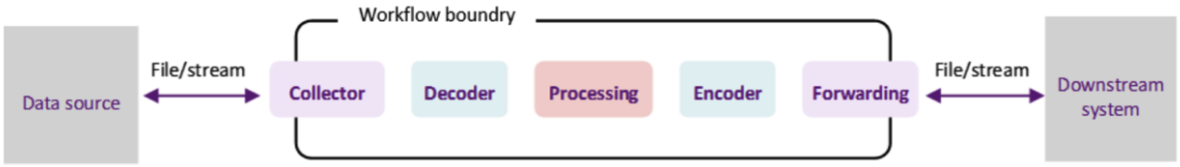
The process modeling mechanism Usage Engine leverages to create applications and data processing logic is through a flexible configuration layer. Its high-level graphical environment delivers a complete set of configurable tools to meet the challenging requirements of our customers.

Usage Engine Agents

Workflows are the cornerstones of process modeling and configuration in Usage Engine. A workflow is a set of nodes that are referred to as agents. Processing models are visualized by connecting the agents to each other using drag and drop.

Workflows contain three types of agents:

- ***Collection agents***are responsible for gathering data into the workflow from data sources or the client of a bi-directional real-time flow. An example of a simple collection agent could be one that reads a file from disk and sends its contents into the workflow.
- ***Processing agents***delivers data on one or many outgoing routes. A processing agent could be as simple as a counter that counts the throughput. It could also be more complex in that it evaluates the data and depending on the result, delivers it on different routes. Amongst the processing agents, so-called transformer agents can be identified. A transformer agent is responsible for translating an incoming byte stream into a UDR object or the opposite. For file distribution the Encoder Agent can be used to create header/trailer records containing meta-data of the file, e.g. record counter, check sum etc. This is commonly referred to as Decoding and Encoding, which the Usage Engine Ultra format system will handle.
- ***Forwarding agents***are responsible for distributing the data from the workflow to other systems or devices. An example of a forwarding agent could be one that creates a file from a data stream and transfers it to another system using FTP.



Workflows

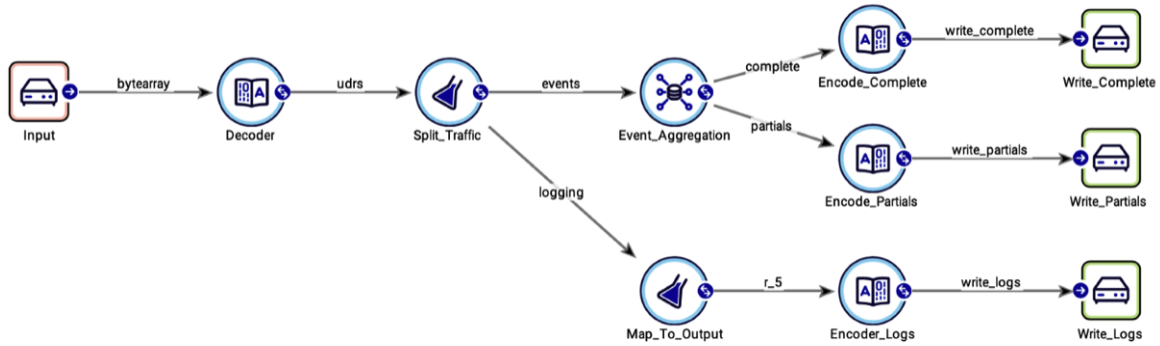
Workflows may be interlinked into data processing and orchestrattion tasks of virtually any complexity, where the output of one Workflow is the input of another.

There are three different types of workflows, that each have a slightly different execution behavior, but their modeling and configuration structure is the same.

- ***Batch workflows***– Are used to collect, process and distribute file-based data, also referred to as an Offline model. Batch Workflows can be configured for multi-threaded execution and enforces a first-in-first-out processing order and a strict transaction boundary based on each batch processed.
- ***Real-time workflows***– Enables online processing of requests/answers with other systems. These Workflows use multi-threading to enable execution of large numbers of independent execution paths simultaneously.
- ***Task workflows***– Used to execute common activities such as cleanup or maintenance tasks. Several System Tasks are pre-configured in Usage Engine and can be complemented with any user-defined activity.

The workflow configuration is used to design and configure workflows, by adding agents and connecting them to each other.

The following illustration shows an example of a simple Batch workflow:



Batch Workflow Example

Workflows that are conceptually related can be grouped together. This enables the setting of common execution or monitoring characteristics on all elements in a group, such as execution scheduling for Workflow Groups or selection of execution environment for Workflows and Workflow Groups.

Workflow and Workflow Group functions include, among other things:

Function	Description
Scheduling	Workflows Groups can be scheduled for execution periodically, on specific occasions or as a result of an event within the Usage Enginee system.
Distributed execution	A workflow or a group of workflows can be directed to execute on a particular execution node, or it can automatically be distributed to a node with the lowest load or lowest number of active Workflows.
Suspend execution	For a period of time a workflow or a group can be suspended from executing as scheduled.
Version handling	All versions of workflows and groups are saved during configuration, and it is possible to view and rollback to any previous version.
Table configuration	Workflows with the same structure can easily be multiplied in a table-based view, where the user selects which parameters that should be possible to define per workflow.
External References	Workflow configuration can be managed via property files called external references, providing the facility for external control of Workflow execution from outside the Desktop.
Meta Information Model (MIM)	Upon configuration, the workflow and its agents provide runtime attributes – MIMs, which can be fetched and distributed to other parts of the system. MIMs can be both static and dynamic. Examples of MIM values are, for example, agent names and names of collected files.
Encryption	As with all configurations, also Workflows and groups can be saved encrypted. In such a case, read or write access to encrypted configurations is only possible if a correct password is provided.

Deployment Options

Usage Engine Private Edition can be deployed on any Kubernetes engine compliant with the upstream version stated in the release notes. This includes OpenShift distributions as well as the hosted Kubernetes services available from all hyper-scalers – Amazon EKS, Google GKE and Azure AKS.

Test and Development Environment

A local environment can easily be setup in lightweight desktop Kubernetes environment like Docker Desktop, Rancher Desktop, Minikube or similar. Standard container runtimes like containerd and dockerd are supported. At least 2 CPUs and 4Gb of memory is recommended for a basic deployment with a few simple workflows.

Deployment in AWS

AWS is currently the most common and well tested environment for public cloud deployments. DigitalRoute provides customizable infrastructure as code (IaC) templates in Terraform format to make it possible to get an environment up and running quickly.

The following services are mandatory in an AWS deployment:

- Amazon RDS for PostgreSQL
- Amazon Elastic File System
- Amazon Route 53
- Amazon Elastic Kubernetes Service
- AWS Key Management Service
- Elastic Load Balancing

Note that Usage Engine require Kubernetes and thus is not possible to deploy in Amazon ECS or directly on EC2s without Kubernetes.

Deployment in other cloud environments

Usage Engine can be deployed in any cloud environment that provides an upstream compatibly Kubernetes cluster. It has been verified in hyperscaler environments like Google Cloud Platform, Microsoft Azure as well as in private cloud using Kubernetes distributions like OpenShift, Rancher and VMware Tanzu.

Usage Engine is delivered as an environment agnostic Helm chart, ready to be deployed in any Kubernetes cluster. To take full benefit from the application and environment, some services need to be configured to provide:

- Shared file system - Corresponding to EFS in AWS
- DNS - Corresponding to Route53 in AWS
- Layer 7 and layer 4 load balancer – Corresponding to ALB and CLB in AWS
- Postgres or Oracle compatible database – Corresponding to RDS in AWS
- Certificate management for TLS traffic – Corresponding to KMS in AWS

Cloud Native Design Principles

Usage Engine has been designed with cloud-native design principles in mind. These concepts are described here.

12 Factor App

The design of Usage Engine has been highly influenced by cloud native design principles. To get an understanding of what that means in practice, we will go through the 12 factors outlined by the 12 Factor App methodology, design principles often referred when building software for the cloud. More on the 12 Factor App can be found on https://12factor.net/.

Codebase

One codebase tracked in revision control, many deploys

This principle dictates that there should be a one-to-one mapping between code base and application. Usage Engine is a platform, which is the application, built from a single code base. On top of the Usage Engine platform there are solutions. These solutions are built from separate codebases and deployed as individual applications.

Dependencies

Explicitly declare and isolate dependencies

The Usage Engine platform and solutions deployed on it utilize an advanced packaging system to manage build- and runtime dependencies. Fine grained runtime isolation on class level is used using the well adopted OSGi modularity framework.

Config

Store config in the environment

Usage Engine has been carefully designed to separate environment configuration from the application itself. In practice this means that things like database connections, different system properties, resource allocation as well as deployment specific parameters related to solutions can be defined in values files separate from the Helm charts used to package the application and solutions.

Backing Services

Treat backing services as attached resources

Given the clear separation between configuration and implementation, Usage Engine also makes no difference to whether consumed services are run locally or provided by third party. An example is the system database, which can be anything from an in-process Derby-database through a Postgres hosted in the same Kubernetes cluster to a fully external database service offered by the cloud provider (like AWS RDS) or a central Oracle database operated by a DBA team.

Build, Release, Run

Strictly separate build and run stages

Usage Engine uses a separation of the build and execution stages, both for the platform application itself but also for solutions. For solutions, the workflow package concept provides this separation. A solution is designed in the development environment, tested, and then deployed without modifications into a runtime/production environment together with environment specific configuration. To change solution implementation, the process is restarted from the beginning, i.e., the development environment.

Processes

Execute the app as one or more stateless processes

Usage Engine platform and solutions are executed as processes, hosted in Kubernetes Pods, and orchestrated, scaled and lifecycle controlled by the Kubernetes scheduler. State that needs to survive a restart of such a Pod is persisted to database or other persistent storage.

Port Binding

Export services via port binding

The Usage Engine platform and its solutions are self-contained and does not rely on runtime injection of a webserver into the execution environment to create network-facing services. The solutions do however have a dependency to the availability of the Usage Engine platform service to be able to download resources, which is a design decision. This is not violating the Port binding factor though as the contract with the execution environment is fulfilled by the binding of IP ports. Note that batch-based solutions are different in nature and are not serving user requests or input as online solutions are. The port binding factor still applies for batch-based solutions in how the batch jobs are configured and scheduled through an API, managed by Kubernetes API server.

Concurrency

Scale out via the process model

Usage Engine makes use of the process concurrency model with autoscaling mechanisms to achieve scale out. Through the ECD concept, certain workloads and tasks are assigned to certain process types, individually configured, and scaled to suite the requirements of their respective task in the best way.

Disposability

Maximize robustness with fast start-up and graceful shutdown

Usage Engine processes are disposable, meaning they can be started or stopped at a moment’s notice. They use graceful termination by acting on SIGTERM signals to make sure executing jobs and workflows are shut down in a controlled manner without losing data.

Dev/prod Parity

Keep development, staging, and production as similar as possible

The Dev/prod parity factor states that development and production environments should have as small a gap as possible. How well this is fulfilled is up to the team operating the environment, which in the case of Usage Engine is typically the customer.

What Usage Engine provides though is the means to facilitate the process of keeping environments as similar as possible. By packaging both the platform application and solutions in Helm charts with separate parameterization, it is easy to keep environments up to date. Usage Engine provides all the tools, including pre-built templates, to setup fully automated CI/CD pipelines.

Logs

Treat logs as event streams

All Usage Engine log data is routed to standard out where it becomes available through the Kubernetes ‘logs’ function. It is encouraged to collect all log data using log collection service like Fluent and forward it to a centralized storage like Elasticsearch for further analysis. This process is described in the user documentation.

Usage Engine by default formats log data as structured JSON events to make it suitable for analysis in Elasticsearch or similar tool.

Admin Processes

Run admin/management tasks as one-off processes

Usage Engine provides the purpose-built command line tool ‘mzcli’ to enable running admin processes as one-off tasks.

Micro Services

We are in the era of cloud computing and many companies are already moving their processing workloads from bare metal deployments in to private or public cloud deployments. This transition is not only about taking existing deployments and move them into containerized one, but it's also a transition in solution architecture and in how things are operated. In particular, the paradigm of "micro services" fits particularly well into cloud deployments and operations.

To set the scene, what is a micro service? https://microservices.io/ summarizes the micro service architecture as follows:

...an architectural style that structures an application as a collection of services that are:

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

These properties show what kind of values this architecture principle can bring to an organization in terms of processes, ways of working and structure.

If we look more into the technical side of a micro service architecture, we can identify other kinds of benefits. By deploying your solution in a micro service manner on a capable cloud native application platform, you can achieve benefits such as:

- Fully automated lifecycle control
- Controlled zero downtime upgrades
- Adaptive and dynamic scaling (which gives you cost control and availability)
- Abstraction from underlying infrastructure (which gives you cloud environment agnostic systems)
- Clearly defined APIs and separation of concerns
- An alignment with a rich ecosystem of cloud native operations tools

Working with Micro Services in Usage Engine Private Edition

Usage Engine has been architected to support building micro service architectures and ways of working.

Orchestrated Container Architecture with Kubernetes

Usage Engine Private Edition has been designed with Kubernetes based container deployment as the primary deployment environment. A lot of effort is put into making the application take advantage of powerful Kubernetes features such as:

- Self-healing and monitoring
- Auto scaling
- Networking and hooking into cloud environment load balancers
- Management of packages, certificates, persistence, and other resources

- Automatic zero downtime upgrades
- Location transparency and DNS based service discovery
- Optional interaction into powerful tools like:
    - Istio for service mesh architecture with traffic control and visualization using Kiali
    - Prometheus for automatic metrics-based monitoring and visualization using Grafana
    - Fluentd, Elasticsearch and Kibana for log collection and visualization
    - cert-manager for automatic certificate management

By using the Operator pattern to seamlessly integrate application specific functionality with Kubernetes orchestration functionality and extending that to modern Web UIs, we enable a smooth journey from business logic design into production ready deployment.

Dynamic API Driven Architecture

By enabling an API driven way of operating the system we achieve a separation of the development phase from the operational phase. This split is especially important for several reasons:

- The people designing the workflow logic are usually not the same people as the ones operating the system. Thus, they should also have different interfaces to work with.
- It is crucial that the business logic can be tested thoroughly in a test environment before deploying it to pre-production or production environments.
- It must be easy to move a solution between different environments in a reliable way, without environment specific information affecting the move.

Usage Engine has been architected to scale and manage assets like workflows and profiles in an operational phase, without touching business logic or design.

With this separation between development and operations in place it is possible to add operational APIs to automate the control and operation of the solution in the target environment. Usage Engine provides powerful operational APIs to control the following:

- Workflow creation and management
- Scheduling of batch workflows
- Global parameter control ("external references")
- User management
- Worker process (EC) management and scaling
- Monitoring of metrics and health

Test Framework

With the Python based test framework introduced in Usage Engine it is possible for users to write automated test suites of business logic and workflows as part of the solution.

Automated testing is a crucial factor to successfully adapt the CI/CD and micro service way of working. By triggering test suites from build pipelines, it is possible to detect errors and bugs in the solution at an early stage and avoid the pain of detecting them in production. It is also possible to bundle the test suites within workflow packages to be able to execute them even in the target environment.

The test framework is described in more detail later in the document.

Solutions as Workflow Packages

To be able to work with solutions (i.e., workflows and related configuration objects) in a controlled and automated CI/CD manner, the concept of Workflow Packages is introduced. A Workflow Package is an atomic, version handled and self-contained package that can contain a full solution. Installing or upgrading such a solution in a production environment is an instant, atomic operation that can easily be reverted in case of errors.

A Workflow Package can exist in multiple versions in parallel. This is a particularly important capability to achieve smooth zero downtime upgrades, like Blue/Green deployments or canary upgrades. A traffic management tool like Istio can then be used to control how traffic is distributed between the different versions of the packaged solution.

Workflow packages are described in more detail later in the document.

EC Deployments

The EC Deployments (or ECD) concept is the Kubernetes native extension of the well-established Usage Engine Execution Context (EC) concept. An EC is a worker process, designed to execute business logic in a distributed and scalable way. Adding the 'D' to the 'EC' gives us:

- The ability to instantiate ECs fully automatic through API or UI
- The ability to scale ECs fully automatically and apply that to batch and real-time contexts
- Location transparency - no manual editing of IP addresses
- The ability to manage the EC and associated real-time workflows as a single scalable unit
- The full control of networking for real-time solutions including location transparency, service discovery, load balancing, public and private interface routing
- Control and quota control of resources like storage, memory, and CPU

The ECD is implemented by extending the Kubernetes API using custom resources and backing that with the Kubernetes operator pattern to integrate it with the orchestrator as well as with Usage Engine.

In short, the ECD is your micro service. By adding business logic in the form of workflows to an ECD and integrate it in a CI/CD build pipeline, you have created a micro service and can start taking benefit from all the powerful characteristics of the cloud native micro service paradigm.

OpenAPI and HTTP/2

Usage Engine workflows are of course not limited to HTTP traffic and RESTful interfaces. You can build micro service type of architectures using any IP based protocol, or even file IO. However, as cloud native computing (including telco 5G architecture) is becoming increasingly standardized, many interfaces are converted to use HTTP(2) as the communication protocol and OpenAPI as the API definition language.

Usage Engine natively supports OpenAPI and HTTP/2 and integrates natively with JSON in the processing language APL. This makes it straight forward to build, connect and manage the lifecycle of workflows that use these standards. Using HTTP(2) as communication protocol also gives seamless integration with many tools within the CNCF (Cloud Native Computing Foundation) ecosystem like Istio service mesh, Jaeger for distributed tracing etc.

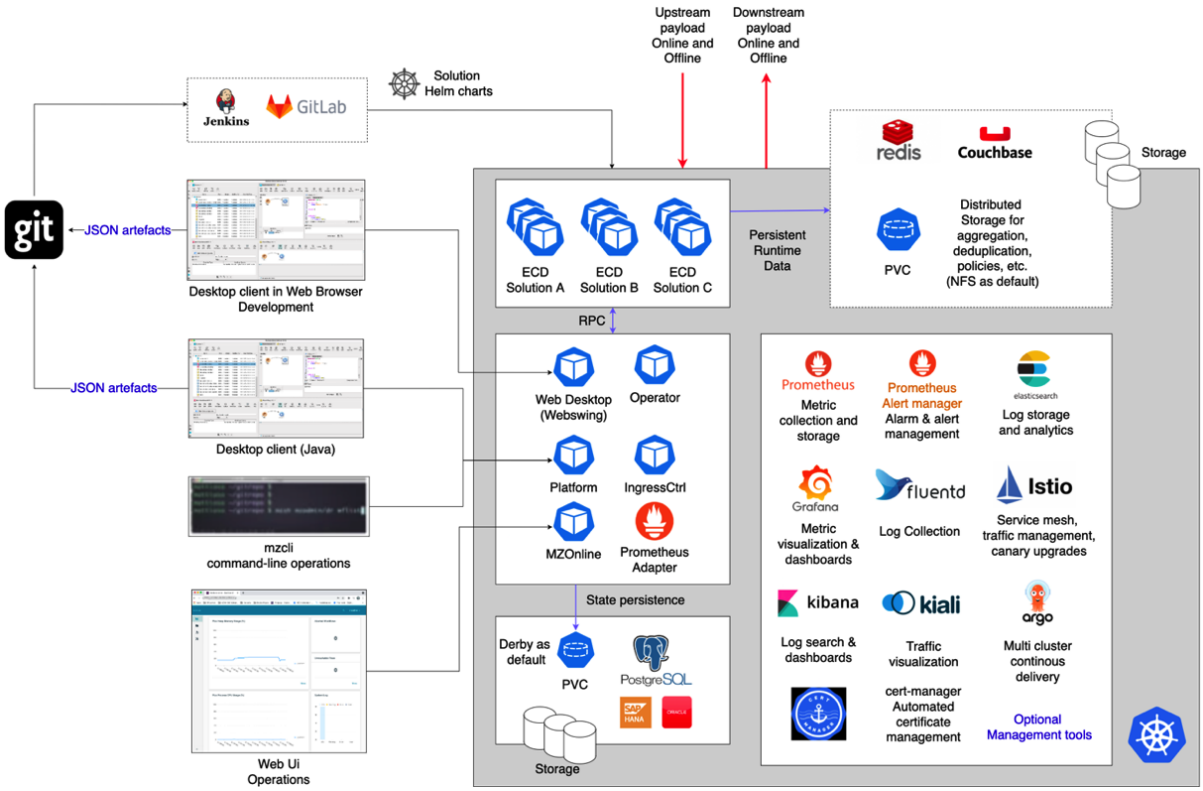
By combining OpenAPI specifications with workflow packages to allow you to run multiple versions of a workflow in parallel and with ECDs and Ingresses to enable exposing different interface versions on difference URL paths, you have the necessary tools to support API lifecycle control.

CI/CD

By combining the features described above and in specific the ECD and Workflow Package concepts, solutions built with Usage Engine on Kubernetes are well prepared to be integrated in a fully automated CI/CD pipeline. It is straight forward to take a solution packaged as a workflow package and build an image out of it. This image can then be deployed and managed as an ECD deployed in your Usage Engine Kubernetes cluster, i.e., turning it into a micro service.

This process fits well into an automated CI/CD process using tools like GitLab or Jenkins. By using a service mesh like Istio, the CD part of the pipeline can also be turned into a fully automated canary deployment process that gives you safe upgrades with no data loss.

Usage Engine provides example templates on how to build a micro service and use it with Jenkins and Istio to achieve a canary upgrade CI /CD pipeline. The CI/CD framework is described in more detail later in the document.



Usage Engine Architecture

Automatic Scaling

A cloud enabled architecture must be well adapted to a virtualized hardware environment, where it is expected to be cost-effective and utilize only as much hardware as is needed at any point during the day. Effective management of micro-services is key.

Scaling in PE is achieved by multiplying ECD definitions, containing one or several workflows. The workflows can be based on the same or any number of different configurations. The configuration, behavior and management differs when scaling ECDs for realtime versus batch workflows.

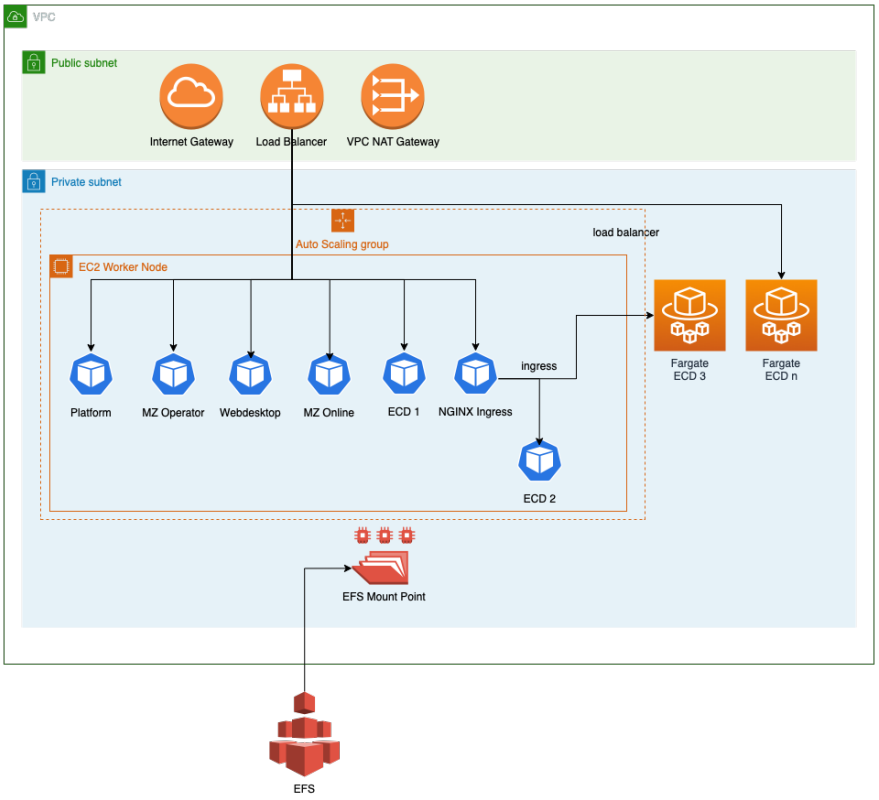
Realtime

- Realtime workflows typically execute continuously.
- An ECD configured to run dynamic realtime workflows will always be up once started. A user can configure minimum and maximum number of ECDs to run simultaneously. Even if a workflow aborts, Kubernetes will restart it until it is manually attended to.
- Scaling out and in of the ECD is automatic, and based on thresholds of configurable metrics. For instance CPU load.

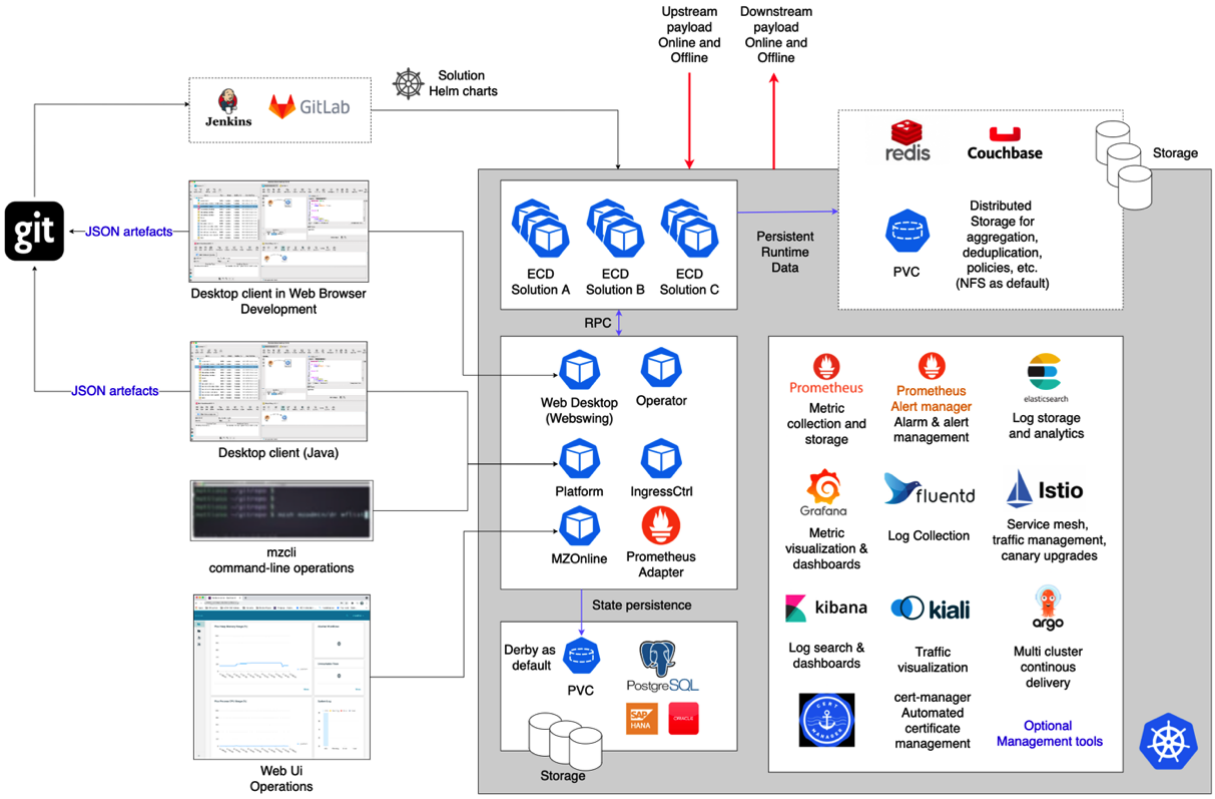
Batch

- Batch workflows typically execute periodically.
- An ECD configured to run batch workflows needs to be started before the workflows are scheduled to start.
- Scaling of batch ECDs is based on triggers. For instance; starting periodically, and stopping when workflows have finished processing all files.

Image below shows an example of a deployment in AWS, where Fargate hardware virtualization service is used when executing batch workflows that are scheduled a couple times a day. Realtime workflows that need to be executing continuously, are deployed on an EC2 instance. Any combination is possible - only EC2s, only Fargate, or a combination, depending on the price and/or architecture of preference.



Architecture



Usage Engine Architecture

- PODs
- Storage
- Operational APIs
- Networking
- Third Party Components
- Security

PODs

The core processes, hosted in Kubernetes as Pods, of the Usage Engine architecture are described below.

Platform

The platform process is the system orchestrator. It orchestrates application specific resources like:

- Scheduling of batch workflows, including complex grouping, dependencies, triggers etc.
- APIs – providing programmatic as well as Web UI access to system resources
- Alarms and Events
- Database connections

Operator

The operator process is the orchestrator and controller of Kubernetes resources. It makes sure that the application state is the same as what is defined in the ECD deployment descriptors. It does this by continuously monitoring the Status section of deployed ECDs. If the status differs from the desired state defined in the Specification section, the Operator will call APIs in the Platform service or in the Kubernetes API Server to update the state accordingly.

Execution Context

The Execution Context (EC) processes are the highly scalable worker processes that are responsible for executing the solution business logic that is deployed in the system. They are the heavy lifters that consumes payload, enriches, aggregates, correlates, loads, forwards, verifies, takes action and provides feedback based on the usage data flowing through the system. The number of ECs in a system can vary between 0 and several hundred depending on the load of the system. They can also be configured in bulk or individually, depending on the granularity used to control the solution.

How ECs are deployed, scaled, and configured is controlled in the ECD custom Kubernetes resource, which is the tool used by operators to control and orchestrate the runtime properties and behavior of solutions.

DROnline

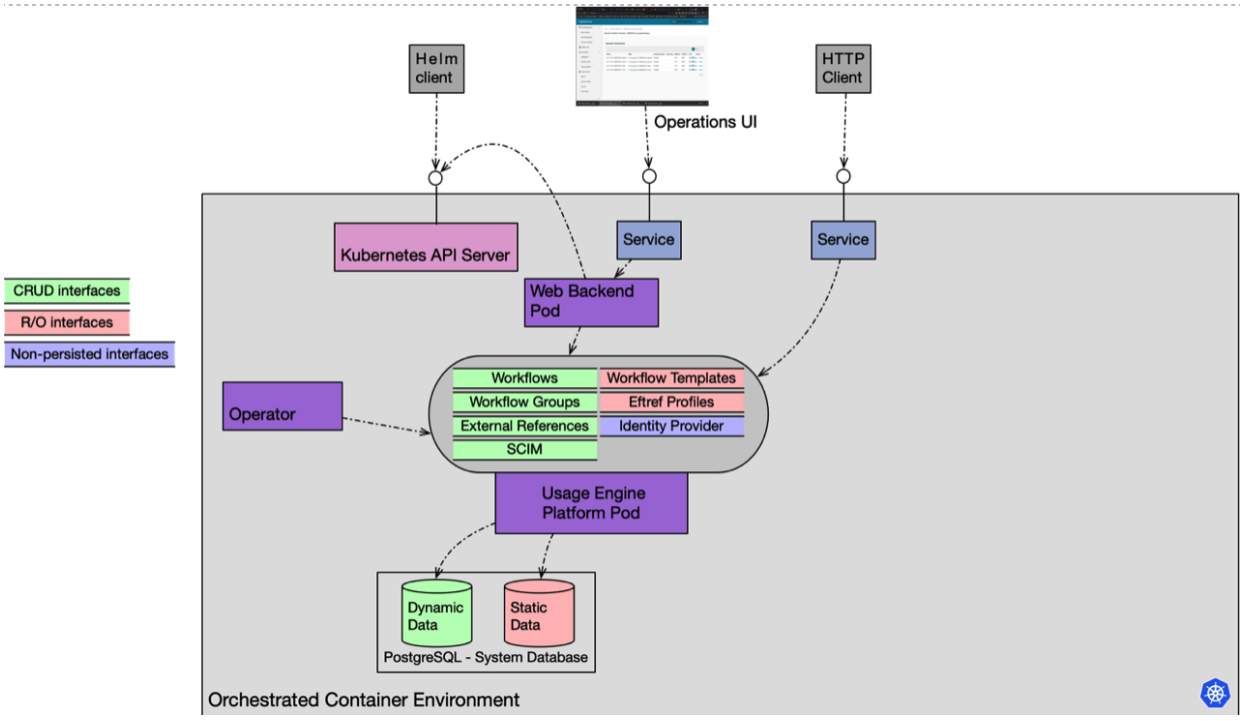
The DROnline pod is the backend of Usage Engine's web UIs. It connects the UIs to REST and other APIs in the system.

Storage

For certain use cases, such as performance critical aggregation with transaction safety, Usage Engine can utilize disk storage. Because of the distributed architecture, the storage should ideally be distributed. NFS compliant file systems, including high performance hosted NFS implementations provided by the hyper-scalers (such as AWS EFS) works well with Usage Engine. For high performance on-prem deployments, SAN based storages can also be an option.

Operational APIs

The Usage Engine platform contains several powerful Operational APIs to enable dynamic configuration and provisioning of the system in runtime.



Workflows API

API to read and configure workflows and workflow parameters. See OpenAPI specification in chapter "Release Information" in user documentation for detailed information.

Workflow Templates API

API to read information about workflow templates. See OpenAPI specification in chapter "Release Information" in user documentation for detailed information.

Workflow Groups API

API to read and configure workflow groups and scheduling rules. See OpenAPI specification in chapter “Release Information” in user documentation for detailed information.

Workflow State API

API to read and configure states of workflows. See OpenAPI specification in chapter “Release Information” in user documentation for detailed information.

External References API

API to configure parameters and parameter values on database backed External Reference Profiles. See OpenAPI specification in chapter “Release Information” in user documentation for detailed information.

External Reference Profiles API

API to configure database backed External Reference Profiles. See OpenAPI specification in chapter “Release Information” in user documentation for detailed information.

SCIM (System for Cross-domain Identity Management) API

API to configure users and access groups. See chapter “SCIM” in user documentation for details.

Identity Provider API (OIDC)

API enabling Usage Engine to act as the Identity Provider in OIDC based authentication flows. See chapter “OIDC Identity Provider” in user documentation for details.

Networking

Network Plugin

A Kubernetes cluster requires a networking plugin to function. Usage Engine works well with several different plugins, including the ones provided by the hyper-scaler platforms. For on-prem deployments, Calico is a powerful, robust, fast, and heavily tested network plugin.

Kubernetes Networking

To operate, Usage Engine needs to expose its services on IP ports reachable from outside of the Kubernetes cluster where it is hosted. Kubernetes offers multiple ways achieve this and Usage Engine offers flexibility in how to utilize Kubernetes networking features.

These are the different Kubernetes networking resources available for an application to use.

Service

A Service is what bridges a group of executing pods with external or internal IP ports. A service consists of three pieces of information: a name, a way to identify the pods in its group (typically a label), and a way to access those pods (port and protocol).

Services come in a few distinct types:

- **ClusterIP** – Accessible only from within the cluster.
- **NodePort** – Accessible on an externally exposed IP port on each node in the cluster. Easy to use but exposes only one service per port, are limited to the 30000-32767 port range and are lacking security policies.
- **LoadBalancer** – Connects a port in the cluster with an external Load Balancer, provided by the cloud environment. LoadBalancers support multiple protocols and multiple ports per service. They do consume one IP address for every service, which can add cost and overhead.

Usage Engine supports dynamic creation of all these Services.

The task of translating the service definitions into routing rules, effectively mapping the incoming traffic to the right pod, is handled by native components in Kubernetes, called kube-proxy and endpoint-controller. These work silently in the background. Usually all you need to know is that they exist and reliably solve the task of making sure the Service objects you have defined result in traffic being routed correctly, regardless of how Pods move between nodes or scale.

Ingress

An ingress is a routing rule applied on incoming traffic on a single externally exposed port. With Ingresses, payload data can be load balanced internally in the Kubernetes cluster without taking the overhead of assigning an IP address or port per endpoint. Ingress controllers can work on the transport- (OSI layer 4 - TCP/UDP) or application- (OSI layer 7 - HTTP/HTTPS) layer, though Usage Engine currently limits the support to the application layer.

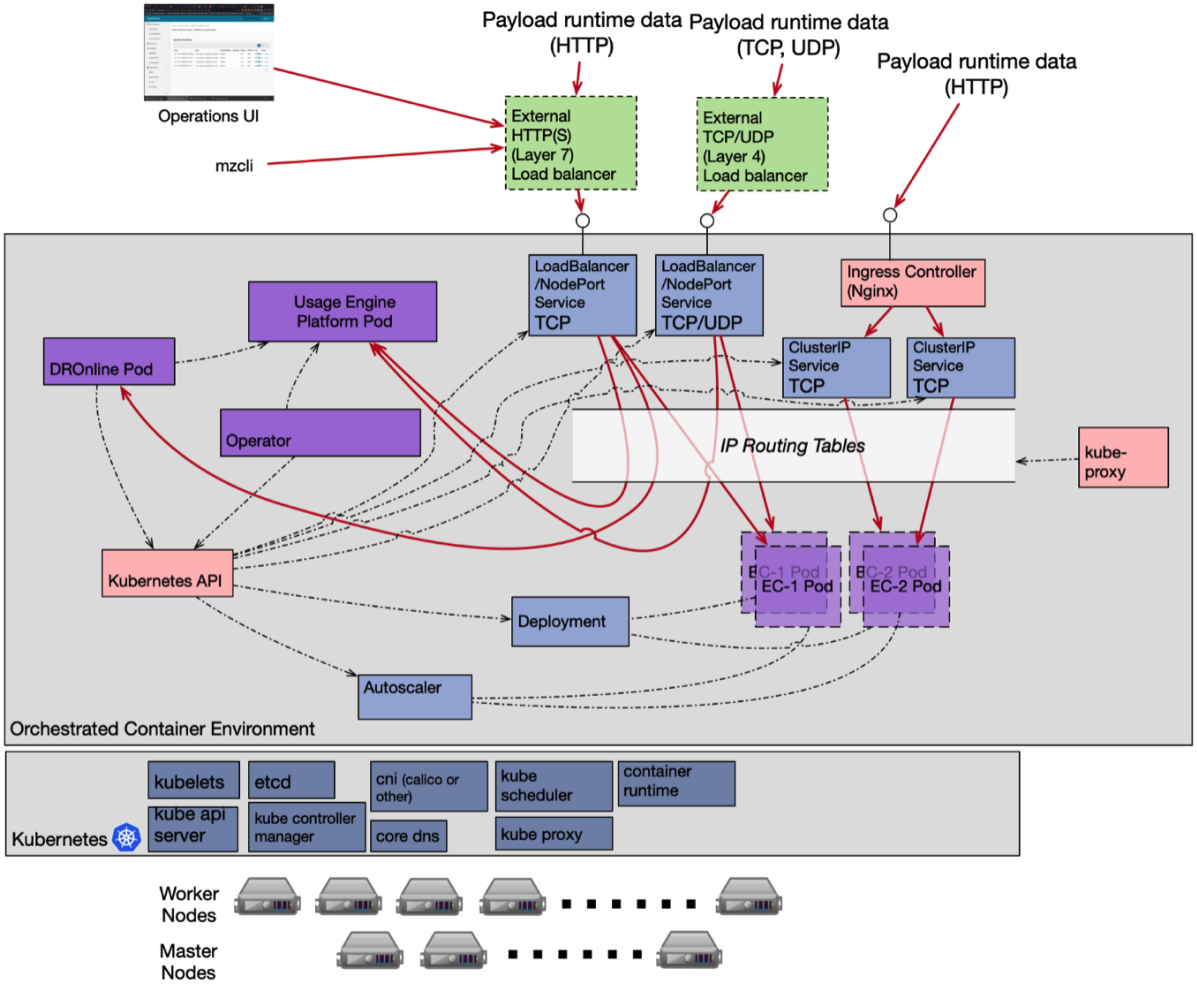
Ingress Controller

Ingresses require an Ingress Controller to work. The Ingress Controller takes care of applying the routing rule defined in the Ingress resources as well as routing the incoming traffic to the correct backing Service. The backing service does not have to be exposed outside of the cluster since the Ingress Controller exposes its own port, usually through a NodePort.

Usage Engine comes with a preconfigured Nginx Ingress Controller. The use of that is optional and it works perfectly fine to replace Nginx with another Ingress Controller, as needed.

Usage Engine and Network Resources

An overview of how networking resources can be used by Usage Engine is illustrated below. Note that this is only an example, and that this can depending on the implemented solution and on cloud environment specifics.



Usage Engine and Network Resources

We distinguish between network of the Usage Engine platform itself and the solutions implemented on it using ECDs. Note that there is no built-in network level separation on traffic between these kinds of network traffic, although it is possible to achieve this through configuration in the environment

Platform Networking

The Usage Engine platform and web backend processes need to expose ports to enable access to REST APIs. To connect an external Desktop client also TCP level access, need to be exposed.

The ports are:

Pod Port	Cluster Port	Used for	Protocol
9000	platform:80 or platform:443	Operational APIs Desktop client authentication	HTTP(S)
6790	platform:6790	Desktop control interface	TCP
9999	wd:9999	Web Desktop - Web UI	HTTP(S)
8080	mzonline:80	DROnline - Web UI	HTTP(S)

The ports in the table are by default exposed as NodePorts, but this can be changed during installation.

Solution Networking

Solutions deployed in Usage Engine are defined using ECD descriptor files. As part of an ECD descriptor any type of service objects can be created to expose IP ports bound by a solution to outside of the cluster.

It is important to be aware of that the creation of certain types of services can lead to cloud infrastructure costs. When configuring services as part of ECDs, be careful to consider the implications of your choice.

ClusterIP

ClusterIP is the natural choice if the port should be exposed for intra-cluster communication only, like to connect workflows from two Pods with each other.

NodePort

NodePort is the natural choice to expose ports for TCP or UDP based peer-2-peer communication. For some protocols that require source IP validation (for instance Radius) it is necessary to also bind the Pod hosting the server workflow to a specific physical node and to configure the NodePort with “External Traffic Policy” set to Local, to avoid translation of source IP addresses.

LoadBalancer

LoadBalancer can be a viable choice if you want to leverage LoadBalancers from the cloud environment and is not concerned about the cost or overhead coming from the fact that each configured load balancer allocates an IP address and a load balancer resource in the cloud environment.

Ingress

Ingress is the most cost effective and powerful choice to enable advanced routing and load balancing of incoming HTTP(S) based traffic.

Third Party Components

Usage Engine optionally integrates with several different third-party applications and systems to add additional power to the platform. The most important are:

- **PostgreSQL** – A powerful and commonly used system database for Usage Engine. It can be consumed as a service in a hyper scaler cloud environment, as an external on-prem installation or deployed alongside Usage Engine in the same or different Kubernetes cluster.
- **Oracle** – Supported as an alternative to PostgreSQL for use as system database
- **Nginx (Or another Ingress Controller)** - Used to manage ingresses, i.e., entry-points to HTTP-based solutions deployed in Usage Engine.
- **Prometheus** – Powerful metrics subsystem which Usage Engine can push metrics directly to or provide over a REST API for Prometheus to scrape. Alongside Prometheus the Prometheus Adapter can be used to drive custom metric scaling, and Prometheus Alert Manager to manage alarms and alerts.
- **Grafana** – Can be used to provide dashboards driven by the metrics in Prometheus. Usage Engine provides a default set of dashboards.
- **Fluentd** – Can be used to collect all log files from the system and forward the to a single destination.
- **Elasticsearch** – Can be used as a single destination for all log data to provide powerful search and filtering capabilities as well as more advanced features like machine learning.
- **Kibana** – Can be used to provide a graphical UI to Elasticsearch.
- **Redis** – Can be used as a distributed storage of aggregation or other payload data.
- **Couchbase** – An alternative to Redis, providing similar functionality.
- **Cert-manager** – Can be used to automate the process of generating and renewing certificates used for encrypted communication.

- **Jenkins / Gitlab / GitHub Actions (Or another CI tool)** – Can be used to setup automated CI/CD pipelines to drive solution integration and deployment.
- **Jaeger –** Usage Engine provides distributed tracing according to the OpenTracing standard where Jaeger can be used as a visualization tool for the traces.
- **Istio** – Can be used to orchestrate service mesh architectures with Usage Engine. A particularly useful feature is the possibility to perform fine grained canary upgrades of online workflows.
- **Kiali** – Can be used with Usage Engine and Istio to visualize traffic flows.
- **ArgoCD** – Can be used to drive the CD part of an automated solution pipeline.

Security

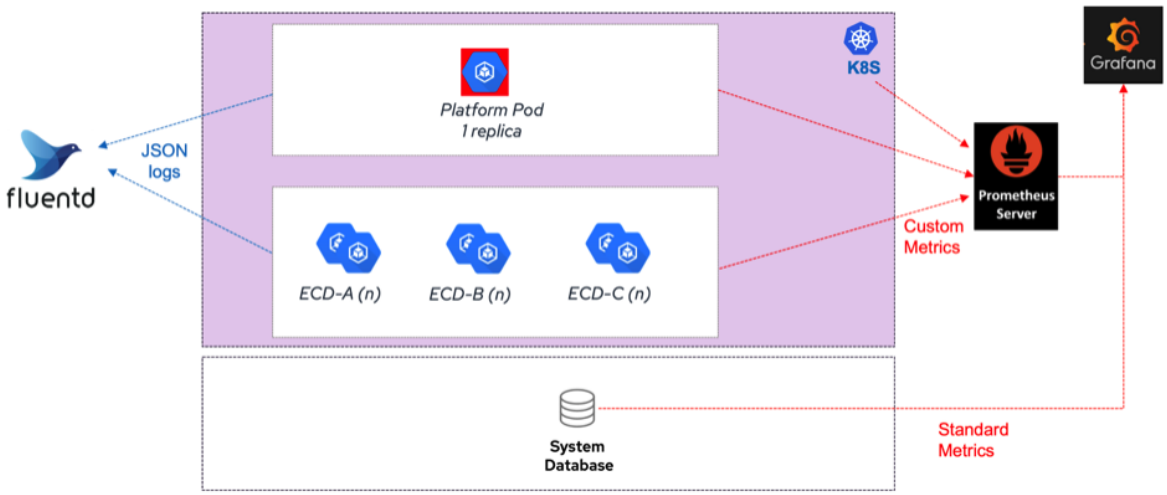
Usage Engine is built with a strong focus on security and has a password policy that conforms to NIST 800-63B. It provides the following security features:

- Token based authentication.
- Role base access control.
- OIDC Identity Provider.
- Encryption at rest – using tools to encrypt the data before writing it to storage.
- Encryption at transit – Using TLS encryption (note – a few agents/protocols still do not provide TLS. If encryption at transit is needed for such features, it must be enabled in the networking layer).
- Immutable images, scanned with image scanners for CVEs.
- Automated certificate management, using the cert-manager tool to integrate with CA.

Operational Framework

Usage Engine provides several alternatives and ways of working when it comes to the operational framework used to monitor and administer the system. For standard functions like logging, monitoring, and tracing, we have opted for a plug-and-play approach to best of breed third parties.

Note that these third-party products are optional. There is basic functionality included in Usage Engine that can be used instead.



Operational Framework

Templates for integration with recommended third-party products are included in the software.

- Logging
- Monitoring (System Insight)
- Notifications and Alerts
- System Statistics Application
- Tracing
- Audit
- Trouble Shooting

Logging

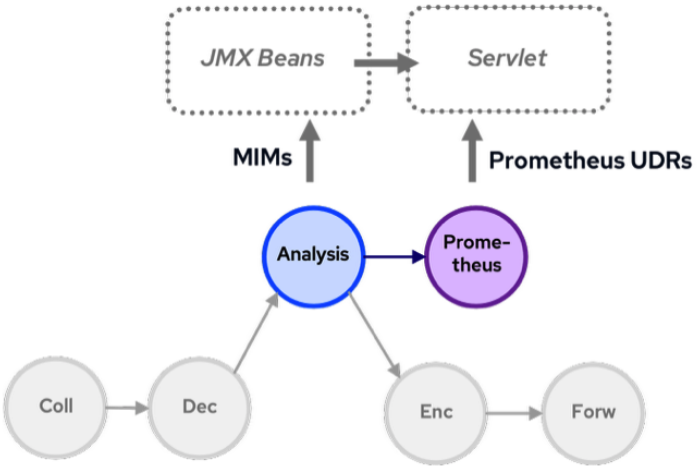
All logs – From Kubernetes (providing logs from services and pods), and from Usage Engine (providing logs from Execution Context, Platform, and workflows) – are easy to send to a Unified Logging Layer like Fluentd. The file format is JSON (log4j & RFC5424 standards). If no third party is used, the files can be read from a shared file system or sent to Kubernetes storage.

The visualization and management of messages can be done with a product like Elasticsearch.

Monitoring (System Insight)

Metrics from all layers – From Kubernetes (providing metrics from services and pods), and from Usage Engine (providing metrics from EC, Platform, and workflows) – are easy to send to a Unified Monitoring Layer like Prometheus. Metrics from Usage Engine are of two types:

- MIMs (simple metrics): these are gauge or sum metrics with one fixed label only. Only metric name and value can be customized. There is a set of automatically generated MIMs, but a user may also add customized MIMs. MIMs have fixed publishing intervals.
- Prometheus UDRs (complex metrics): for use cases where several labels – that can vary – are needed for a metric, Prometheus UDRs are used. Any parameters in the UDR, including time stamp, can be customized.



Metric Monitoring Approach

MIMs and Prometheus UDRs are sent to a Servlet, where they are scaped regularly by Prometheus.

For visualization, Grafana can be used. Usage Engine provides a set of predefined Dashboards for System Statistics. Also, in the Operations UI, a base set of System Statistics for the past 10 minutes can be viewed.

Notifications and Alerts

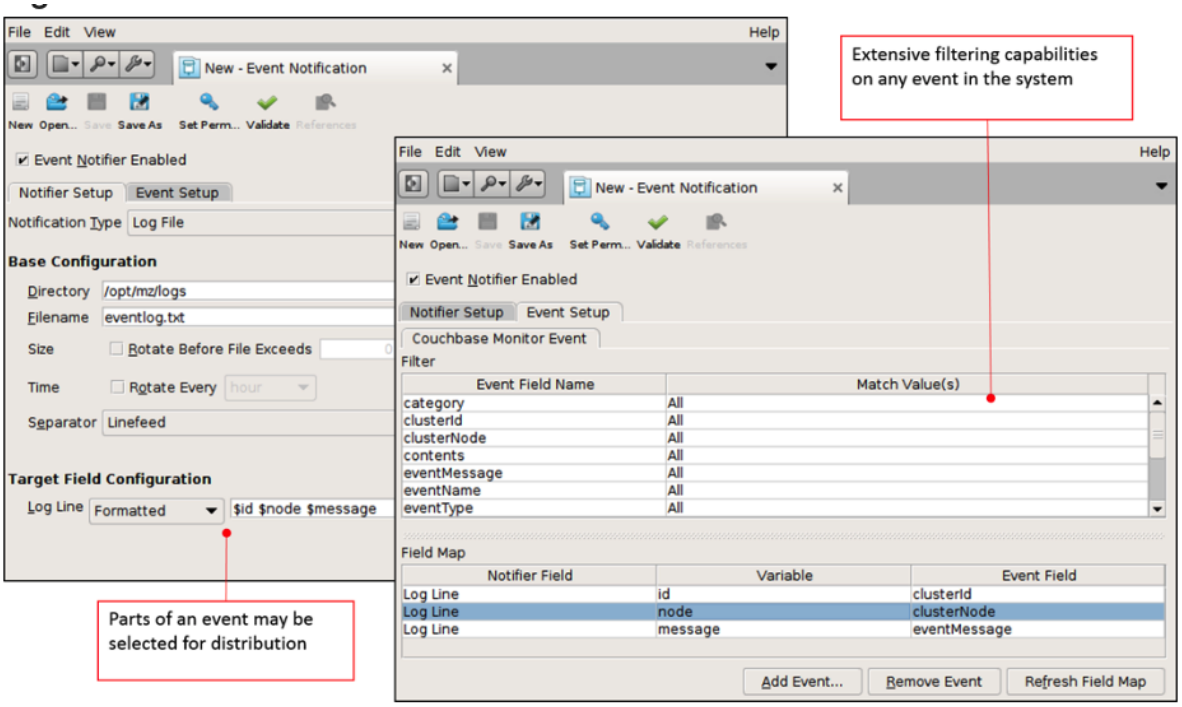
In Usage Engine you can get notifications by configuration Event Notifications and alerts by using the Prometheus Alert Manager.

Event Notifications

Usage Engine includes an internal event broker, which is configured using the Event Notificationfeature. An Event Notification configuration is used to intercept standard, as well as user-defined, events. This information can subsequently be dispatched to any number of target applications.

The Event Notification configuration acts as a filter that collects and streams events to any of the following target systems:

- Log file
- Database
- E-mail
- SNMP Trap receivers
- Usage Engine System Log



Event Notification Configuration

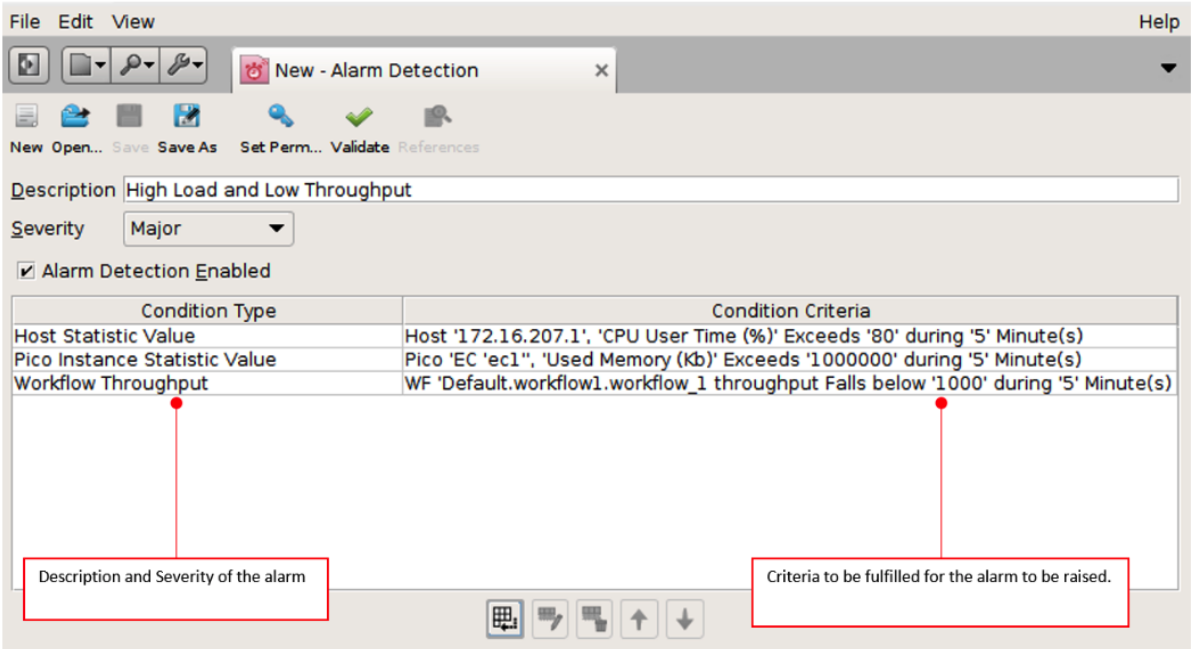
Alarms and Alerts

Since Prometheus is the central integration point for metrics it is straight forward to also implement alert functionality here using the Prometheus Alert Manager. See https://prometheus.io/docs/alerting/latest/alertmanager/ for details.

Usage Engine does however also include off-the-shelf capability to proactively monitor many different parts of the system and raise stateful alarms if a configured condition is detected. Alarms can be managed in the operator's UI or be sent to any external source such as a network management system using protocols such as SNMP traps, or e-mail.

An Alarm Detection configuration is used to define the criteria that should result in the generation of alarms; such as workflow status change conditions, user or system status activity etc.

Combinations of multiple alarm conditions can be defined as triggers for an alarm.



System Statistics Application

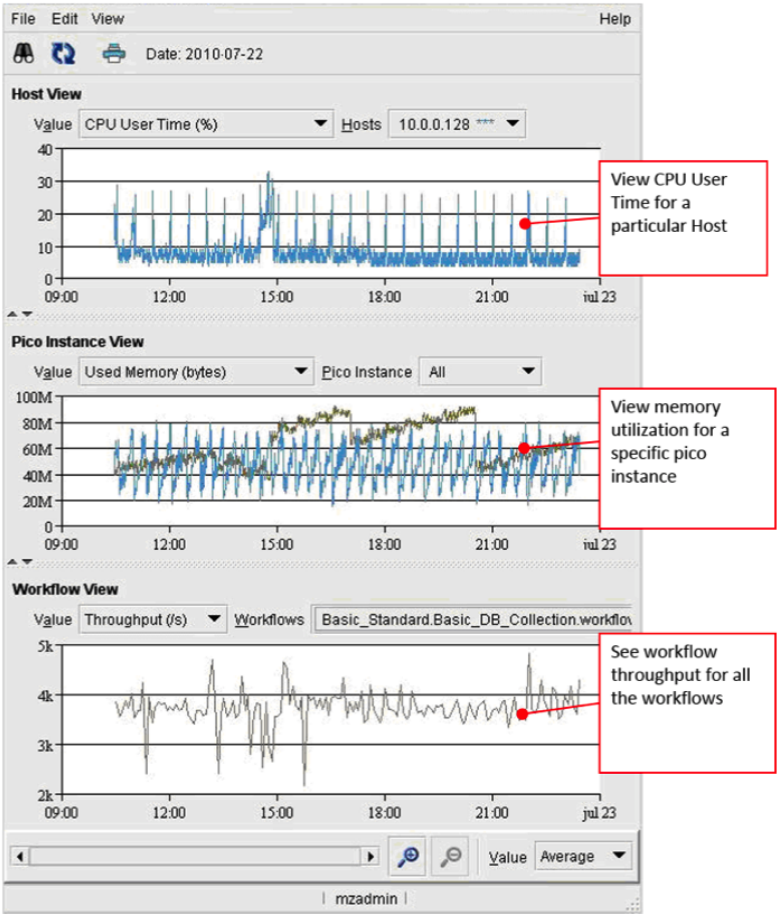
The System Statistics application in Usage Engine collects and consolidates execution information automatically. When the system is active, resource utilization and workflow processing is continuously monitored, and the statistics is saved in the internal database. Graphs can be viewed in the Desktop, DROnline, and – if installed – Grafana.

Desktop

Using the System Statistics UI, a user can select statistics for hosts, pico instances, and workflows for periods dating back two months. Available metrics:

Host	Pico Instance	Workflow
• CPU User Time (%) • Context Switch (/s) • Swapped To Disk (blocks/s) • Swapped In From Disk (blocks/s) • Processes Waiting For Run (#) • Processes In Sleep (#)	• Used Memory (bytes) • Maximum Memory (bytes) • Process CPU Time (%) • Open File Descriptors • Garbage Collection Count (#) • Garbage Collection Time (ms) • Thread Count (#)	• Throughput (/s) • Queue Throughput (/s) • Simultaneous (#) • Queue Size (%)

Example of Dashboard view in the Desktop:

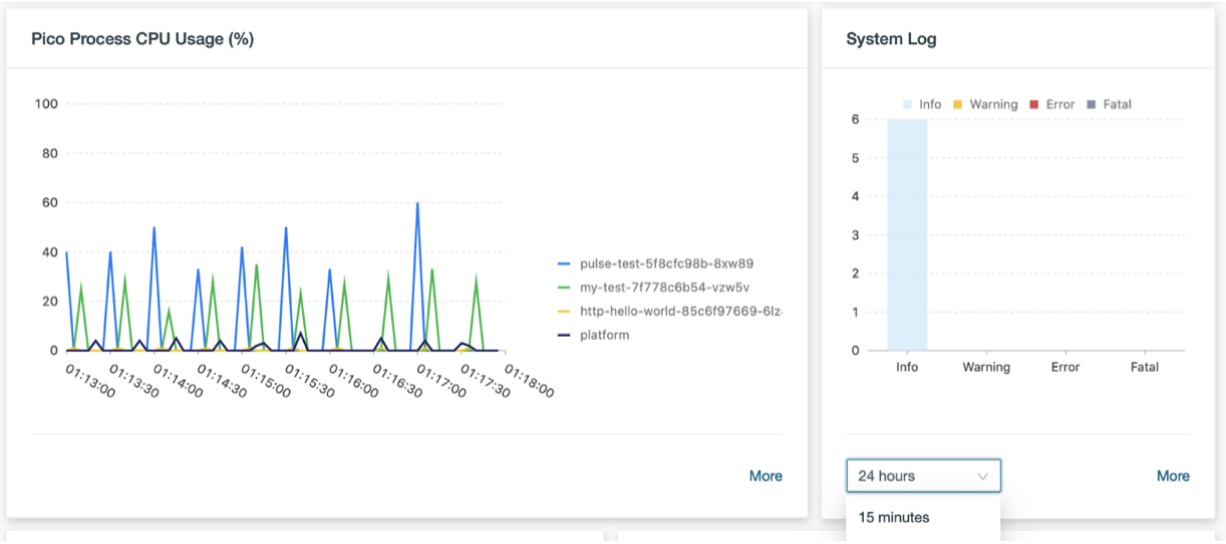


System Statistics Dashboard in Desktop

DROnline

The Dashboard view in DROnline, contains the following standard graphs:

- Pico Heap Memory Usage (%) for the last 5 minutes.
- Pico Process CPU Usage (%) for the last 5 minutes.
- Aborted Workflows.
- Unreachable Picos.
- System Log, number of messages per category and time.
- Pico Response Time (ms).
- Workflow Uptime.
- Workflow Throughput.



System Statistics in WebUI

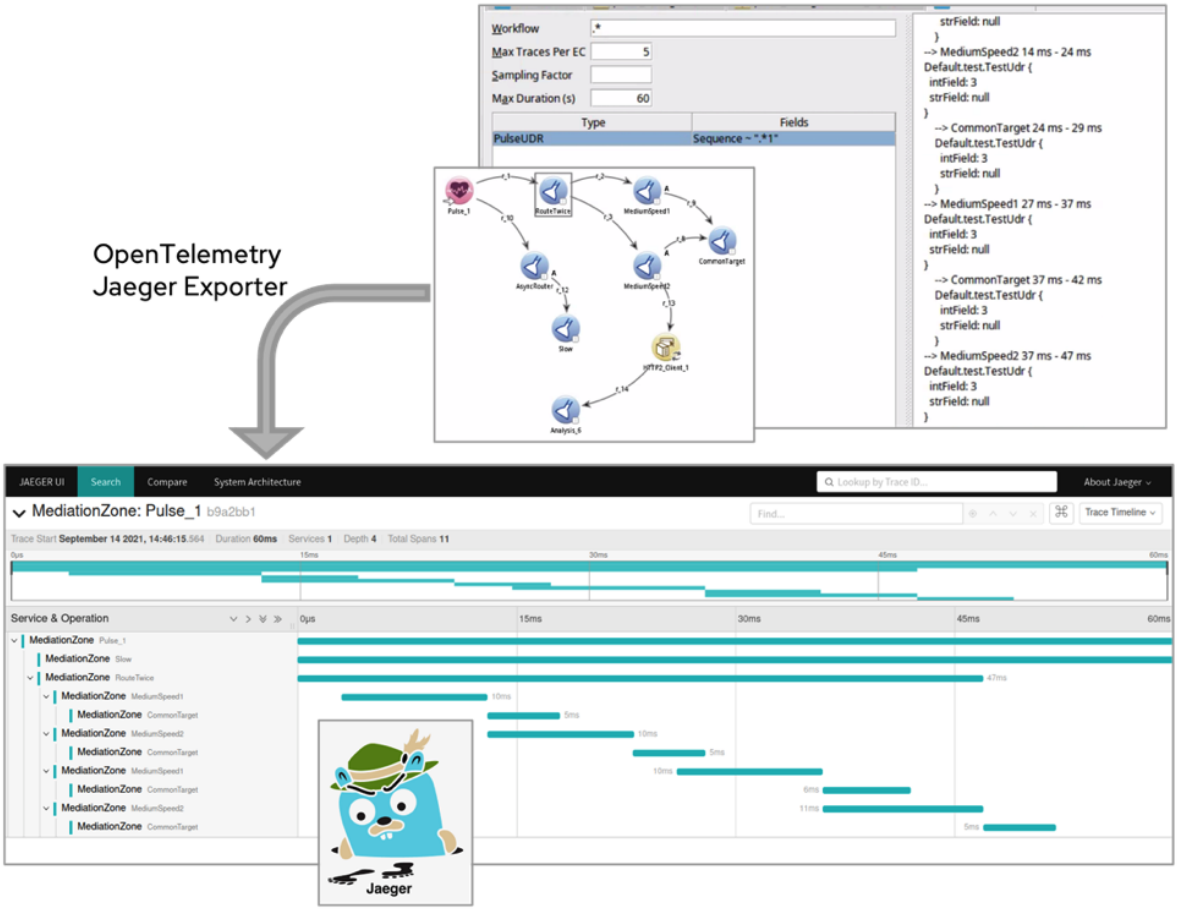
Grafana

Grafana can be used to create customized dashboards. Usage Engine includes a set of example dashboards to display System Statistics in a similar fashion to the Desktop view. **Please refer to section 7.2** for more monitoring options using Grafana.

Tracing

Conditional Trace makes it possible to examine records being processed in running workflows, making it a very powerful tool for troubleshooting in production environments. The main advantage is that there is no need to restart the workflows - you can at any time configure search criteria and turn on tracing for a workflow or set of workflows. Conditional Trace utilizes Open Telemetry Jaeger Exporter. For an optimal user experience the traces are examined using Jaeger, but you can also view them directly in the Operations UI, or export as files in JSON format. The configuration is a three-step procedure:

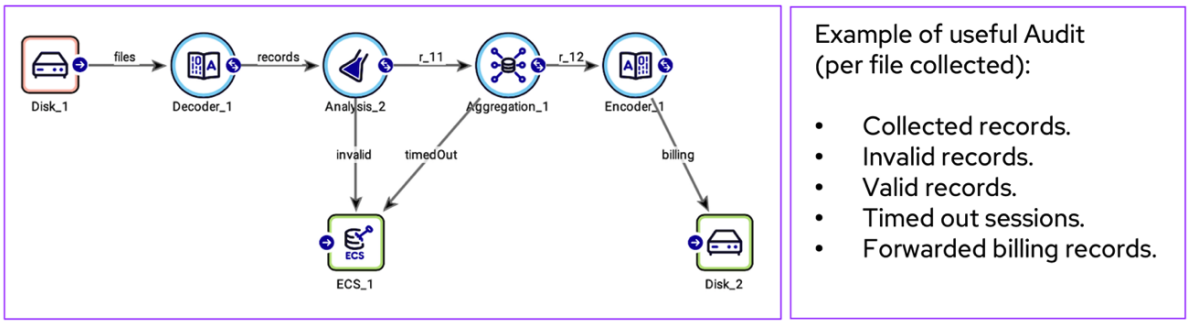
- Defining a Trace Template:
  Templates are configured in the Desktop and define which workflows they apply to, including the search criteria. To avoid getting too much data, templates also contain configuration to limit search results.
- Starting a Trace:
  Traces are started from the Operations UI. A user selects an existing template and sets the values for the required field/fields.
- Examining a Trace:
  Trace output records can be examined directly in the Operations GUI, exported in JSON format as txt files, or – for optimal usability – in Jaeger.



Conditional Trace Concept

Audit

The Audit functionality in Usage Engine enables gathering of exact processing metrics in batch workflows. A user defines customized database tables where audit data is saved by Usage Engine every time a file has been processed successfully. The only requirement for these audit database tables, is that they contain a column reserved for file identification. Examples of typical audit counters/metrics; number of specific records collected, forwarded, cloned, deleted, sent to error correction system, correlated, aggregated, etc. Also, size and time stamp of files collected, discarded, and forwarded is typically saved.



Usage Engine Auditing

Note that audit is ***exact*** metrics and can thus only be applied to batch workflows. If audit for real-time workflows is required, you need to forward data from the real-time workflow to a batch workflow.

Trouble Shooting

There are several ways to troubleshooting Usage Engine and solutions deployed on it.

Debug Printouts

Debug from Workflows can be viewed directly in the Workflow Monitor or be sent to a file. Debug is turned on/off in the Workflow Monitor UI, or by using CLI commands. There are standard events for some agents that will appear as debug – for example when files are collected, forwarded, and discarded, and when duplicate files and records are found. User defined debug events are defined from APL code. For each debug event, timestamp and originating node is automatically added to the message.

System Log

Events and errors are saved in the System Log, which is managed from DROnline. There are three categories of events originating from the Usage Engine system, from workflows, and from user activities. Besides the automatically generated logs, a user can configure system log messages from APL code. To avoid overflowing the log, duplicate messages are not repeated. The first and last occurrence is displayed, including how many times it was repeated.

Purging of the system log is done automatically and using a predefined clean up task. The frequency is configurable.

Data Veracity

When unexpected things happen during processing of payload data, for instance decoding errors or unexpected type codes, Usage Engine provides a powerful subsystem called Data Veracity to help resolving the error condition.

Data Veracity is used when UDRs and Batch files fail validation and manual intervention is needed before they can be successfully processed. Erroneous UDRs are sent to database tables, following the structure of the UDRs.

Refer to chapter 11 (Sub Systems) for further detail on Data Veracity.

Error Correction System (ECS)

To preserve compatibility with configuration migrated from older MediationZone systems, Usage Engine also supports ECS as an alternative to Data Veracity. ECS does not provide as much functionality for working with erroneous data as Data Veracity.

Refer to chapter 11 (Sub Systems) for further detail on ECS.

Inspectors

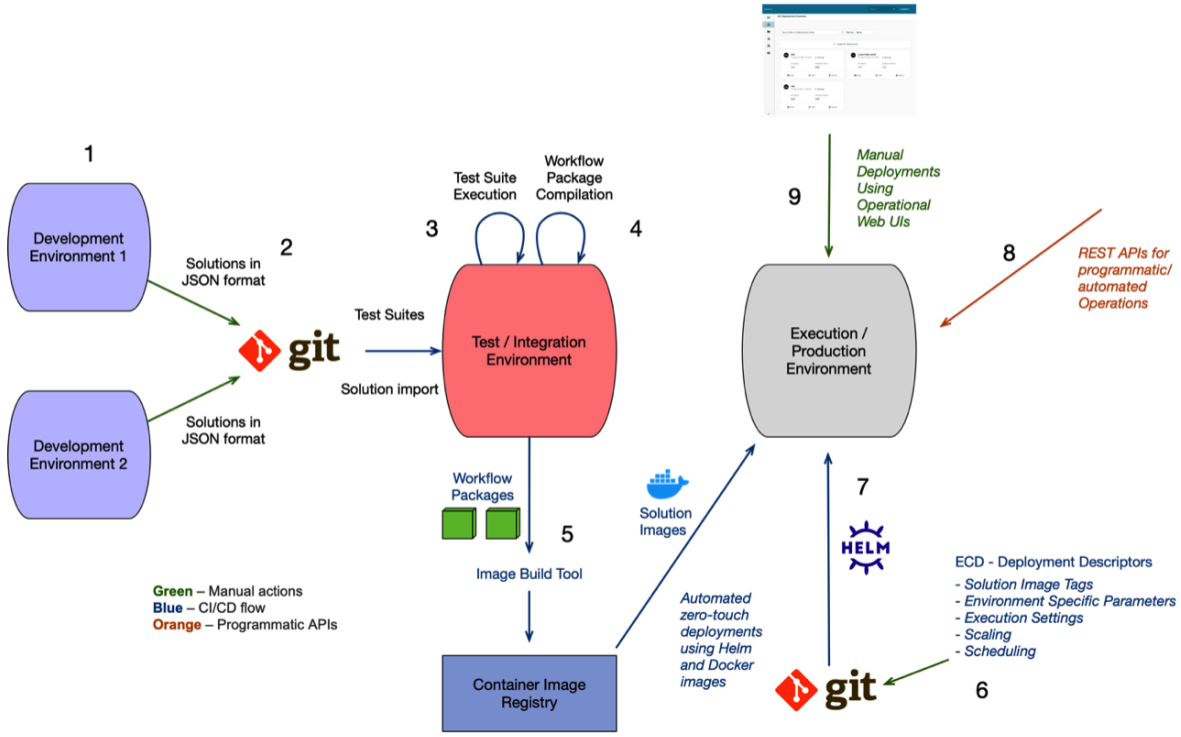
Usage Engine provides inspectors to give insight into data at rest. Inspectors exist for Aggregation storage, Duplicate DUR detection, Duplicate Batch detection and Data Veracity.

Continuous Integration/Continuous Delivery (CI/CD)

CI/CD is a process for delivering code to production by introducing automation into all stages of deployment. Specifically, CI/CD introduces ongoing automation and continuous monitoring throughout the lifecycle of code, from integration and testing phases to delivery and deployment. CI/CD principles:

- Less manual tasks increase quality and lets testers and developers focus on their core tasks.
- Frameworks encourage good practices.
- Automation increases time to market, frequent rollouts, and encourages creativity.

The picture below shows the automated flow of solutions going from development, via test and into production.



CI/CD Model and Structure

The steps shown in the picture are:

1. **Development Environment –** Solutions are created in a development environment. These can be hosted locally on the designer's laptop, as a tenant in a shared private cloud infrastructure or in a public cloud infrastructure. The designer builds the solution using the Usage Engine low coding tools. He/she tests the solutions locally by executing the workflows using parameterization that work in the development environment.

2. **Source Control** – When the designer is ready with the solution, he/she triggers an export of the solution in a JSON format, suitable for version control. The JSON data is committed and pushed to a version control system like Git.

3. **Test Suite Execution –** A CI/CD automation tool like Jenkins, GitHub actions or similar acts on the newly committed data and starts the test and build pipeline. The test pipeline starts up a Usage Engine test/integration environment and executes a suite of Python tests to verify the solution. The Python tests are implemented using the Usage Engine TestKit.

4. **Packaging –** If the test suite execution is successful, the pipeline triggers the build step. The build step transforms the solution into compiled Workflow Packages which contains the full solution, including compiled code that is auto-generated from the implementation.

5. **Image Generation** – The Workflow Packages are then used as input for image generation. A tool such as docker or nerdctl is used to build a container image readable by the container runtime in Kubernetes. Ready images are stored on a container image registry, such as Docker DTR, Amazon ECR or similar.

6. **ECD Descriptor Implementation** – ECD descriptor files are parameterizations of ECD Helm charts, i.e., a YAML formatted files of parameter values. These are designed by an operations team. The ECD descriptor files tell Usage Engine how the solution in the Workflow Packages should be deployed, interact with external systems and orchestrated. ECD descriptors are stored in version control.

7. **ECD descriptor deployment** - The CD part of the CI/CD pipeline makes sure that the ECD descriptors are deployed in the target environment and turn the Workflow Packages into executing micro services. This is implemented using tools like Helm or ArgoCD.

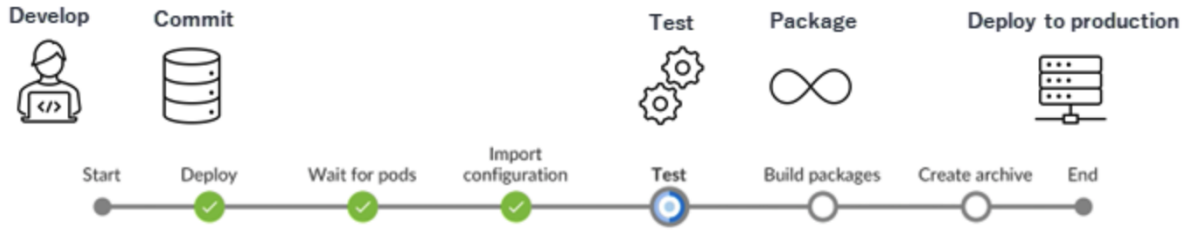
8. **Automated operations –** An executing solution generates many kinds of metrics and other operational data. External monitoring systems can be configured to act on this data and feedback to the application and solution over the provided APIs, thereby closing the automation feedback loop.

9. **Manual observation** – Apart from automation Usage Engine also provides powerful UIs to monitor and operate the solutions. Also, tools like Grafana, Kibana, Jaeger, and Kiali can be used to visualize and build dashboards on metrics, log data, tracing data or dataflows.

CI/CD Example Pipeline

A CI/CD pipeline is a series of steps that must be performed in order to deliver a new version of software. Continuous integration /continuous delivery (CI/CD) pipelines area practice focused on improving software delivery with DevOps in focus.

Usage Engine includes an example pipeline, that has been developed using Jenkins as orchestration tool and bitbucket as code repository. Specifically, the pipeline consists of a set of scripts, and a code repository layout, that can easily be adapted to similar tools of preference.



The included pipeline contains a set of examples, showing typical procedures for maintaining configuration in any Usage Engine system in production:

- A fully functioning realtime workflow, complete with test workflows.
- Storage, test, and deployment of new workflows.
- Rolling upgrade of running workflows.
- Canary upgrade of running workflows, using Istio service mesh for traffic load balancing between old and new version of workflows.

CI/CD Test Framework

A Test Framework is a vital component of any CI/CD pipeline. Usage Engine has a built-in test framework, making it convenient for developers to create test cases in parallel to developing use cases.

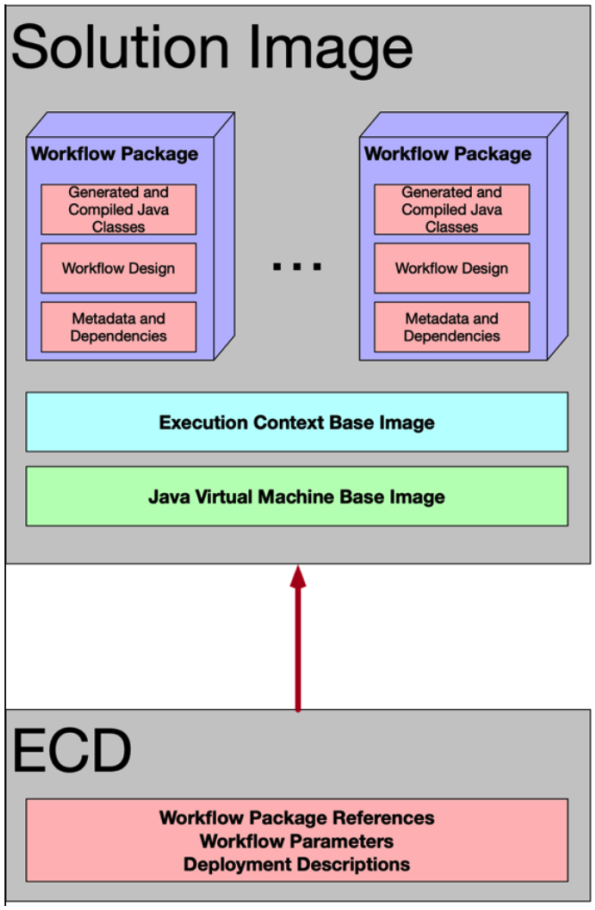
The tests are developed in Python and executed as Task Workflows. You can test workflows, or individual APL functions configured in the “APL Profile” UI. There is a rich set of supporting functions, for example:

- Initiating tests with test data using Arguments (if testing functions), or data files.
- You can also instantiate UDRs in the framework and populate with test values.
- Verification is done on return values (if testing functions), workflow states & events, and output files.
- The results are viewed directly in the Task Workflow UI as debug, or in files.
- You can also produce report files in a format that is readable by Jenkins.

Note! The Test Framework is used internally at DigitalRoute for automatic Unit Testing of features.

CI/CD Workflow Packages

Workflow Packages are a way of deploying configuration to production and forms a corner stone for the microservice concept. They are pre-compiled, self-contained, and read-only. One or more Workflow Packages built into a container image and referenced from an Execution Context Deployment definition form a scalable microservice, with easy and fast upgrades, safe rollbacks, and the possibility of having multiple versions of the same package active simultaneously. This means zero downtime upgrades using blue/green or canary deployments can be done.



Execution Context Deployment

The classic way of handling configuration in DigitalRoute's product for bare-metal and virtual machine deployments, MediationZone, are regular configuration exports. Those are also available in Usage Engine to provide product level support for migrations of existing on-premises deployments to cloud infrastructure and the Usage Engine. The main differences are outlined in the table below.

Standard Configuration Exports	Workflow Packages
Configuration is compiled when imported. This may be time consuming and introduces a new version.	Configuration is pre-compiled, and version is set. A plug-and-play approach.
Updating shared configuration in production will affect all workflows using it when they are restarted.	"Shared" configuration is part of the package, and valid for that package only.
Rollback is achieved by importing the previous configuration.	Easy rollbacks – packages are an integral part of the image/pod and can easily be switched back to the previous pod.
Only one version of configuration may be active.	Several versions of packages can be active, enabling blue/green and canary upgrades.

Usability

Usage Engine has been designed with two main user personas in mind. The workflow designer, and the system operator. Functionality geared for either persona is outlined below.

Designer

The designer configures workflows and creates the solutions running on Usage Engine. The main user interface used by the designer is the Desktop, where configuration related to workflows is created and managed. Features typically used by the designer include:

- Workflow Editor – creating and editing workflow configuration
- APL code editor – creating and editing shared APL code
- Profiles – creating and editing common configuration used across agents and systems
- Format (ultra) definitions – creating and editing format definitions for inbound and outbound data feeds

- Aggregation rules – creating and editing usage aggregation and correlation rules
- Test framework – creating and editing unit tests, verifying the correctness of configuration items as part of a test suite

Operator

The operator monitors running workflows along with the overall system health and performance. Operators can also manage and control the deployment of created configuration, through management of the EC Deployments. The main user interface for controlling and operating the running system used by the Operator is the Web UI. For monitoring purposes, Usage Engine integrates with Prometheus and Grafana where centralized application monitoring typically resides for multiple applications.

Workflow

Workflows consist of software agents, each providing specific functionality, that is linked into automated data flows of virtually any complexity. An intuitive and powerful drag-and-drop management user interface covers all aspects of workflow life-cycle management.

Using the workflow concept, interface and processing agents can be configured, deployed, and maintained in a modular way. Workflows may also be inter-linked into scenarios of virtually any complexity, where the output of one workflow is the input of another. Workflow hierarchies can thus be defined, from technology-specific collection flows to convergent service-specific flows for downstream systems.

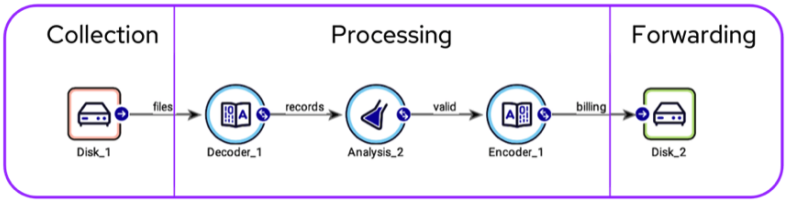
Once configured, workflows are automatically distributed on designated servers that are part of a deployment and executed in accordance with defined scheduling criteria. High availability capabilities ensure that real-time workflows are executing at all times. Batch mode workflows can be configured to execute at period intervals. Execution plans can be defined to ensure workflow execution according to internal strategies and processes.

Workflow Types Description

There are three different types of workflows: batch, real-time and tasks. Each have a slightly different execution behavior, but their modelling and configuration structure is the same.

Batch Workflows

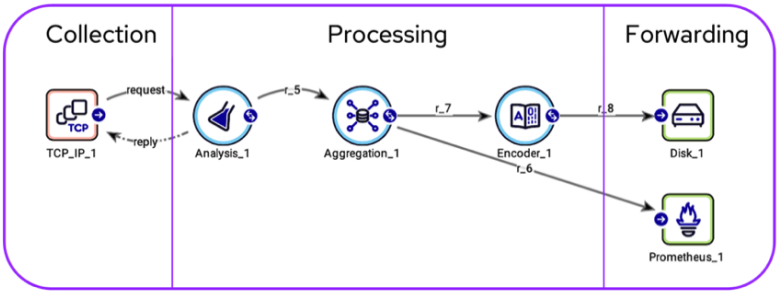
Batch workflows are used to collect process and distribute file-based data, also referred to as an offline model. These workflows can be configured for multi-threaded execution, enforcing a first-in-first-out processing order, and a strict transaction boundary based on each batch processed. Batch workflows are transaction safe as data processing can be rolled back and reprocessed if required.



Batch workflow

Realtime Workflow

Real-time workflows enable online processing of requests/answers with other systems. These workflows enable execution of large numbers of independent execution paths simultaneously using a multi-threading execution model.

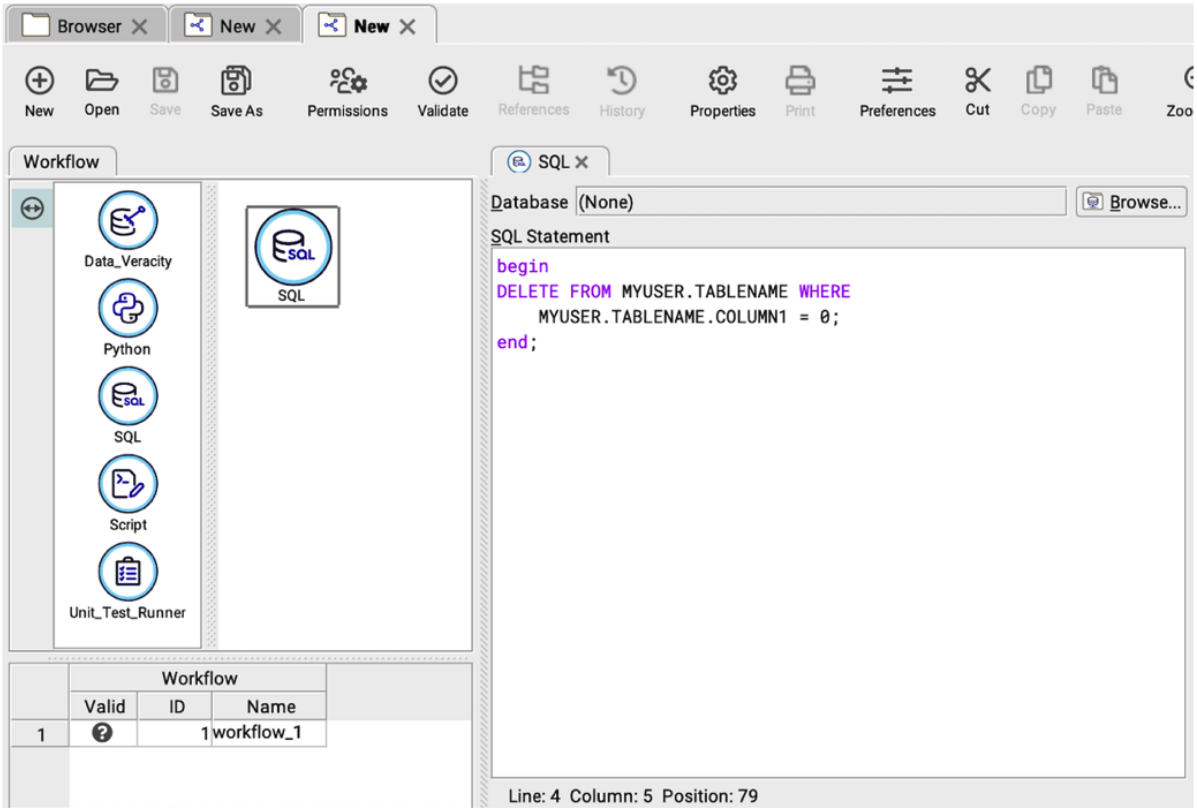


Realtime workflow

Task Workflow

Task workflows are used to execute common activities such as test, cleanup or maintenance tasks. A number of System Tasks are pre-configured in Usage Engine and can be complemented with any user-defined activity. These user defined tasks are scheduled and executed through Task Workflows, using one of the following agents:

- **Data Veracity** – Used for repairing/altering UDRs sent to the error storage Data Veracity.
- **Python** – Executes Python script.
- **SQL** – Executes SQL statements or scripts.
- **Script** – Runs a shell script or other executable.
- **Unit Test Runner** – Executes unit tests (Python scripts).



Task workflow

Workflow Configuration

The workflow configuration is a central part of the Desktop GUI. This is where all workflows are designed and configured by adding agents and connecting them to each other to form a data flow.

The workflow configuration operates in two modes:

- Design mode – where workflows are created and configured
- Monitor mode – where workflows are started or stopped, and the status of its individual agents is monitored

A workflow that is fully configured will have access to all modes. This is determined when a workflow is opened and saved. If the workflow is not fully configured or deleted, the Monitor and Profiling modes are disabled.

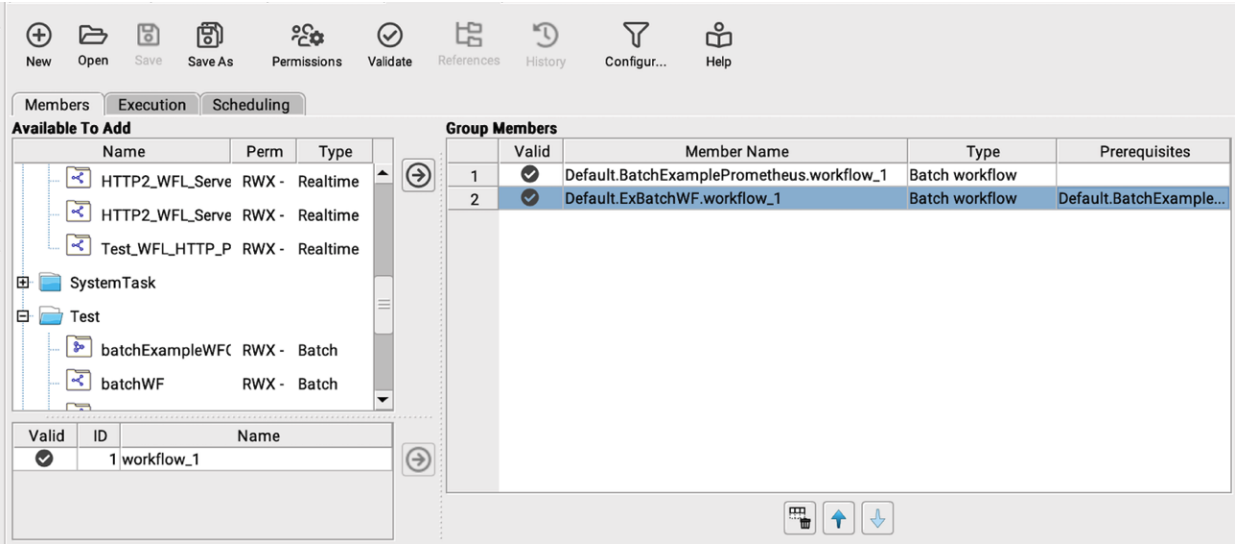
Workflow Group Configuration Description

The Workflow Group configuration enables management of workflows.

A workflow group can consist of one or several workflows, each with a diverse setup of scheduling, load balancing, and event notifications. Workflow groups enable you to configure these as a single entity.

Groups are either of type batch or real-time, which means that batch and real-time workflows cannot be mixed in the same group.

The example below shows a workflow group consisting of two batch workflows with a dependency (the first one must finish before the second can start).



Workflow Group Editor

Grouping of workflows can be useful in the following scenarios:

- There are dependencies between workflows within one line of business (e.g., collection, processing and forwarding) which can be managed through prerequisites, and the workflows need to be executed in a certain order.
- To limit the resource usage when executing groups with multiple workflows in parallel, you can control the maximum number of simultaneously running workflows.
- If there is a need to complete all collection before beginning the processing step, this can be achieved by creating a collection group and a processing group that belong to a super group.
- To simplify the monitoring as the groups, show their status in the Execution Manager.

Runtime Modification of the Workflow Configuration

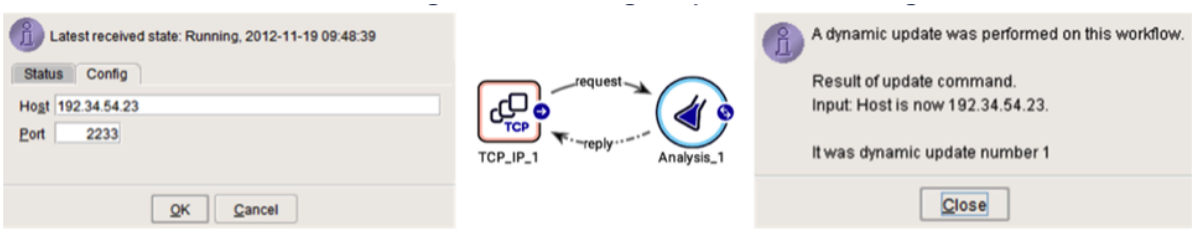
Usage Engine includes two mechanisms by which an operator can modify a workflow that is already running.

While a workflow is executing, it is possible to change the configuration of certain agents in their respective **Configuration** tabs which is available in the Monitor mode of the Workflow Designer.

These changes will be automatically sent to the running workflow and provisioned into the agent that is being reconfigured.

In this case, the changes are applied without affecting the execution of the workflow.

The example below shows how this is performed for the TCP/IP Agent of an active Workflow. In this scenario, the IP address used for listening on incoming requests is changed.

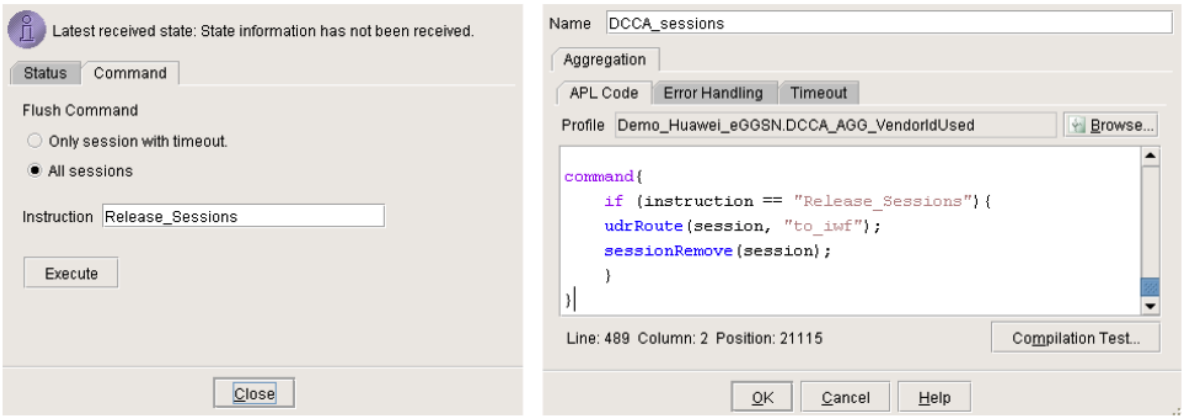


Runtime modification

Usage Engine includes a mechanism through which it is possible – while the workflow is running – to send a signal to an agent to instruct it to perform some action without affecting the active workflow.

The example below shows how this capability is used to instruct an Aggregation agent to execute the instructions within the command block for all sessions that have a timeout condition configured. This can be used, for example, to implement the capability to force a hard-flush of sessions in a Charging Gateway deployment.

Executing command block from monitor mode in the graphical user interface:



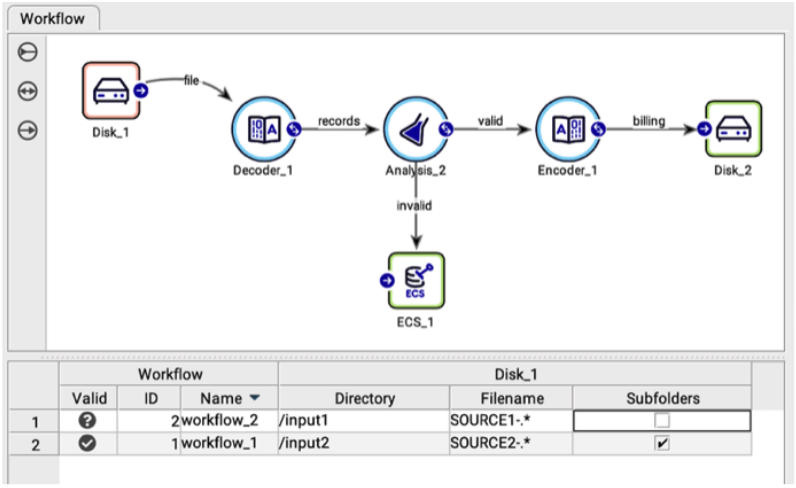
Command block

A command block can also be executed from the command line interface, by using the wfcommand. This enables the workflow logic to be executed based on input from the Command Line Interface.

Profiles in Workflow Table

The Workflow Table can be configured with:

- Manually entered values.
- External References, pointing to values in property files and databases. Note that most profiles can also activate parametrization, making those profile parameters available in the Workflow Table.



Workflow table configuration

Import and Export of Workflow Table

When managing multiple environments (e.g., test, pre-production, and production) it can be beneficial to export the different workflow tables to CSV-format. This allows one configuration export to be valid across several environments in combination with importing the appropriate CSV-file containing the specific configuration. Also, when managing many rows in the table, it can be a good idea to import /export the table to a CSV-format (for easy editing in Excel).

Workflow Management

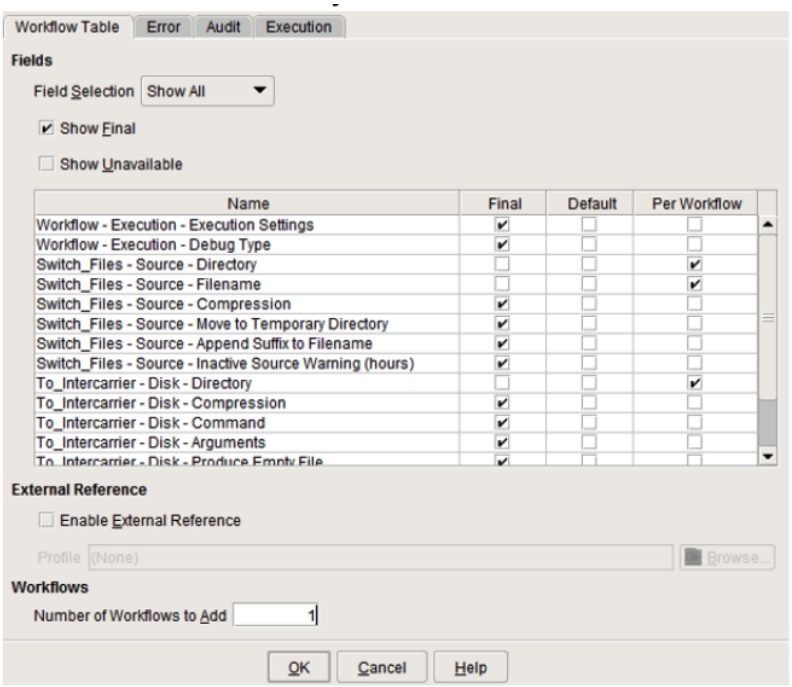
You can manage your workflows by creating multiple workflows interfacing similar sources and targets in a table format, and schedule execution.

Table-driven Workflow Instancing

The Workflow Table can be used to instantiate multiple workflows interfacing similar sources and targets. Although almost every configuration aspect can be parameterized, the start/end-point identifiers (e.g. IP addresses, directories, login, passwords, etc.) are normally the most important.

The example in the figure shows the **Workflow Table** tab in the Workflow Properties, where additional instances of the template workflow can be added. The parameters that are going to be available in the Workflow Table are configured in this tab as follows:

- Configuration parameters denoted as ***final*** are not shown in the Workflow Table and the final values from the template workflow are used in all workflow instances.
- Configuration parameters denoted as ***default*** are shown in the Workflow Table and have the value from the template workflow set as default.
- Configuration parameters denoted as ***per workflow*** are shown in the Workflow Table and are not set and should be set manually for each workflow.

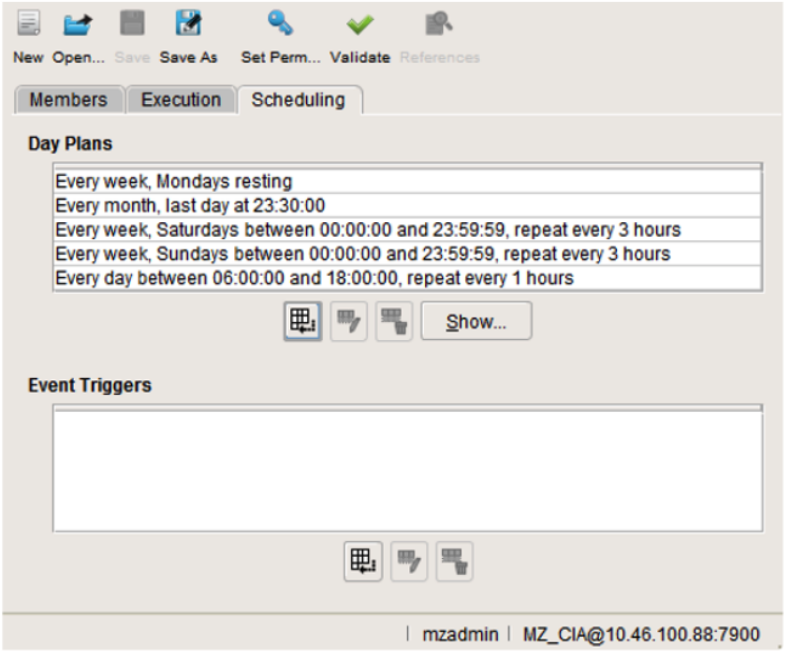


Workflow Table tab

Workflow Execution Scheduling

Usage Engine provides support for manual, real-time and scheduled execution of any kind of workflow group holding processing functions (i.e., collection, processing, distribution, or any combination thereof):

- To perform a manual execution, a group is executed through the Execution Manager.
- Real-time workflows are always active after they have been activated. If such a workflow terminates, it can either be manually re-activated, it can be re-activated by the scheduler or it can stay in state abort.
- For scheduled execution of groups, the frequency of collection and delivery of information in Usage Engine can be configured based on a period schedule (e.g., collect all files from a certain directory every 15 min). This schedule is based on day plans and can be configured to run every day, a specific weekday, or a specific day of the month. On each day, the activity can be specified to run only once or periodically at different intervals between various times.

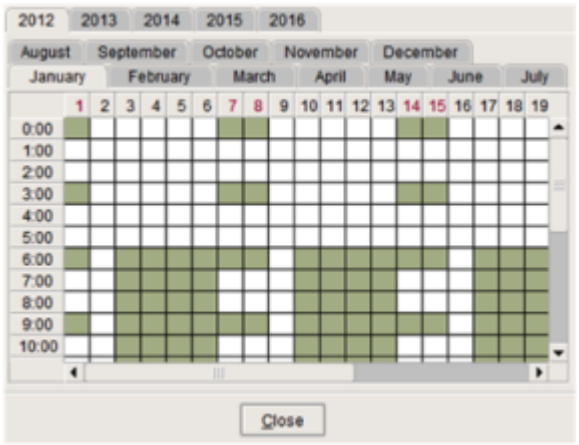


Scheduling

The scheduling criteria above instruct the system to collect/deliver data:

- If it is a Monday – do not collect/deliver any information.
- Otherwise, if it is the last day of the month, only generate collect/deliver information at 23:30.
- Otherwise, if it is a Saturday or Sunday, collect/deliver information every 3 hours.
- Otherwise, if the time is between 06:00 and 18:00, collect/deliver information every hour.

Usage Engine also provides the ability to graphically view when execution occurs during any day of any year, as depicted below:



Execution settings - graphical view

Workflow Execution

There are three main elements at play in terms of driving Workflow execution in Usage Engine.

1. The Execution Context Definition (ECD)
2. The Kubernetes Operator
3. The Usage Engine Platform process

The Execution model, and role of these elements depends on the nature of the workflow to be deployed. There is one model for batch and task workflows, and one model for real-time workflows:

Batch and Task Workflow Execution

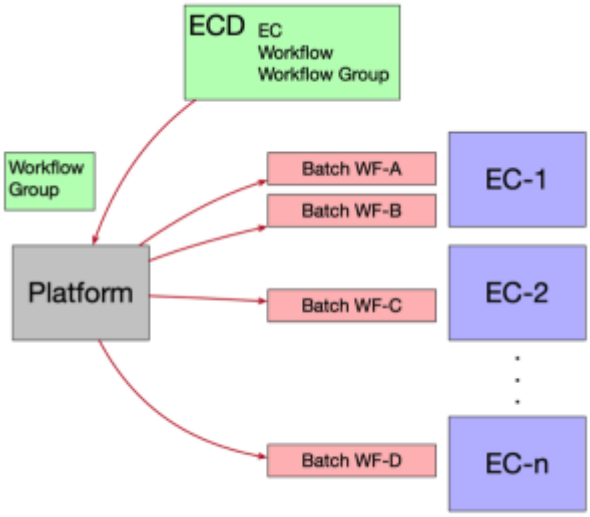
Batch workflow execution can be driven in the following ways:

Defining the Workflow Group Scheduling as part of the ECD definition

In this model, the workflow group scheduling rules are defined as part of the ECD configuration. The rules are then read and executed by the platform process. This is **only** applicable to Batch workflows

Defining the Workflow Group Scheduling as part of the Workflow Package

In this model, the scheduling defined in the Workflow Group Editor, is included in the Workflow Package, and is read and executed by the platform process. The model is applicable for Batch and Task workflows.



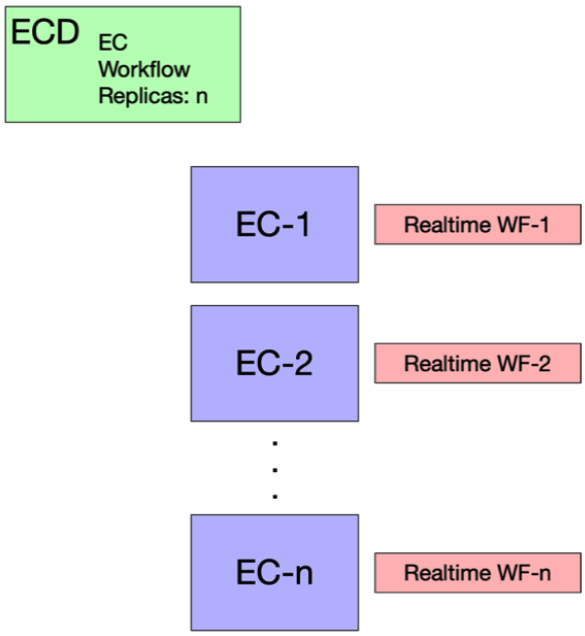
Batch workflow execution

Real-time Workflow Execution

Real-time workflow execution is driven from the microservice/Pod definition created by the ECD, which is managed and controlled by the Kubernetes Operator.

The Platform process does not play an active role in the scheduling and execution of the real-time workflow, this is instead managed by the Kubernetes Operator.

There is a direct mapping between the workflow and the pod. Hence, the number of executing instances of the real-time workflow maps directly to the number of Pods deployed, and controlled, by the Kubernetes Operator.

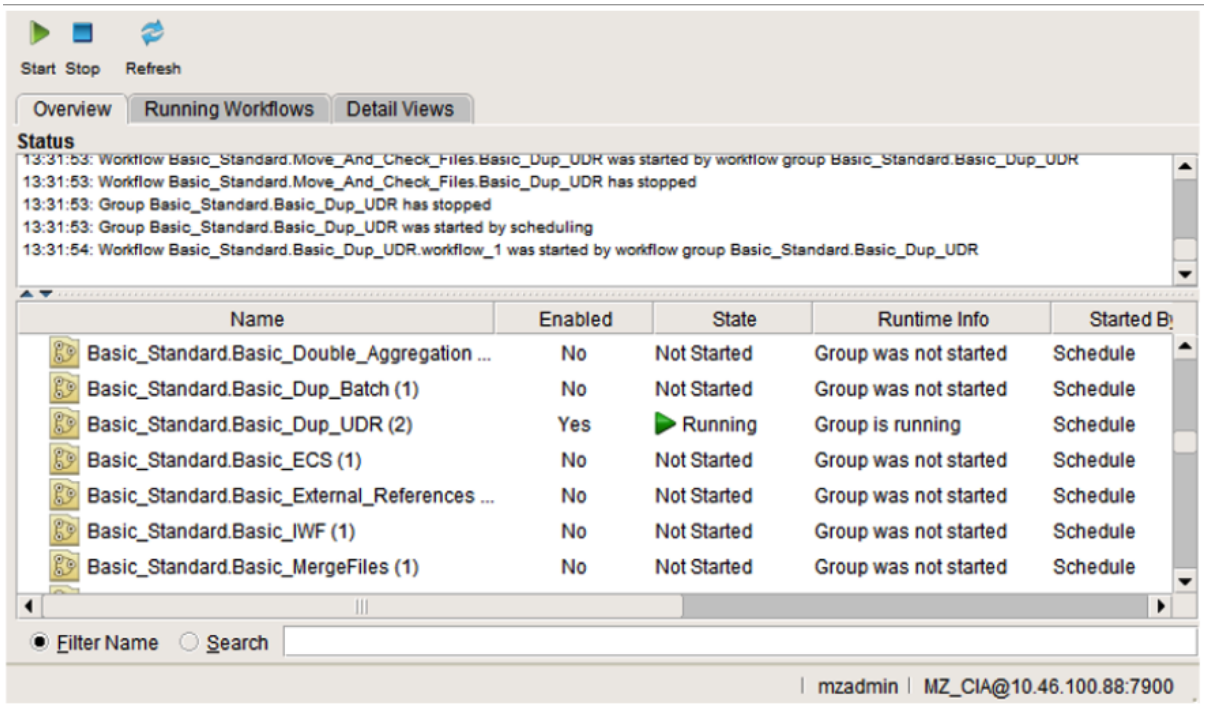


Realtime workflow execution

The Execution Manager

The Execution Manager provides a graphical interface to manage the execution of workflow groups and the monitoring of their status and schedule. It also provides views of running workflows and detailed views of the contents of the workflow groups including throughput and other statistics.

Execution Mangers exist both in the Developers and Operators UIs. Below is an example of the Execution Manager in the Desktop (Developer UI), showing a few workflow groups and their status and scheduling.

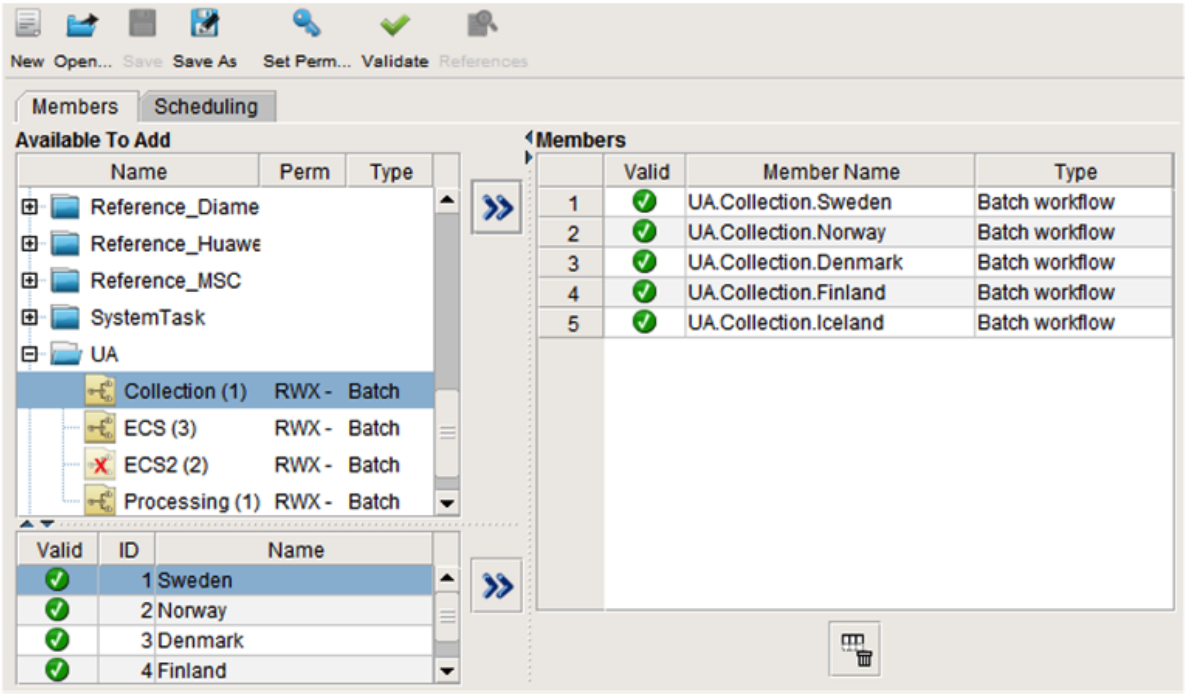


Execution Manager

Workflow Execution Suspension

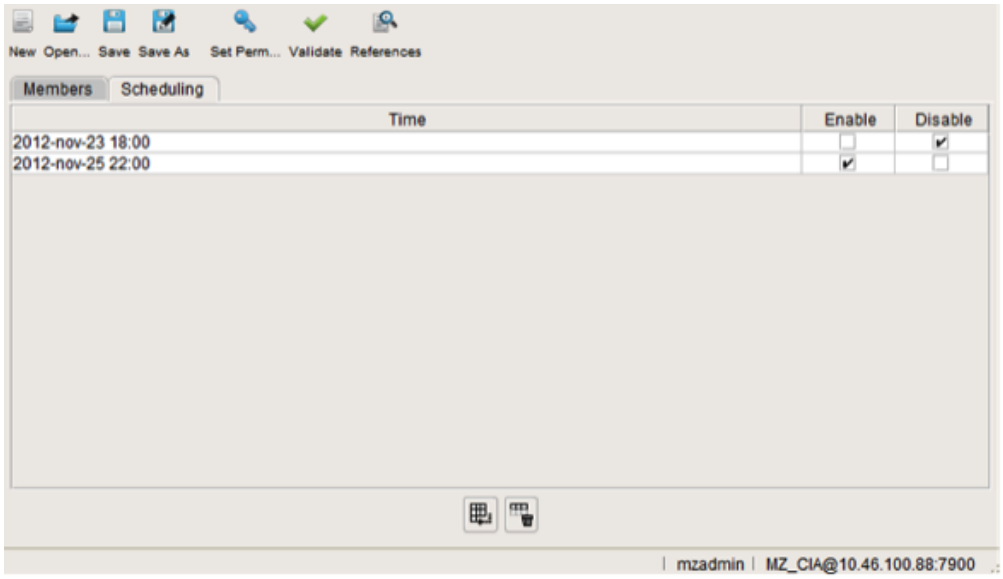
A Suspend Execution configuration enables you to apply a restriction that prevents specific workflows and/or workflow groups from running in a specific period of time.

In the example below, five workflows are grouped. Note that these are not workflow groups used for defining prerequisites or execution criteria, but rather groups only for the purpose of disabling and enabling at certain points in time.



Example of workflow group

The selected workflows are configured to be disabled over a weekend with a scheduled disabling and enabling of groups. This can be useful to disable workflows ahead of time when maintenance windows are planned.



Example of workflow group scheduling

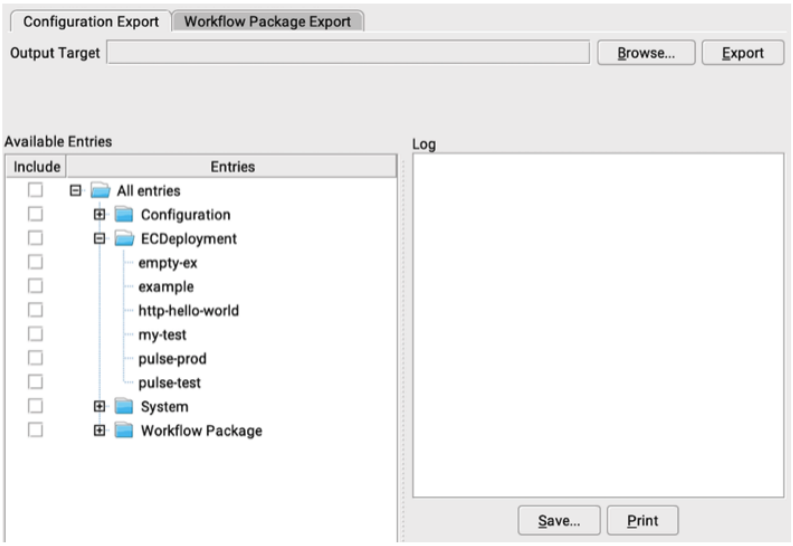
Configuration Import and Export

All, or parts, of the configuration related to a Usage Engine installation can be managed using the System Export/Import feature. An export /import can contain any of four categories:

- **Configuration**; workflow configurations in their original format (*).
- **ECDeployment**: Execution Context Deployments in their original format (*).
- **System**: system configuration in its original format (*).
- **Workflow Package**: workflow configurations in a compiled, immutable format. This format always automatically includes all related configurations to ensure the package can execute independently. This format is typically preferred when importing to production environments.

(*) The original format is the "source code" of configuration. When imported and started it will compile into the executable format.

Exporting configuration objects with this mechanism will create a compressed XML file that contains selected configuration. Exported information can also be protected through a one-way encrypted password.



Configuration export

Using the export and import feature, it is possible to develop and test all functionality on one system, then export it and later import it into another environment. This feature can be combined with the external references feature that allows definition of all configuration parameters in a text file which gives the possibility to use one configuration export file across all environments which all have their individual parameter file.

In addition to configuration data, the export/import function includes the capability to export runtime data from, for example, ECS and Archive systems.

External References (Parametrization)

External References enable loading Usage Engine with parameter values that originate from sources external to the workflow configuration:

- S3 Properties File.
- Environment Variable.
- Properties File.
- Properties Database. Note that these parameters and values are stored in the local Usage Engine database.

This allows Usage Engine system administrators to have specific variables for development, test and production clusters that work for the same export.

The example below shows a properties file containing mapping of values and variables that can be used in Usage Engine profiles, APL code and the workflow table.

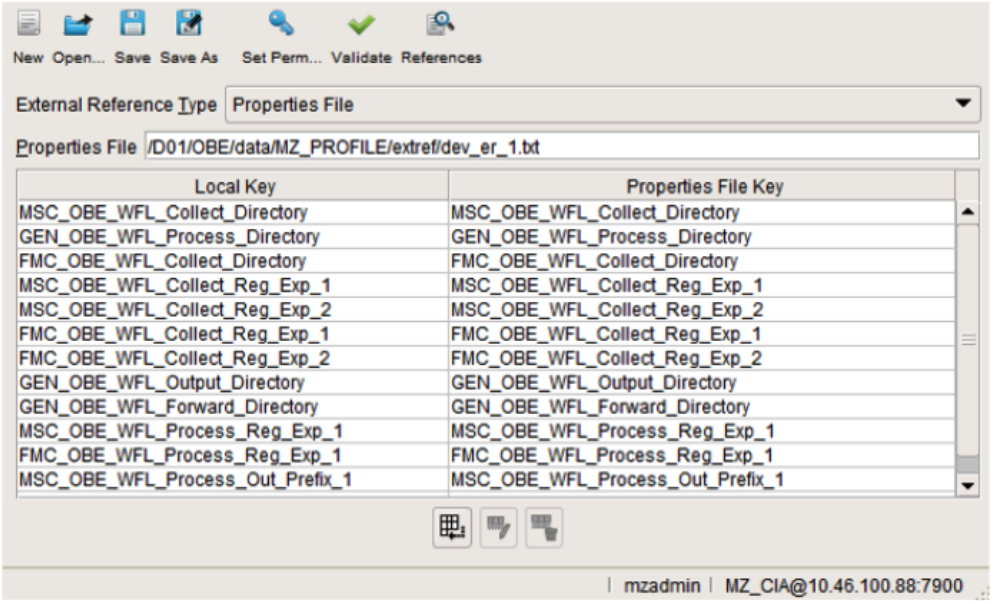
```
GEN_OBE_WFL_Process_Directory=/app_mzobep01/obe/data/MZ_PROCESS
GEN_OBE_WFL_Output_Directory=/app_mzobep01/obe/data/MZ_OUTPUT
GEN_OBE_WFL_Forward_Directory=/app_mzobep01/obe/data/MZ_FORWARD
GEN_OBE_WFL_Duplicates_Directory=/app_mzobep01/obe/data/MZ_DUPLICATES
GEN_OBE_WFL_Reprocess_Directory=/app_mzobep01/obe/data/MZ_REPROCESS

ALU_OBE_WFL_Collect_Directory=/app_mzobep01/obe/data/MZ_INPUT
ALU_OBE_WFL_Collect_Reg_Exp=^FRC\.ALU..\..*
ALU_OBE_WFL_Process_Reg_Exp=^FRC\.ALU..\..*
ALU_OBE_WFL_Process_Prefix=OBE_CTDC
ALU_OBE_WFL_Collect_Prefix_1=FRC.ALU00.

VMS_OBE_WFL_Collect_Directory=/app_mzobep01/obe/data/MZ_INPUT
VMS_OBE_WFL_Collect_Reg_Exp=^VMS\.CHUB..\..*
VMS_OBE_WFL_Process_Reg_Exp=^VMS\.CHUB..\..*
VMS_OBE_WFL_Process_Prefix=OBE_CTDC
VMS_OBE_WFL_Collect_Prefix_1=VMS.CHUB00.
```

External Reference file example

The image below depicts the External References Profile where properties files are selected and the variable names from the file are mapped to internal variable names.



External Reference profile

The local key variables can then be mapped in the workflow instance table by activating External References in the relevant cells.

	Workflow			Switch_Files		To_Intercarrier
	Valid	ID	Name	Directory	Filename	Directory
1	✓	1	workflow_1	GEN_OBE_WFL_Process_Directory	GEN_OBE_WFL_Process_Reg_Exp	GEN_OBE_WFL_Forward_Directory
2	✓	2	workflow_2	GEN_CED_WFL_Process_Directory	GEN_CED_WFL_Process_Reg_Exp	GEN_CED_WFL_Forward_Directory
3	✓	3	workflow_3	GEN_ASC_WFL_Process_Directory	GEN_ASC_WFL_Process_Reg_Exp	GEN_ASC_WFL_Forward_Directory
4	✓	4	workflow_4	GEN_QWC_WFL_Process_Directory	GEN_QWC_WFL_Process_Reg_Exp	GEN_QWC_WFL_Forward_Directory
5	✓	5	workflow_5	GEN_MMS_WFL_Process_Directory	GEN_MMS_WFL_Process_Reg_Exp	GEN_MMS_WFL_Forward_Directory
6	✓	6	workflow_6	GEN_SMS_WFL_Process_Directory	GEN_SMS_WFL_Process_Reg_Exp	GEN_SMS_WFL_Forward_Directory

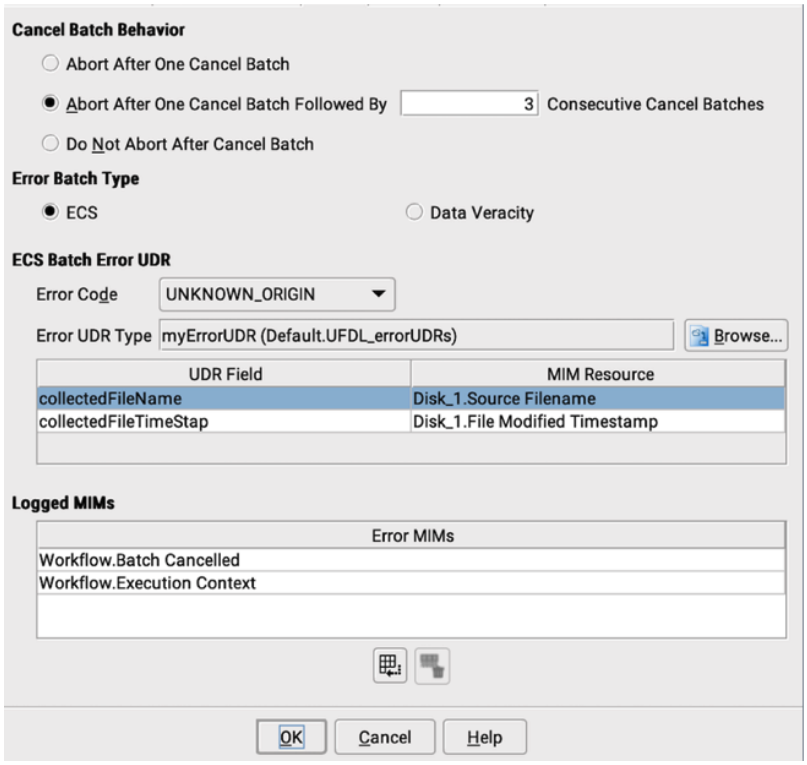
Workflow table

Workflow Tables can also be parameterized in the Execution Context Deployment YAML files.

Error Handling

Many types of error handling can be implemented using the workflow properties. Automated re-transmissions can be executed according to the preferred behaviour, e.g., make a specified number of retries before aborting the re-transmission attempts as configured in the example below.

All retries will be logged and after the final retry the erroneous file will be sent to the error correction system (ECS) together with the specified file attributes, e.g., timestamp, source file name, etc.

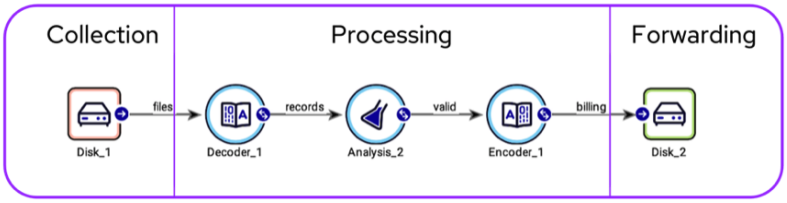


Workflow Properties Error tab

Agents Description

The building blocks of a workflow are the agents that provide the actual mediation functionality. The design of the workflow/agent model enables development of small, specialized agents, that when combined with other agents in a workflow provide comprehensive mediation functionality. Benefits of this approach include the ability to rapidly develop additional agents that complement existing agents, with minimal side effects.

Agents can also be identified as interface (collection and forwarding) and processing agents, where interface agents are used to integrate Usage Engine with external systems, such as network elements, billing systems, IN platforms, etc. They implement the protocols used for exchange of data and control information. Usage Engine provides a wide range of interface agents, which provides communication capabilities to standardized as well as vendor-specific protocols. For a list of interface agents, see Appendix A – Usage Engine Interfaces



Workflows consist of agents

Collection Agents

A collection agent is responsible for gathering data into the workflow from external systems or devices. An example of a simple collection agent could be one that reads a file from disk and sends the file contents into the workflow.

Processing Agents

A processing agent expects to be fed data and is expected to deliver data on one or many outgoing routes. A processing agent could be as simple as a counter that counts the throughput. It could also be more complex in that it aggregates, correlates, and consolidates the data and depending on the result delivers it on different routes.

Amongst the processing agents so called transformer agents can be identified. A transformer agent is responsible for translating an incoming byte stream into an UDR object or the opposite. For file distribution the Encoder Agent can be used to create header/trailer records containing meta-data of the file, e.g., record counter, check sum etc. This is commonly referred to as Decoding and Encoding, which the Usage Engine Ultra format system will handle.

Forwarding Agents

A forwarding agent is responsible for distributing the data from the workflow to other systems or devices. An example of a forwarding agent could be one that creates a file from a data stream and transfers it to another system using FTP.

Sub Systems

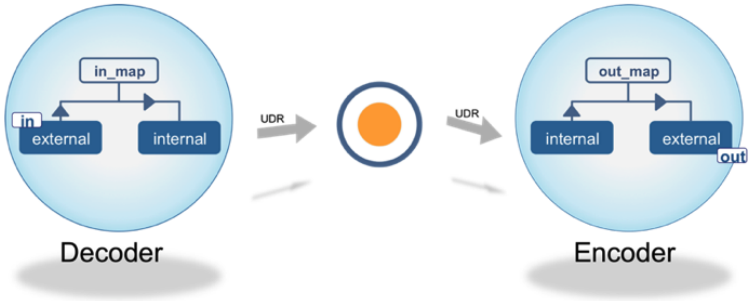
There is a wide range of different sub systems available in Usage Engine Private Edition, which are described in this section.

Handling of Data Formats (UFDL)

The Usage Engine Ultra formatting subsystem manages transcription of incoming data into internal records, and internal records into outgoing data structures valid for the downstream systems receiving data. The internal records are referred to as UDRs (usage data records) and can be processed by Usage Engine agents, such as the Analysis Agent and Aggregation Agent.

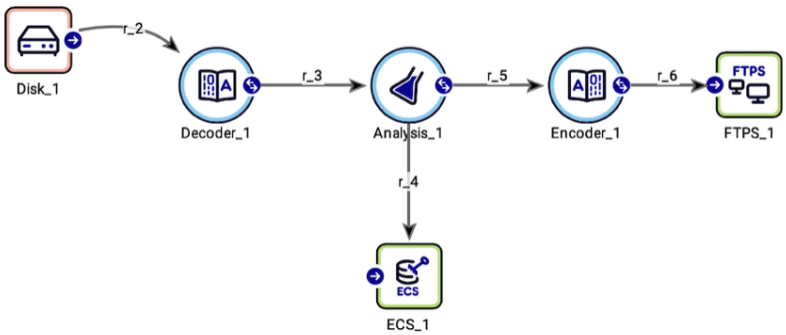
Ultra is configured with a language called the Ultra Format Definition Language (UFDL). There are 6 building blocks of UFDL:

- **External formats** - describes the way data is structured physically. This information is used by Ultra when decoding and encoding the data to include real-time requests and answer messages.
- **Internal formats** - describes a UDR (fields and types) that can be accessed from a Usage Engine agent
- **In maps** - describes how to map the information of an external format to an internal format (for a decoder)
- **Out maps** - describes how to map the information of an internal format to an external format (for an encoder)
- **Decoders** – a declaration of a decoder that converts an external format to an internal format
- **Encoders** - a declaration of an encoder that converts an internal format to an external format



UFDL handles conversion of data formats

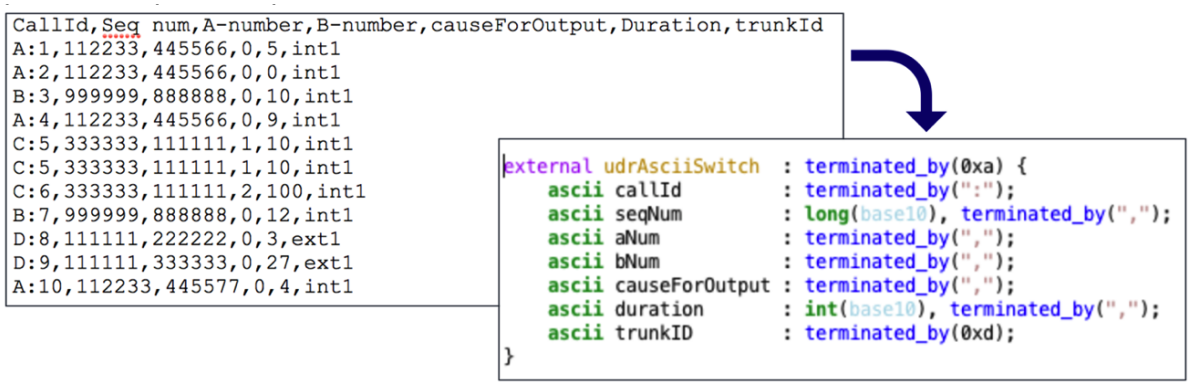
Example below of a workflow that collects data from local disk, decodes it into an internal UDR, then validates the data in an Analysis node. The validated data is either sent to the Error Correction System (ECS), or to be Encoded into an external format before forwarding.



Example workflow

Ultra includes support for, and has been deployed using, numerous different physical formats. Direct support is included for any format described in ASN.1 (BER and PER), and XML Schema (XML). Ultra also contains a specification language to describe other physical formats, such as proprietary tagged records, and fixed size records. Among other formats, this language has been used to provide support for AMA, INAMA, TAP2, TAP3, EMI, and a number of proprietary equipment vendor formats.

Important to note is that, based on format specifications, Usage Engine generates code for decoding and encoding functionality automatically. This means that there is no interpretation, with associated performance penalties, performed at runtime.



External sequential format

Proprietary Formats

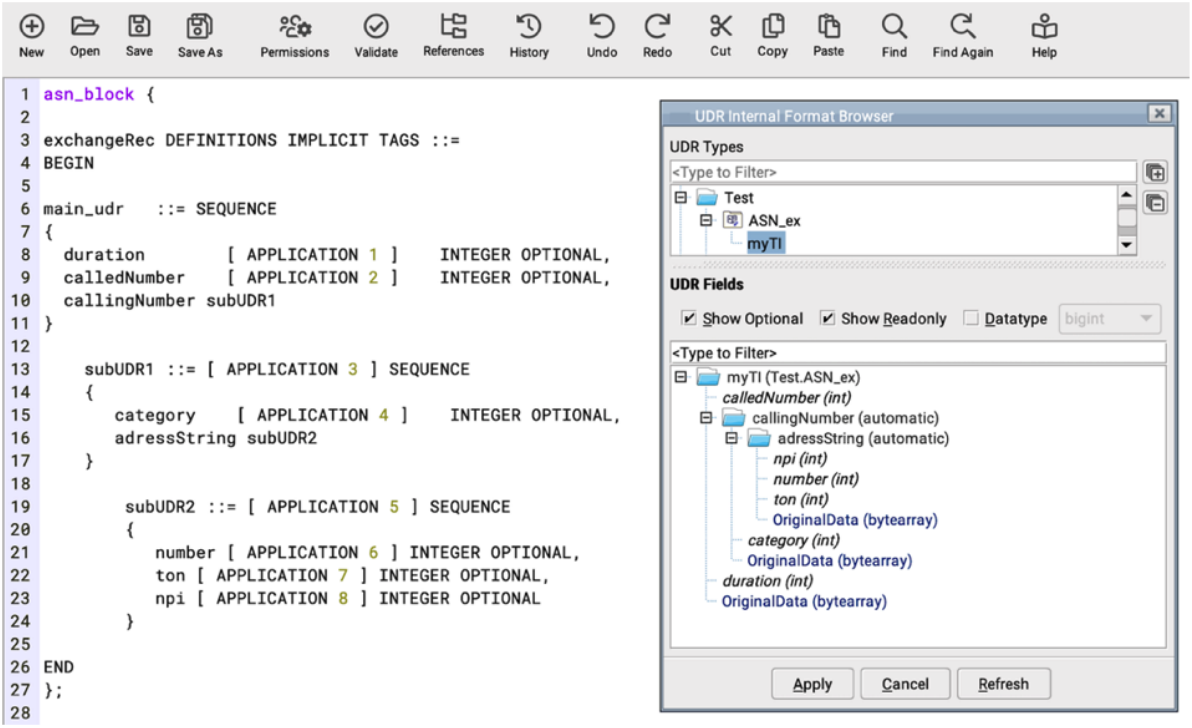
Any kind of proprietary format, ASCII or binary based, is supported. The fields comprising the structure can be of the following types:

- Integer types (int, bigint, byte, short, long)
- Float types (float, double)
- Raw data – bytearray
- BCD encoded (bcd; read from left or right)
- ASCII or EBCDIC

Field can have size specifications (static or dynamic) or be defined to be separated by a specific character. Field separators can be based on any character, HEX or ASCII. Additionally, UDRs can contain other UDRs as well as lists of primitive or composite types, recursively.

ASN.1 Formats

Usage Engine includes direct support for ASN.1 format specifications. Any ASN.1 specification can be directly imported into the system and a corresponding BER or PER decoder/encoder is automatically generated. Using this capability, Usage Engine can directly support any ASN.1 based specification, such as GSM, TAP/RAP, GPRS, etc.



ASN.1 format example

XML Formats

Usage Engine also includes direct support for XML Schema definitions. Any XML Schema definition can be directly imported into the system and a corresponding XML decoder/encoder is automatically generated. Using this capability, Usage Engine can directly support the translation from/to any XML document.

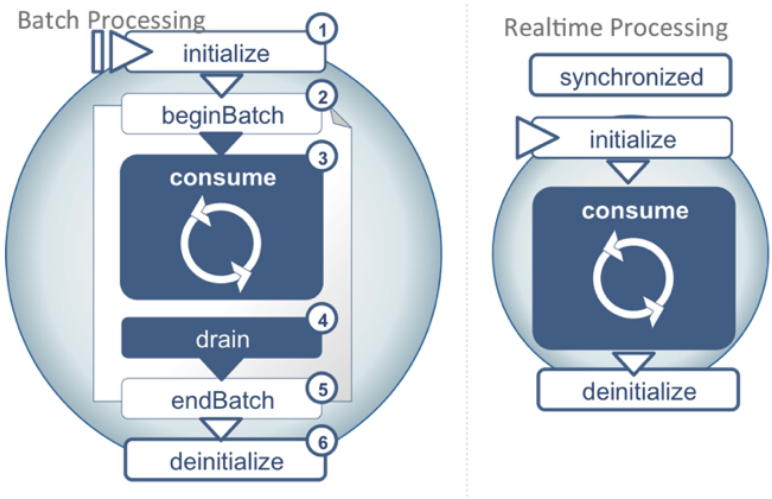
Processing Data (APL)

The Processing Agents provide different ways of processing data and can be divided into two categories; agents that are configured using UI parameters only, and agents that are configured using the Analysis Programming Language (APL). The Analysis and Aggregation Agents belong to the latter category.

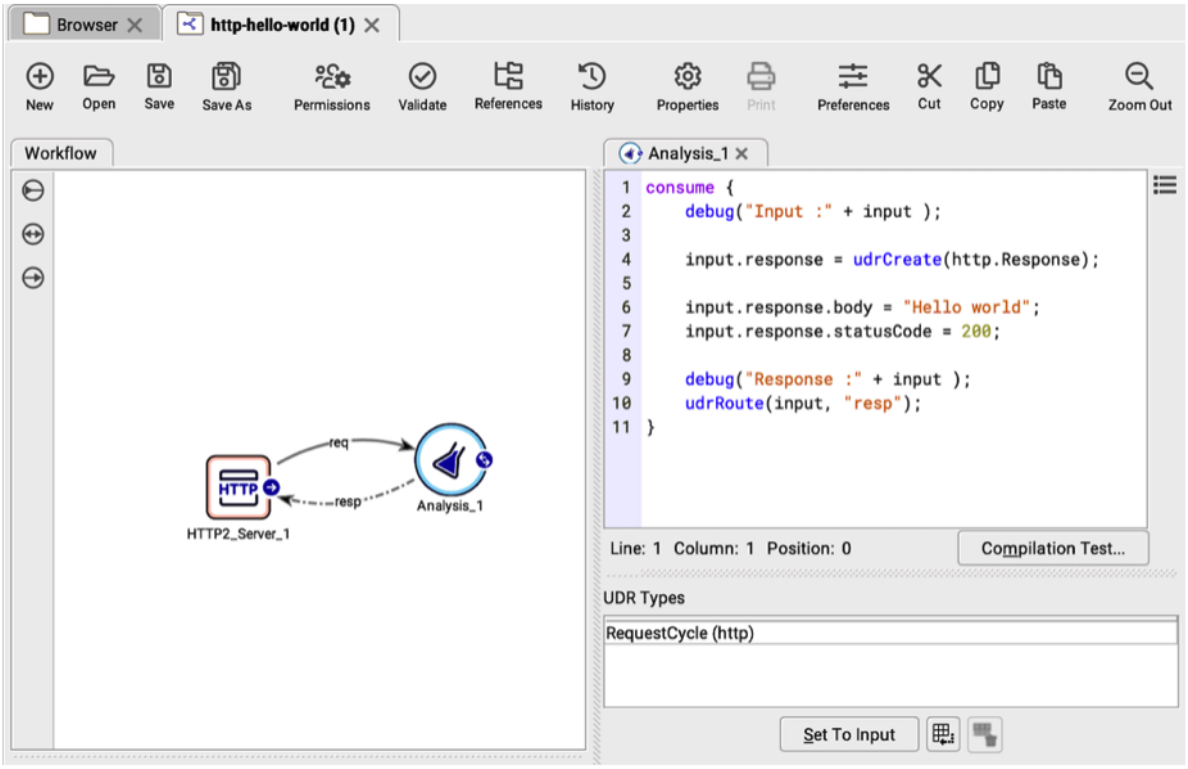
APL is a structural language with a syntax that to a great extent resembles Java or C/C++ programming languages. It supports the standard features of a programming language, plus interfaces to the Usage Engine platform enabling logging, auditing, access to the Error Correction sub-system (ECS), etc.

Function Blocks Description

APL code is divided into function blocks where each block is executed at different stages of the data processing. Number and type of function block executed depends on if the Analysis (or Aggregation) Agent is used in a batch or real-time workflow. Figure below illustrates the executed function blocks and their order depending on workflow type.



APL function blocks



APL code example

All logic configured in APL will be dynamically compiled to machine code at runtime for performance reasons. No interpretation of the logic is performed.

Besides being accessible from the Analysis and Aggregation agents, APL is also used in various applications in the system where flexibility is required. For instance, in Ultra Format Definition Language (UFDL).

APL is tightly integrated into the UDR and workflow models, enabling access and manipulation of runtime-, persistent-, and meta-data. Examples of such functionality include:

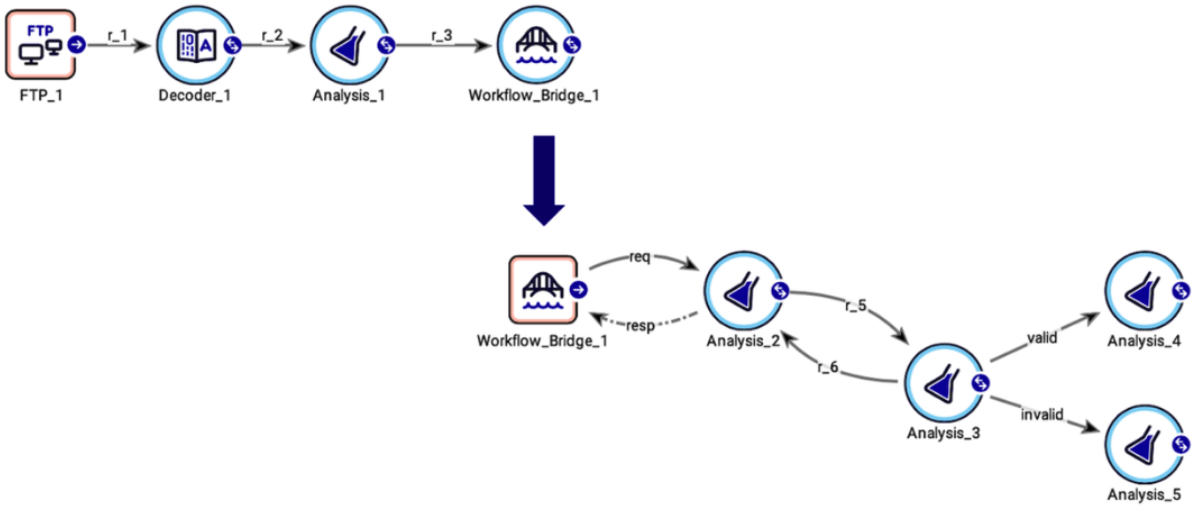
UDR operations	UDRs can be viewed, filtered, routed, created, cloned, decoded, encoded, and modified, etc.
	Meta information variables available for a workflow (MIMs) can be accessed, created, published, and assigned. Any published information is available to any agent within the workflow.

Meta Information Model (MIM)	
Error Correction System	UDRs failing specified validation criteria can be classified with respect to the error and be routed to the Error Correction System (ECS) for corrective action.
Audit	Audit and statistical information based on workflow execution can be logged in user-defined database tables.
Lookups	SQL/LDAP statements and calls to stored procedures can be executed from APL. A number of functions for memory caching and indexing are available, as well as prepared statements and bulk-SQL features for high performance table lookups, with minimal overhead
External References	Property based definition of APL values based on a certain key. This functionality enables the APL configuration to make dynamic runtime decisions based on parameters in an externally stored property file.
Events	Usage Engine provides various ways of dispatching information to different parts of the system or externally: Events can be sent to the System Log classified as ‘Error’, ‘Warning’, or ‘Information’ Events can also be sent directly to the Event Manager for distribution to any other target (SNMP trap, log file, database table, etc.) Debug information can be sent to a log file, or directly to the Workflow Monitor
Alarms	It is possible to configure and trigger potential Alarm conditions from within the workflow configuration as well as from the alarm detection GUI. These conditions provide a very flexible mechanism to visualize

Workflow Bridge Description

A Workflow Bridge agent acts as a bridge for communication between realtime workflows, or between batch and realtime workflows, within the same Usage Engine system. There are several benefits of using Workflow Bridge:

- Utilizing realtime-processing capabilities while keeping transaction safety when using the bridge between bridge and realtime workflow.
- Scaling processing of high volume streaming data by distributing processing load between execution contexts when bridging between an upstream data collection workflow and one or more downstream Realtime workflow(s). When sending data to multiple workflows the UDRs can be sent in a load-balancing scenario, where specific data can be distributed to a specific workflow. Data can also be broadcasted to all downstream workflows.



Workflow Bridge collection and forwarding agents

Communication

The Forwarding and Collection Bridge Agents communicate by using a dedicated set of UDRs. Communication is either done in-memory when workflows are executing within the same execution context or over TCP/IP, when workflows are executing on different execution contexts. This provides for efficient transfer of data.

To maintain transactional integrity across multiple workflows, the workflow state changes will be communicated over the workflow bridge. This enables downstream workflows to take action when upstream workflow state changes, and upstream workflows can maintain transaction integrity if downstream workflows fail to execute. For every transaction a session context with arbitrary data can be kept in the Realtime Collection agent. This can be used as a session object during the transaction.

In order to optimize performance, it is possible to collect and send data in a bulk from the Forwarding agent. When the Workflow Bridge Realtime Collection agent has received the data bulk, it is unpacked and forwarded as separate UDRs by the agent. The bulk is created by the Workflow Bridge Forwarding agent after a configured number of UDRs has been reached, or after a configured timeout. This is specified in the Workflow Bridge Profile.

Aggregation, Consolidation and Correlation

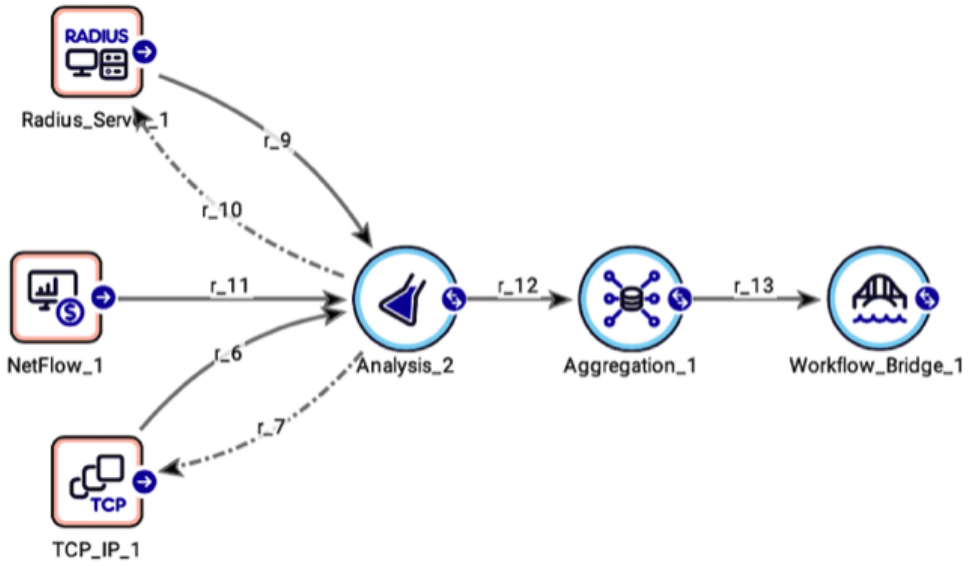
The Aggregation Agent can correlate, consolidate and aggregate records of different data types within the same processing stream and node. It is available for both realtime and batch workflows. Processing is in-memory; transaction safety is optional and achieved by storing sessions in database or file. Several database technologies are available. Business logic is highly configurable and deployed in the Aggregation Profile and the Aggregation Agent. The Aggregation Profile defines the objects and their relation:

- Which UDR types to accept for correlation/consolidation/aggregation.
- Rules for matching.
- Data structure of the session.

The Aggregation Agent defines all rules for all stages of processing, using the Analysis Programming Language (APL).

- Rules for creation and update of (correlation/consolidation/aggregation) sessions.
- Rules for forwarding of sessions, both valid and invalid sessions.
- Rules for timed out sessions.

In the example below, a realtime workflow collecting different types of data from three sources is illustrated. The incoming records are matched on specific fields.



Workflow with Aggregation agent

Sessions can be viewed and deleted manually using the Aggregation Session Inspection GUI.

Flush functionality

In some maintenance situations – like system upgrades or changes to business logic – it is desirable to flush existing aggregation /consolidation/correlation sessions before applying changes. For this purpose, specific “flush” code can be invoked:

- For batch workflows, the APL function (*aggregationHintFlushSessions*) executes the business logic in the timeout block for all sessions that have a timeout value.
- For realtime workflows, execute an agent command via the workflow monitor or the command line interface.

Detection of Duplicates

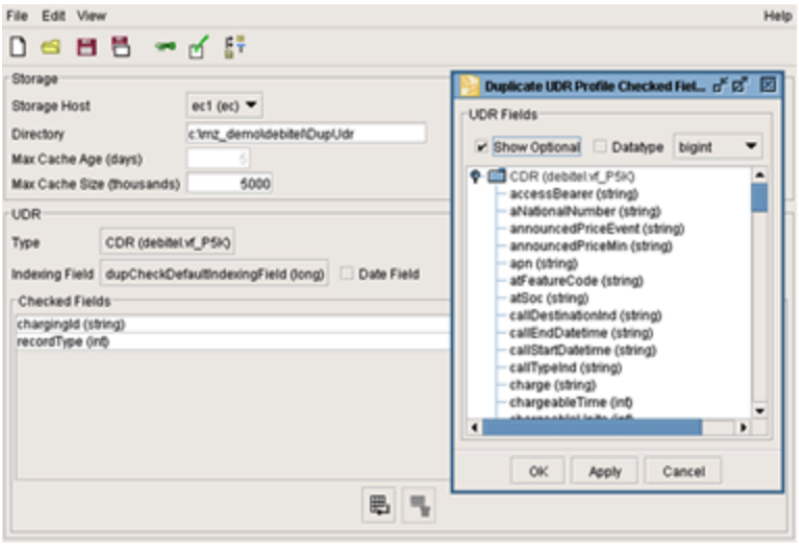
Usage Engine supports duplicate detection both on Batch and UDR level.

When a duplicate batch or UDR is detected, it can be routed to the Error Correction System for further analysis or discarded. An alarm can also be generated and mapped to any type of action, e.g. SMS, Email, Log file, and System Log.

The **Duplicate Batch Detection Agent** allows the user to configure which parameters should be used to check for duplication:

- Size of the batch.
- Checksum value calculated on the batch contents.
- Meta data related to the batch (such as a file name).

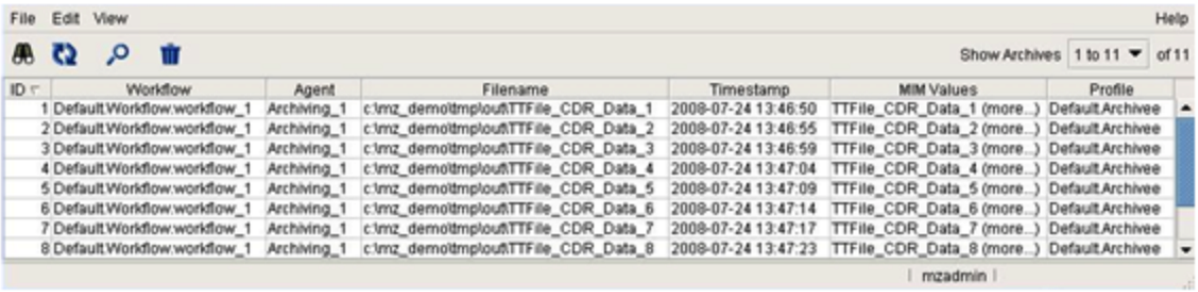
The **Duplicate UDR Detection Agent** also allows the user to configure which attributes from a UDR should be stored and used for duplication control.



Duplicate UDR detection

Archiving Description

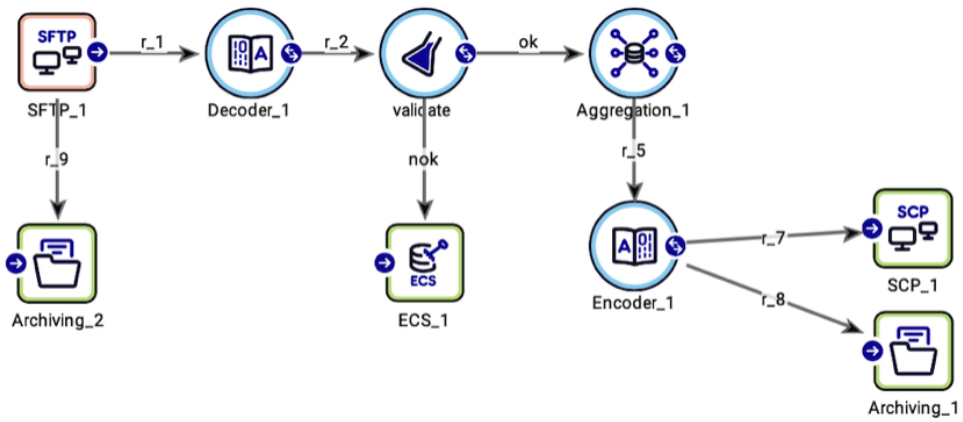
Usage Engine offers the possibility to archive data batches for a configurable period of time. The Archiving Agent is used to send all data batches it receives for archiving. Each data batch is saved as a file in a specified repository with a corresponding reference to the file in the database. Using the Archive Inspector, archived files can be browsed and deleted manually. Automatic deletion is handled via a predefined task that is executed regularly. The frequency of executing this clean up task, as well as the time period to keep the files, is configurable.



Archive Inspector

Depending on the selected profile, the Archive services are responsible for naming and storing each file and to purge outdated files at a regular basis. Utilizing the Directory Templates and base directories specified in the Archive Profile window, directory structures are dynamically built when files are stored. It is up to the administrator of the system to define which structure is suitable for each profile. Should the directory structure be changed on a daily basis, should it change based on collecting node name etc. The Archive services will automatically create all directories it needs below the base directories.

Each Archiving node can be configured to use a separate archive profile for its files, containing information about the target directory and removal of stored files. In the example below, a workflow is using Archiving Agents to store input and output.

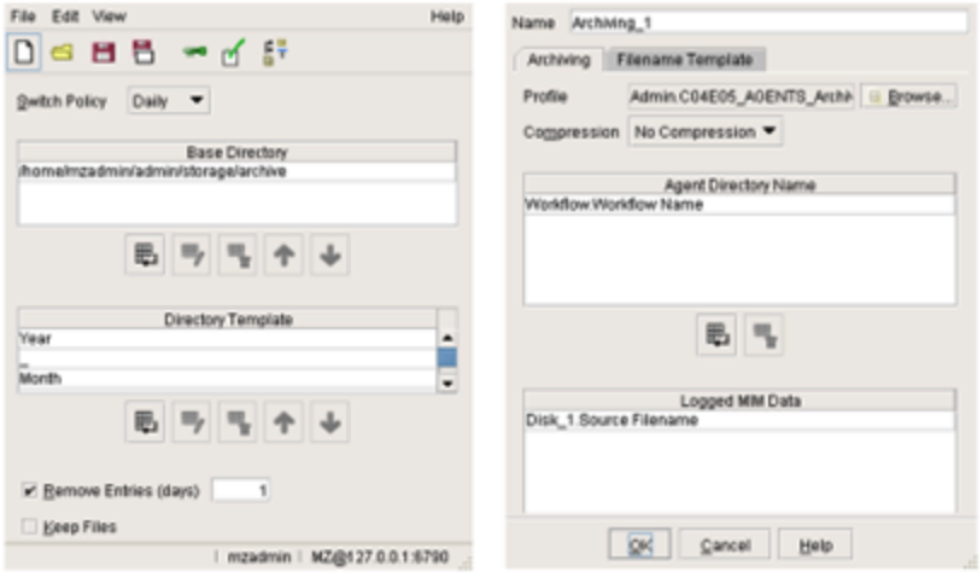


Workflow with Archiving Agents

The example below shows:

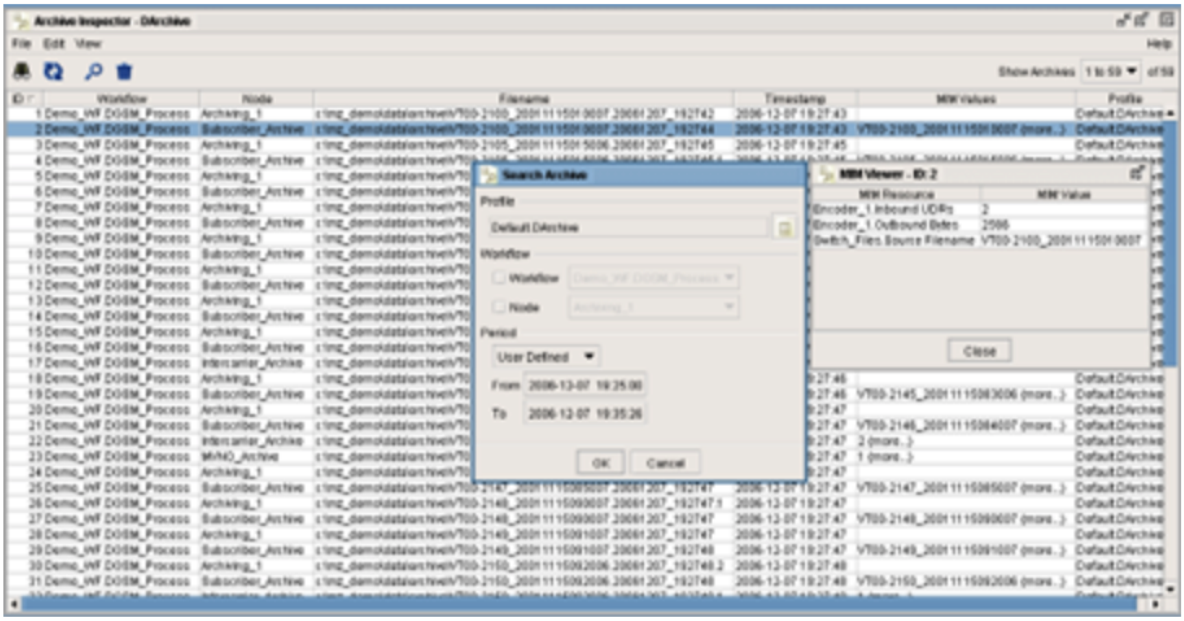
- An Archive profile configuring the base directory, dynamic directory template and the time to keep files in archive.

An Archiving agent configuration where an archiving profile is selected, dynamic MiM values are mapped to the Directory Template structure and selected MiM values are logged in the Archive Inspector.



Archiving profile

Archive Inspection with its search functionality can be used to search for files for a specified period, in order to re-distribute the files to downstream system(s). Figure below is an example of the Archive inspector.



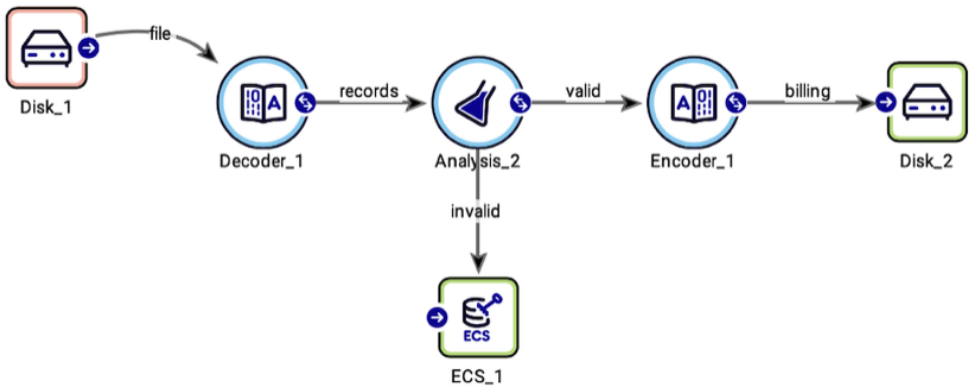
Archive Inspector

Error Correction System Description

The Error Correction System (ECS) is used for storage of UDRs and files that fail validation. Data in ECS can be tagged to enable identification, sorting and reprocessing. The ECS Forwarding Agent is used to send data to ECS, and the ECS Collection Agent is used for reprocessing stored data in a workflow.

There are two predefined error cases – Error Codes – for UDRs that do not match any of the configured session matching criteria for Aggregation, and for duplicate UDRs. A user can also define own Error Codes.

Below a workflow sending data to ECS is shown.

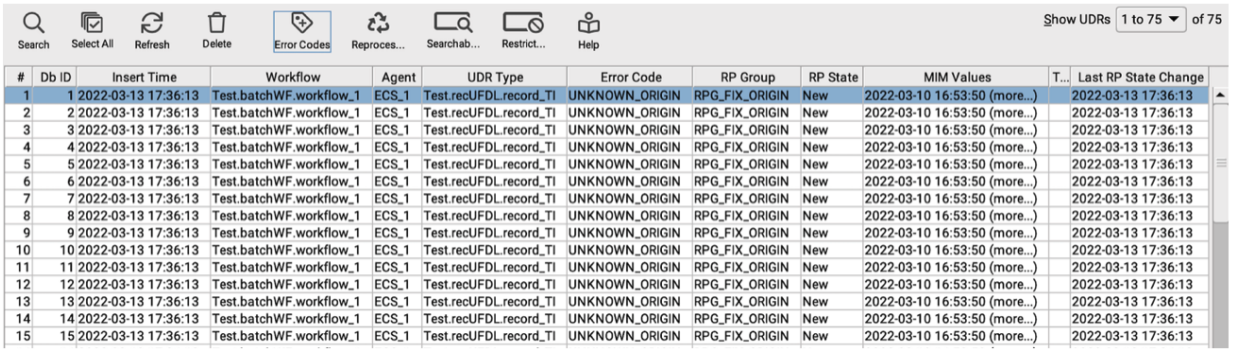


Workflow with ECS forwarding agent

ECS Inspector

The ECS Inspector is used to inspect and maintain UDRs and batches located in ECS. Examples of allowed operations:

- Define Error Codes and Reprocessing Groups. In order to be possible to collect for reprocessing, data in ECS must be tagged with a Reprocessing Group.
- Map Error Codes to Reprocessing Groups for automatic tagging during workflow processing.
- Set Reprocessing Group manually.
- Set Reprocessing State Manually.
- Change data.
- Define which data that cannot be changed.
- Define filters for easy search and view of data sets.
- Delete data.

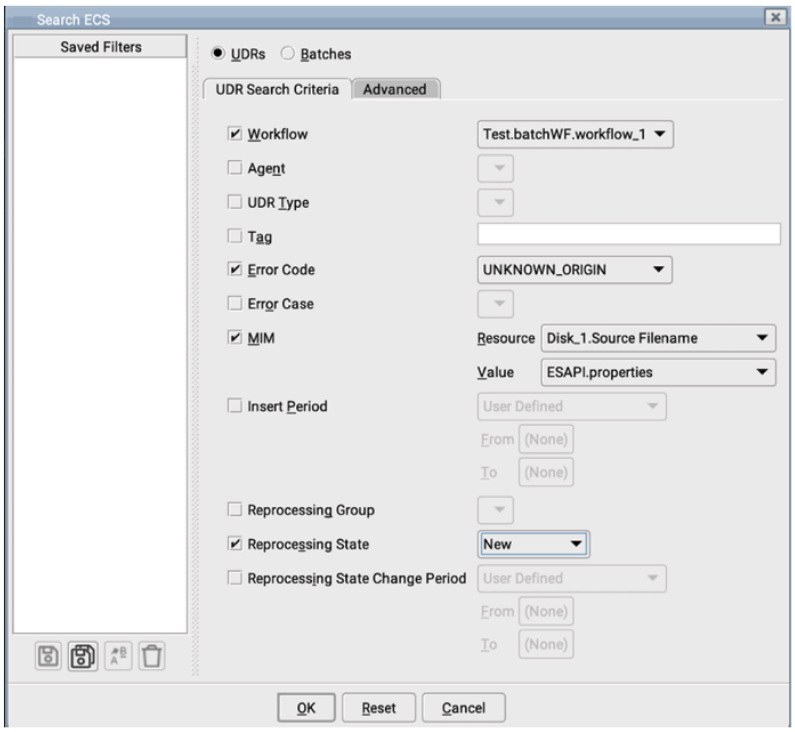


#	Db ID	Insert Time	Workflow	Agent	UDR Type	Error Code	RP Group	RP State	MIM Values	T...	Last RP State Change
1	1	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
2	2	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
3	3	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
4	4	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
5	5	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
6	6	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
7	7	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
8	8	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
9	9	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
10	10	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
11	11	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
12	12	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
13	13	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
14	14	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13
15	15	2022-03-13 17:36:13	Test.batchWF.workflow_1	ECS_1	Test.recUFDL.record_TI	UNKNOWN_ORIGIN	RPG_FIX_ORIGIN	New	2022-03-10 16:53:50 (more...)		2022-03-13 17:36:13

ECS Inspector

Error Search and Examination

The Search ECS UI has a rich set of search criteria. The selections can be saved as a Filter for future use.



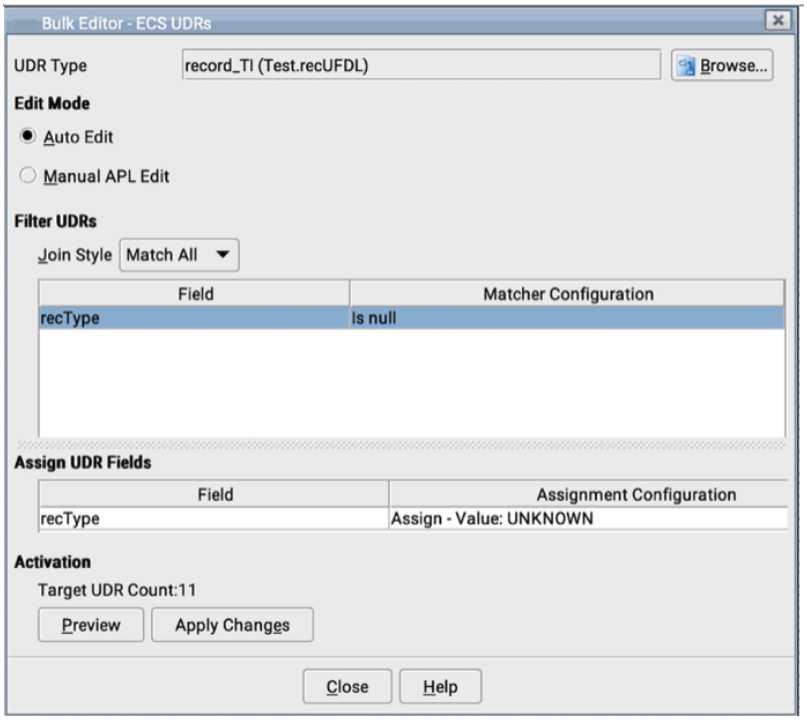
ECS Search criteria

The result of the search is displayed in the ECS Inspector window, where clicking on a row – which represents one entry / UDR – displays the full UDR content.

Manual Error Correction

To manually change a UDR, display the UDR by clicking the row in the ECS Inspector. It is also possible to select several UDRs and make a “bulk” change. There are two ways of making bulk changes:

- The Matcher Configuration of one or more fields to find the records that should be edited and then the Assignment Configuration to set the values in one or more fields as shown in the figure. This option will show the results on each record processed by the edit – the change or the indication of no change.
- Manual APL script that is written into a pop-up window. This is typically used when the predefined operations are not enough to cover the edit.



ECS bulk edit

Any changes to a UDR that is made using the UDR File Editor is logged in the System Log as a user event and includes the user id, date and time of change as well as the ECS ID for the modified UDR.

Error Classification and Grouping

Any UDR can be associated with error codes – and optionally any additional message/information – using APL. For instance, if the UDR is validated for user Id, missing or invalid Id can result in Error Codes “INVALID_ID” or “MISSING_ID”. In case the Id is invalid, you may want to add the value of the current id as the optional message/information, which will make it easy to view in the ECS Inspector.

Error Codes can be automatically assigned to re-processing groups to facilitate corrective actions such as collection from ECS to a correction workflow.

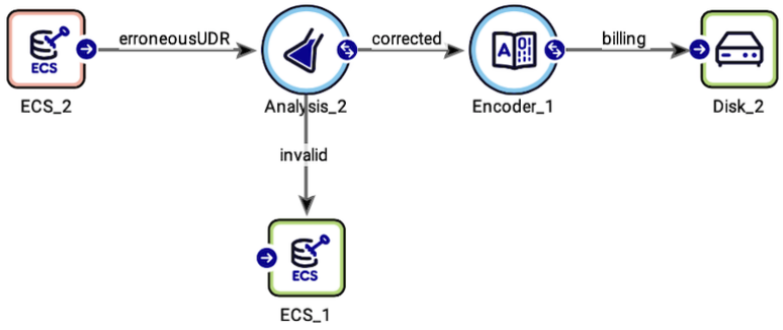
ECS Search and Update Tuning

The ECS Search and Update functionality can be tuned:

- Maximum number of ECS entries that are displayed in GUI simultaneously (default value 500).
- Maximum number of ECS entries that can be updated at one time (default value 15,000,000) with for example Bulk Edit.

Reprocessing of Erroneous UDRs

Data in ECS can be collected for reprocessing using the ECS Collection Agent. There are two prerequisites for collection of data from ECS; it must belong to a Reprocessing Group, and the state must be “New”. Note that collected data will change state to “Reprocessed” but will not be deleted immediately. The deletion is performed by the automatic Task “ECS Maintenance” (the time interval is configurable). Thus, if a user wants to collect data again, the state has to be changed back to “New” from the ECS Inspector.



Workflow with ECS collection and forwarding agents

The example above shows a reprocessing workflow that collects from ECS, applies correction logic and then re-introduces corrected data to the appropriate streams or sends it to ECS again in case the UDRs cannot be corrected.

Data Veracity Description

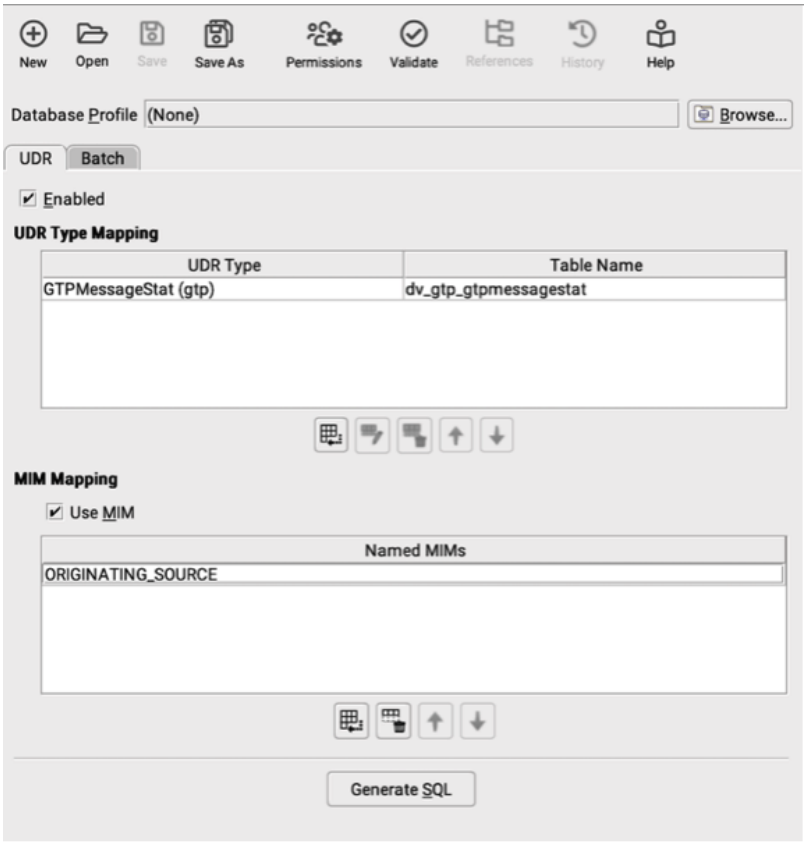
Data Veracity provides a repository of erroneous records and allows users to search, view, modify and reprocess the data. Data Veracity is as an alternative to the Error Correction System (ECS), complementing the existing functionalities, the main difference being:

- Support of larger volumes of data and has a higher performance capability when it comes to searching, filtering, and managing the data compared to the ECS.
- Support of external databases and configurable data models to enable high performance.
- Use of the Task Agent for correction of the erroneous data (as an alternative to collecting it using the Data Veracity Collection Agent into workflows).

Data Veracity has a separate web UI to support data exploration, classification, and modification of records; and three separate agents (Forwarding, Collection, and Task).

Data Veracity Profile Description

The Data Veracity Profile contains settings for selecting the database profile, mapping of UDR types to be used, mapping information for MIM and generation of the SQL script for the generation of tables in the external database.

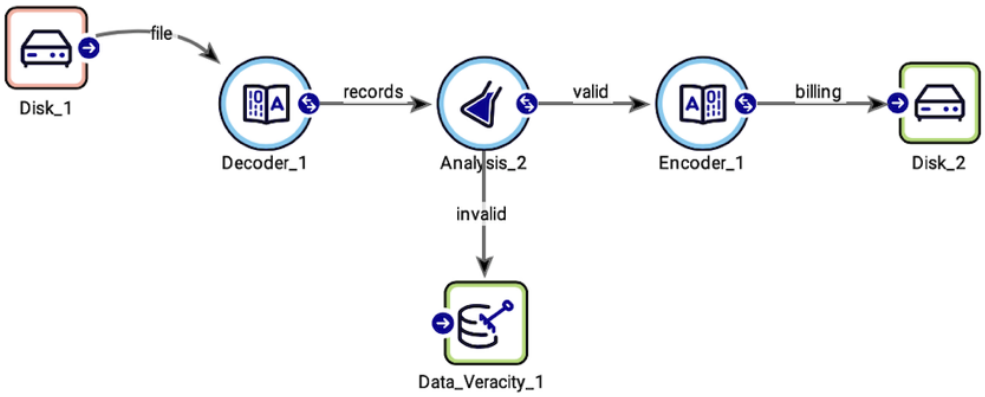


Data Veracity profile

Data Veracity has support for several databases.

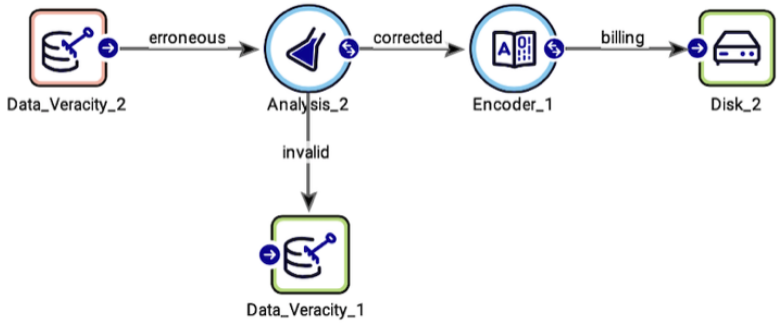
Data Veracity Agents

The Data Veracity forwarding agent connects to the external database and inserts the erroneous UDRs and any related metadata into the specific table. A standard Data Veracity Forwarding workflow will typically involve a collection agent, an analysis or aggregation agent to perform the UDR filtering, followed by the Data Veracity forwarding agent for storage of the erroneous UDRs.



Workflow with Data Veracity forwarding agent

The Data Veracity collection agent collects erroneous UDRs from the database using the search filters or error codes defined in the Data Veracity web UI. The state of the erroneous data will be changed from NEW to UPDATED after collection. Deletion of data is done at regular – and configurable – time intervals via a predefined cleanup task. The same data can be reprocessed multiple times.

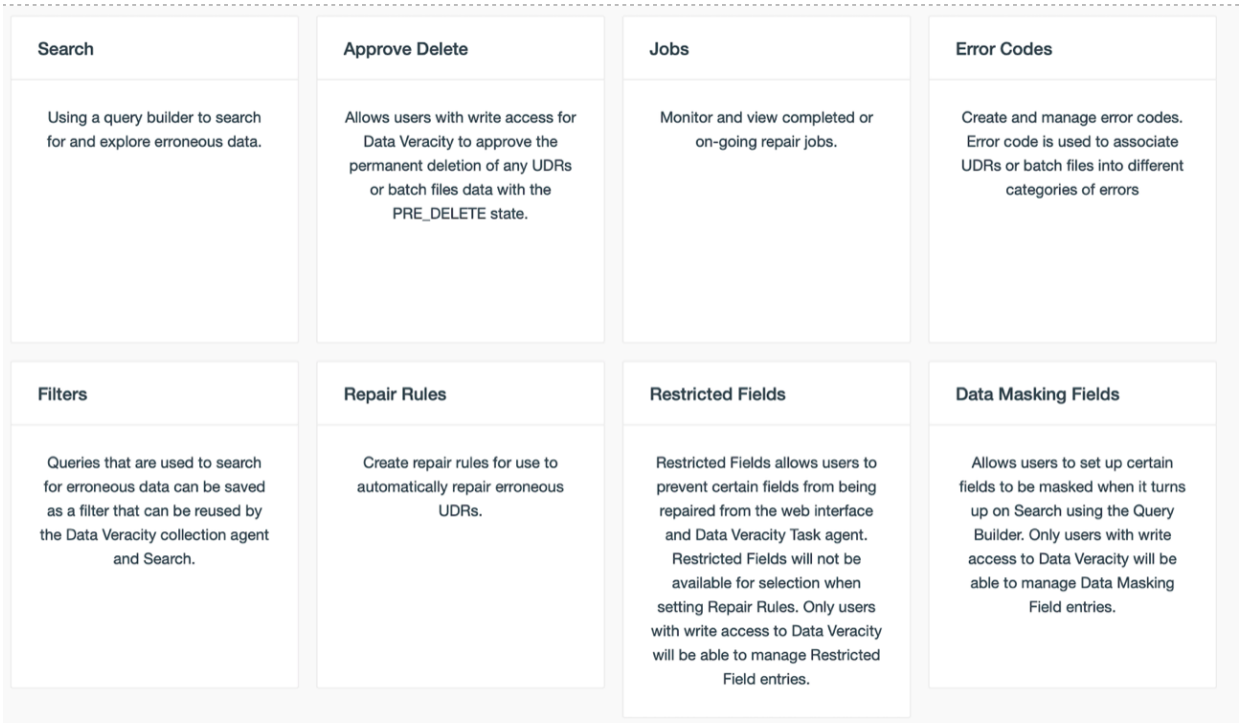


Workflow with Data Veracity collection and forwarding agents

As an alternative to a reprocessing workflow containing a Data Veracity Collection Agent, the Data Veracity Task agent can be used. It configures and schedules automated repair tasks for UDRs located in Data Veracity tables.

Data Veracity Operator UI

TheData Veracity web interfaceallows the user to inspect and maintain erroneous UDRs and batch files located in Data Veracity. Operators can view and remove Error Codes, Filters, Repair Rules as well as Approve or Reject UDRs that have been marked for deletion and edit their content.



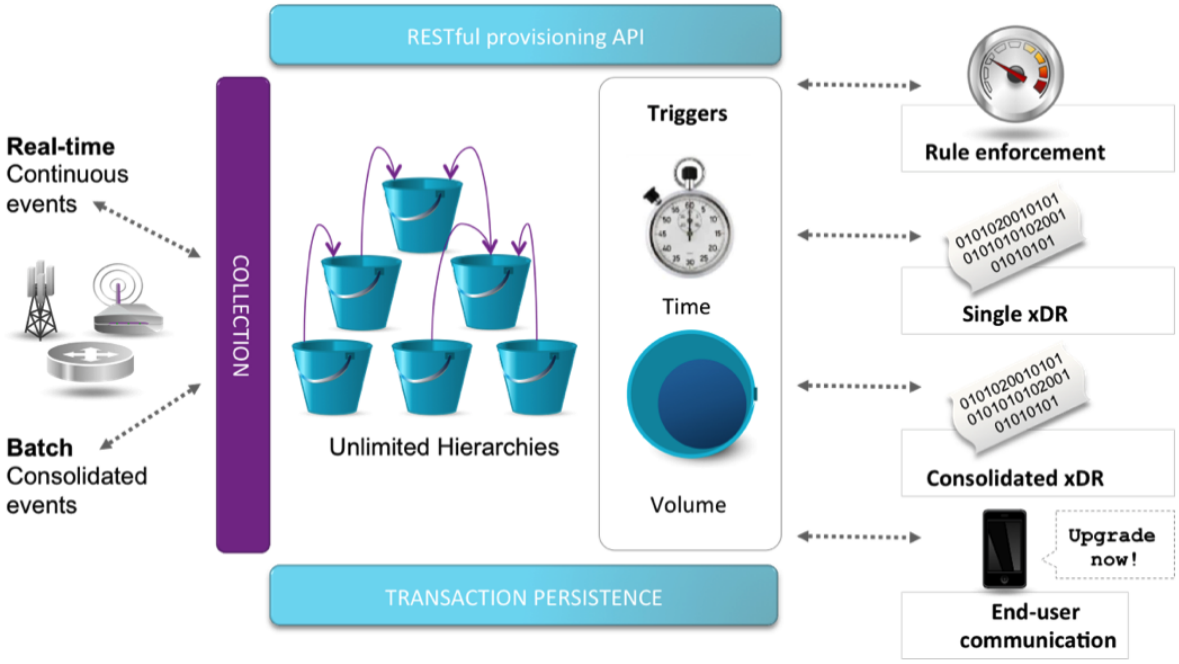
Data Veracity operator UI

Policy and Charging Control

Usage Engine includes PCC; a framework of functions that enable solutions for Policy Control, Usage Management and Routing Control. Workflow configurations are available that provide a baseline for rapid implementation of these solutions.

Through PCC functions Usage Engine supports a configurable way to:

- **Collect** usage data from offline- and online usage data sources such as file based (batch) UDRs or online usage data via our collection interface agents (e.g. Diameter etc.)
- **Count** the data based on based on volume (Gb, Kb, etc.), time (duration) and/or number of events and store the counters in unlimited hierarchies of usage counter containers called buckets.
- **Notify** external systems or end consumers about reached thresholds of the usage levels for the subscription.
- **Control** usage data consumption by triggering and sending configurable enforcements to external enforcement system such as a PCEF.
- **Route** Diameter messages or any type of realtime data to different destinations based on message attributes such as user identifier, source, destination, protocol, application etc.



Policy and Charging Control

Data Model

PCC consists of a configurable, persistent and pre-defined data model containing definition and relationships between several entities presented in subsequent chapters.

Buckets

Buckets are central entities in usage counting that encapsulates time- and volume-based usage counters for a subscriber that can be used for different purposes such as charging, policy, analytics, etc. Buckets are populated with usage data collected from continuous streamed events and/or consolidated file-based batches of events.

Bucket Data Holder groups for one subscriber or a group of subscriber (such as in a family or a company) all its counters with current values, effective dates and expiration dates, passed notification and sessions.

Each bucket references one product and has a start time and stop time that determines when the bucket should be activated and expired.

Bucket Content

Counter can be defined as different usage types:

- Volume (Gb, Kb, etc).
- Time (duration).
- Events (#SMS, #MMS, etc).

Notifications are triggered based on capacity threshold, to notify external systems or subscribers of status changes in their subscriptions.

- AoC, Top-Up request, Denial of Service etc.

Enforcement is an entity used by “Products” and “Rules” that contins thresholds when enforcements should be triggered and configured rules be applied.

Bucket characteristics defines:

- Validity, Capacity, Start time, Duration, Re-occurrence, Priority.

Products

Products controls bucket’s behavior by encapsulating configured capacity values of the counters, as well as information about the enforcements and notifications that should be triggered at specific capacity levels, count reset interval, etc. For example, a product can be

defined as a monthly counter with a capacity of 200 events; another product may be a weekly counter with a capacity of 5Gb; another product can be a one off counter, valid for 24h, and a maximum of 100 SMS; another example would be a monthly reset counter with a capacity of 10Gb that is only valid during peak hours.

Rules

With Rules you can create and apply different rules regarding Quality of Service and usage for different subscribers based on subscription type, location, current usage, etc. A key, composed of a set of attributes, identifies each rule.

Batches

Batches consist of set of APL functions that ensures transaction-safe batch processing in Usage Management solutions.

Routing

With Routing you can create and apply routing rules for Diameter messages or other types of realtime data. A key, composed of a set of attributes, identifies the routing functions that should be applied.

Pre-configured routing functions include:

- Routing based on session attributes
- Routing based on subscriber attributes
- Weight-based Routing/Load-balancing
- Workflow Bridge Routing/Forwarding

Additional routing functions for transformation or other types processing can be added through configuration.

Data Persistency

PCC functions are designed for high availability, with high throughput and low latency. The functions require a realtime data repository and several alternatives are available.

PCC Workflow Configurations

In order to ease the deployment, we have created a number of pre-packaged workflow configurations that solves some of the most common use cases for PCC based solutions.

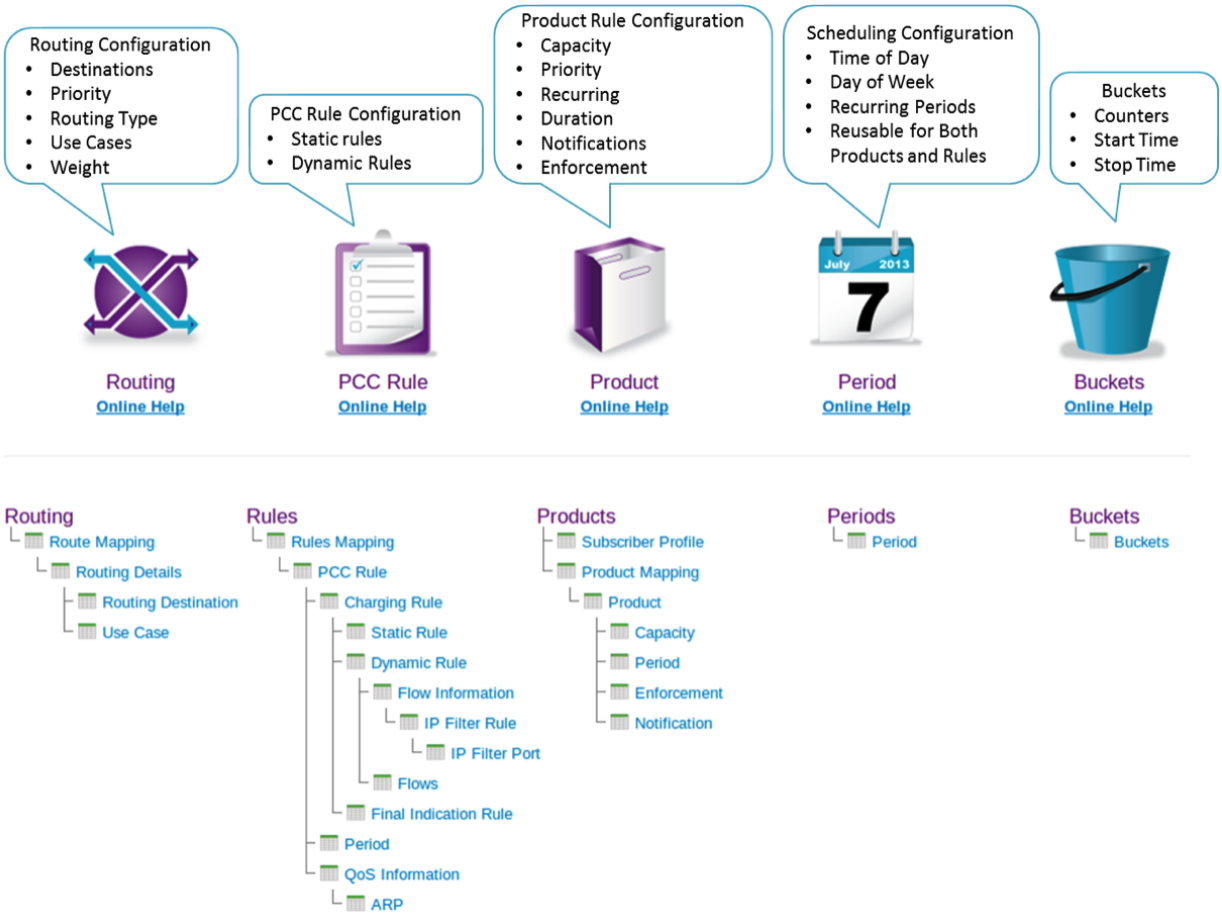
Examples of use cases for policy control:

- Service Differentiation Policy
- Coupon Policy
- Family Policy
- UE Policy
- Location Policy
- AF Initiated Policy

The pre-packaged workflows need to be adapted to meet specific customer requirements but typically reduce the cost and effort of the implementation project with as much as 75%.

User Interface

The configuration of the counters, products, thresholds, triggers & notifications is provisioned through the RESTful interface or a web user interface. Web User interface targets non-technical users for easy provisioning and view of usage data in realtime while RESTful is useful for more technical users or some external provisioning system.



Overview of PCC functions

Appendix A – Interfaces

Usage Engine provides connection interfaces towards a variety of external systems.

Interfaces are grouped into two categories:

- **Offline interfaces**
  Collection and forwarding interfaces specifically used in a file-based environment.
- **Online Interfaces**
  Client and server functionality for workflows that communicate over online enabled interfaces.

Please refer to the Product Catalogue for a list of available interfaces.