Table of Contents

19 Under construction page

III RENware support

I. System overview

ALPHAREN CORE-Integrator (ARINT) System

1 ALPHAREN CORE-Integrator

1.1 Product information

- p/n code: `0000-6151`
- product short name: `arint`
- product version: `0.1`
- product page: http://arint.renware.eu (http://arint.renware.eu)
- initial start: 2021

This product (as a "whole") is manufactured, registered and licensed by *RENware Software Systems* (http://www.renware.eu) which is the copyright holder. On the other hand product components / spare parts are under producers copyright (here can be found detail about components).

1.2 Documentation

Product documentation is divided in:

- **Overview** contains the information to understand your **ARINT** system and start of using it
- **User help** which represents a set of procedures for "day by day" operations
- **System administration** which contains in essence necessary information to install, configure and maintain the system

To access documentation just follow the navigation entries.

1.3 Demo system

The system siste contains also a link to a demo system where you can see & try the system "at work" and try its capabilities.

> **Demo system entered data**
>
> Pay attention that this system is just a demo and no saved data is guaranteed to be preserved even your information is stored under your "user context". Keep in mind that's just a demonstration system and ***do not store sensitive data***.

1.4 Support and assistance

The product site give you information about how to access support channel and how to buy this product. Support channel is ofered from producer site but depending on your country this will redirect you to most appropriate local dealear.

ALPHAREN CORE-Integrator (ARINT) System

2 Core-Integrator System Overview

Table of contents:

2.1 What is ARINT Core

ALPHAREN Core Integrator (aka `ARINT` or `arint`) system is a framework product for automation, integration and interoperability between *distributed systems* or *data sources*, basically aimed to build *API oriented*, *middleware*, *frontend* and *backend* applications.

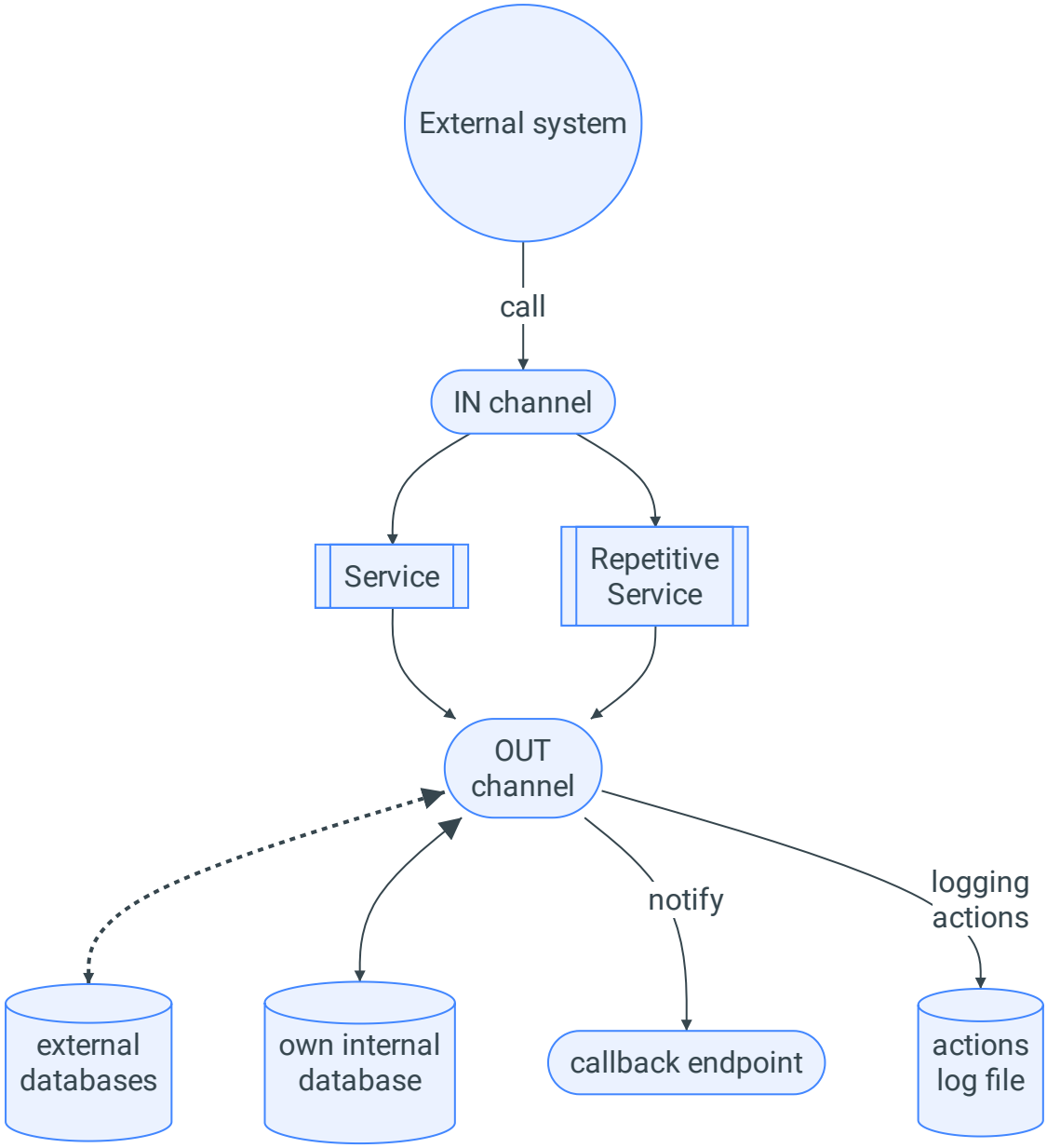
Practically it allows to create small-footprint and focused *business oriented microservices* or to transform "monolith" applications to micro-applications that will act *as a single one application* but with a high degree of *maintainability*.

Product is available as *distinct software* or as *ready to run appliance* (including also some built-in components such as an internal database for business operations).

ARINT as Service bus

ARINT acts as a high level *Service BUS* (ie, ESB or ESOA) to connect different micro-services and to make them to work ***as one***. As example it is already used by all *RENware Software Systems* products. Of course it can be used for **CUSTOMER SYSTEMS and SERVICES** too.

ARINT generic process flow is:



Remarks to diagram

- the `IN channel` establish the way to address the ARINT system (how to call it)
- the `IN channel` establish the security rules in calling the ARINT system (authemtication)
- a `Repetitive Service` is normally called once (ie, to start it) and it begins to repeat operations (in background) at *defined time intervals* and for a *defined period* (or indefinitely)

2.2 Availability and system "presence"

- **ANYWHERE**. can work even the systems that must be integrated are in different non routable LANs (address systems at `http` protocol level)
- **ANYHOW**. is agnostic to format, composition, structure, encoding of information required / provided by systems that must be integrated

- **ANYTIME**. can work as a distributed high scalable cluster of "**ALPHA-REN Integrator Machines**"
- **SECURED**. can work with any public standard (ie, defined at least as `RFC`) of Internet security

> Each ARINT system can run: * on premises or in cloud, * deployed as classic software or Docker application container, Kubernetes node / container or * as any general containerization "standard" method

2.3 Features

For features list go here

2.4 Typical use cases

ALPHAREN CORE-Integrator is used for enterprise, business integrations, data science, IoT and other scenarios that require integrations of multiple systems.

Real-world, production ***ALPHAREN CORE-Integrator*** environments include:

- A platform for processing payments from consumer devices
- A system for a telecom operators integrating CRM, ERP, Billing and other systems as well as applications of the operator's external partners
- A data science system for processing of information related to securities transactions (FIX)
- A platform for public administration systems, helping achieve healthcare data interoperability through the integration of independent data sources, databases and health information exchanges (HIE)
- A global IoT platform integrating medical devices
- A platform to process events produced by early warning systems, (ex SAP EWS)
- Backend e-commerce systems managing multiple suppliers, marketplaces and process flows B2B platforms to accept and process multi-channel orders in cooperation with backend ERP and CRM systems
- Platforms integrating real-estate applications, collecting data from independent data sources to present unified APIs to internal and external applications
- A system for the management of hardware resources of an enterprise cloud provider
- Online auction sites
- E-learning platforms
- Ad-hoc data API for databases for example to protect them to direct access or to hide particular implementation details (especially in legacy old databases) allowing for a smooth and transparent transition to new redesigned implementations

ALPHAREN CORE-Integrator (ARINT) System

3 Product features

Table of contents:

3.1 General features

- Open API full compliant
- Integrations, Microservices, SOA and ESB in Python
- high availability load-balancer
- hot-deployment and hot-reconfiguration - deploy with no downtime
- Browser-based GUI, CLI
- API - easy to use and customize

For an overview of rhe product please see.

3.2 Standards compliance

The following list presents some public (most usual) standards, protocols, data stores, formats, and so on, *that ARINT system can use and is compliant with*.

The actual list is larger and practically *any kind of specific interface* can be assured by services as long as is written the corresponding code.

Standards update

Also verify your system version and update it as new standards can be included in official package releases.

> The order of items is not relevant (meaning that it does not pursue a specific objective). Also classification made is not *an exact one* as some items can be categorized in more places. If someone know exactly what standard is looking for, a traversal of the entire list is best option.

- Protocols:
  - REST
  - SOAP
  - FTP
  - SFTP
  - LDAP
  - Active Directory
  - WebSockets
- Industry standards:
  - HL7 (healthcare - data exchange)
  - RBAC (IT - Role Based Access Control)
  - Swift (banking - Society for Worldwide Interbank Financial Telecommunication)
- Business systems:
  - SAP
  - Odoo
- Mail and messaging protocol and systems:
  - SMTP
  - IMAP
  - Telegram
  - JMS
  - Twilio
  - Slack
- Data languages and formats;
  - OpenAPI
  - SQL
- Databases and broker systems:
  - MongoDB
  - Redis
  - Memcached
  - Cassandra
  - Kafka

- Search systems:
  - ElasticSearch
  - Solr (Apache)
- File oriented stores and depots:
  - Amazon S3
- Queue based communication systems:
  - AMQP
  - IBM MQ
  - ZeroMQ
  - JMS (Java Message Service)
- Security and protection:
  - Vault
  - all cryptography standard algorithms

3.3 Services level issues and features

In writing an ARINT services there are some frequent and repeating issues. An ARINT service address them by offering built-in solutions and ***let developer focus on business aspects***. Addressed issues are:

- How do I connect to ARINT own resources ? How do I send user entered credentials to ARINT?
- How do I connect to external systems or databases to store or extract data?
- How do I convert and / or map messages from one format to another?
- How do I automate my work that it is repeatable in most cases (for example across environments like `development`, `test`, `production`)?
- How can I focus on my job instead of thinking of trivial low-level details?
- How do I debug my code?
- How can I reuse the skills and knowledge that I obtained?
- How do I manage the complexity of having to integrate at least several systems and dozens or hundreds of APIs?

All these questions find answers especially in "Development documentation", Development Overview being a good start.

3.4 Frequent service type examples

Here are some examples of what kind of services can be accomplished by ARINT:

Most popular

Technologies most commonly used in API integrations: *REST | SOAP | Scheduler | Pub/sub | SFTP | WebSockets*

Databases and message queues

How to access commonly used data sources: *SQL | MongoDB | Redis | AMQP | IBM MQ | ElasticSearch*

Business apps

Integrating with 3rd party CRM and ERP software: *Microsoft 365 | Salesforce | Odoo*

Health care interoperability

Integrating health systems meaningfully: *HL7 FHIR | Banking Security Systems*

E-mail

How to send and receive emails: *IMAP | SMTP | Microsoft 365*

Shell commands

Turning shell commands into API services: *Shell commands | SSH | PowerShell*

File integrations

Integrating systems using continuous or batch file transfer: *File transfer | SFTP | FTP*

Requests, responses and data models

Convenient access to request and response data: *Requests | Responses | Data models | OpenAPI*

Cloud integrations

Integrating with popular cloud providers: *AWS S3 | Jira | Confluence | WordPress*

LDAP and OAuth

Integrating with external security providers: *LDAP and Active Directory | OAuth and REST*

These are only some kind of type of supplementary services that can be written in ARINT. More details about services will be found in Development - Service anatomy.

3.5 ARINT own objects and components

Besides the features offered as external systems integration, the ARINT has its internal objects and components (that will be detailed in other documents). These are:

- **System database** containing all ARINT meta-information about its objects
- **Business database** containing all business entities and objects with their effective data
- **Master data** containing all "cross instances" master data objects (more exactly master data objects that are agnostic to business entities and objects content)

A detailed description of *System Data & Objects*.

ALPHAREN CORE-Integrator (ARINT) System

4 System Landscape

Table of contents:

4.1 ARCLST the elementary component

The elementary, indivisible element, "heart" of the *ARINT* system is **ARCLST**. This is an *cluster node* which is exactly as the says, a cluster, having independence and ability to solve all problems that ARINT implementation is designed to address.

An *ARCLST* can be linked with other ARCLST nodes and can work in any mode, ie as *active-active* or *active-passive* at the administrator option to configure system.

ARCLST physical locations

An *ARCLST* must have all machines (servers) **located in the same LAN**. All machines inside should be able to communicate between them at IP level. Many implementations (and is a practice) isolate the *ARCLST* from the installation phase in a dedicated operating system container (ie, not application container) - `Ubuntu LXC` being a very good example of this kind of container.

4.1.1 ARCLST essential components

- `[1 to n]` **ARINT Framework** servers[1] (these will work ONLY in *active-active* mode)
- `[0 to 1]` **ARINT RDatabase** relational database server(s)
- `[1]` **ARINT KVDatabase** NoSQL database server(s)
- `[1]` **ARINT CORE** administration and management service and its user interface (once installed and activated becomes physically part of *ARINT Framework*)
- `[0 to n]` **ARINT products** various other products, applications, etc (once installed and activated becomes physically part of *ARINT Framework*)

Notes regarding used terms

- notation `[n...]` specify a quantity range of allowed resources, instances and follow the usual practices in range specifications
- if not otherwise specified by *server* term is understood a *logical server*, meaning that can be a physical or any form of virtual one, but a dedicated machine perceived as having its own operating system isolation level (for example its own dedicated `root` for *Xenix* kind of machines, an application container, etc)

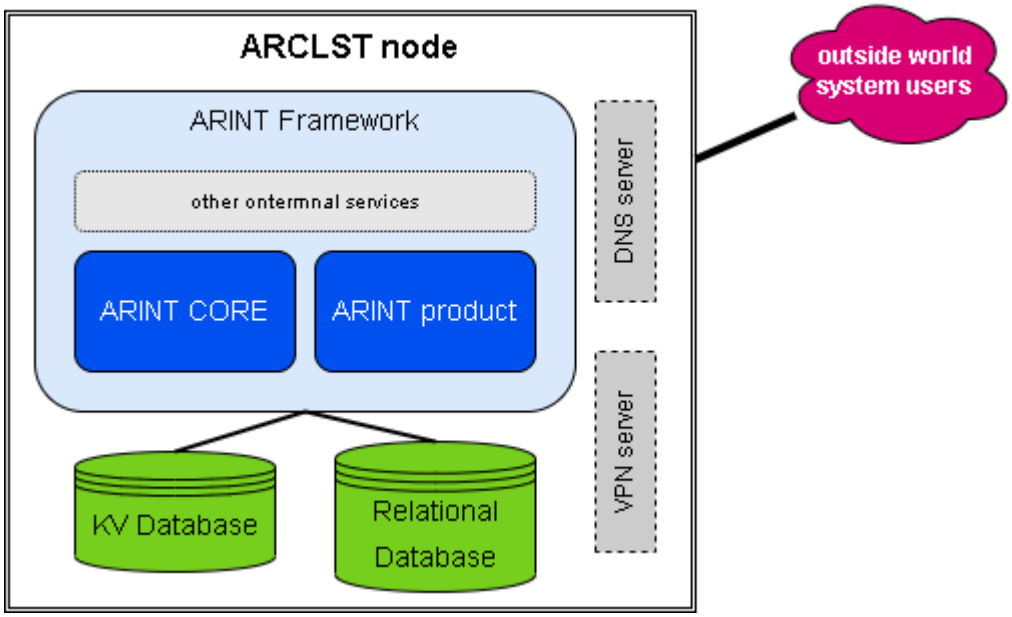
4.1.2 ARCLST auxiliary components

Auxiliary[2] components can help in various implementation projects depending exclusively of client (customer) options and its infrastructure existing assets.

- `[0 to 1]` *internal DNS (or dynamic DNS)* which is a standard DNS inside the *ARCLST*. No ARINT component will use it but will allow for a "nice" outside LAN addressing of *ARCLST* components when customer explicitly wants and needs that, otherwise ARINT has a proxy and balancer included which assure its addressing as sysyem and for business purposes
- `[0 to 1]` *internal VPN server* that will allow to "enter" in *ARCLST* LAN space. That's not needed for normal purposes but only if customer need for its own infrastructure operations and easy management

Normally these components are left at router glance as being normal included functionalities in enterprise / business router.

4.1.3 ARCLST schematic architecture

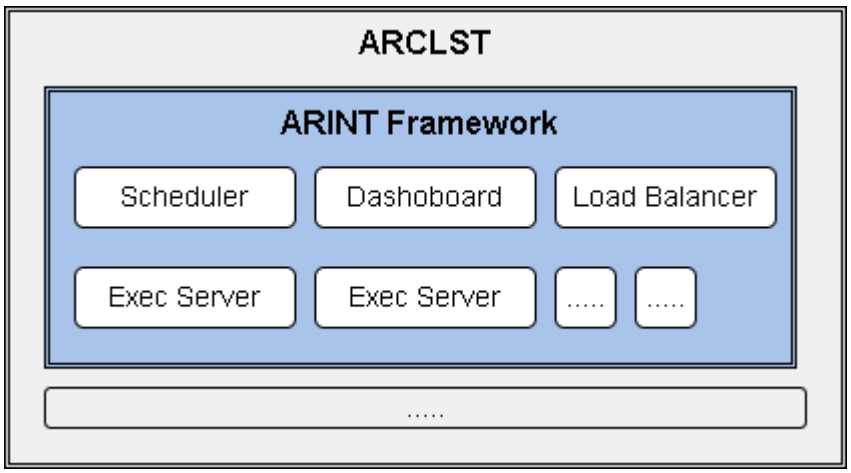


4.2 ARINT Framework

ARINT Framework contains the following components:

- `[1]` **Load Balancer** - this is the load balancer and active-active operating assurance component. Default is build using `HAProxy`
- `[1]` **Scheduler** - this assure all scheduling (one time or repetitive) of services execution
- `[1]` **Dashboard** - this is an own administration dashboard available (access controlled and exposed) through *ARINT CORE* (normally as `iframe`)
- `[1 to n]` **Exec Server** - this are servers (physical or virtual) where services are effectively executed

4.2.1 ARINT Framework schematic architecture



4.3 ARINT CORE

-#TODO wip...

4.4 ARINT RDatabase

-#TODO wip...

4.5 ARINT KVDatabase

-#TODO wip...

4.6 ARINT products

-#TODO wip...

4.7 Auxiliary components

4.7.1 DNS server

-#TODO wip...

4.7.2 VPN server

-#TODO wip...

... IN REVIEW ... PLEASE BE PATIENT - coming soon -(#TODO - continue review ...)

4.8 Notes and remarks

1. ARINT Framework machines was in previous versions called ARSRV ↩
2. These components are considered "auxiliary" because in most cases they are not needed anymore and in consequence are not part of installation process (to not induce useless overhead at system resources level) ↩

Redis- **ALPHAREN CORE-Integrator (ARINT) System**

5 Licensing Editions and Pricing

product version: `0.1`

Table of contents:

5.1 Introductory

This section is intended to briefly present the product components that are relevant for *available Editions* and their *Licensing model(s)*.

5.1.1 Components

The licensing relevant components are:

- the **ARINT framework** as being the core engine of ARINT operations execution. This component assure all routes listening and http request decoding, data formats and conversions, http response preparation, internal scheduler, and so on, keeping all of these in compliance with all enumerated standards, protocols, data formats, etc in document Product Features
- the **Databases** that come as built-in ARINT. This databases (data stores) keeps all system data required in all operations execution (see document System Data and Objects)
- all **ARINT CORE** that represents the:
  - *administration interface* for all ARINT CORE specific data, processes, operations, schedules, system logs analyzer (`ARINT AdminBoard`)
  - *services code* (programs) of all operations that ARINT is able to execute (found in all end user and administration manuals)
- **ARINT products** represents different "ready made products" which have a complete defined functionality. *These are completely eparated products from commercial point of view*, but NOT all editions can accommodate them

5.1.2 Services

Also a set of associated services is relevant. A set of **minimal services** is:

- *infrastructure level* services: product installation, operating system preparation, install different other requirements, users management, etc
- *ARINT basic configuration* representing the services that assure the minimal ARINT configuration in order to start and being able to respond to ezternal systems requests and being able to complete address its own databases (read, write, update, delete...)

There are also more other possible services. The *minimal services set* assure only a right installation and basic system functioning. Supplementary services can include:

- infrastructure tuning in customer specific environment
- system configuration to some basic customer specific information (like name, location, basic domain and ARINT server name, etc)
- more configuration services that are subject to a dedicated project (or project work package)

5.2 Editions

Feature	ESE	SE	SBE	EE
LMe[1]	pcs	station	role	n.usr
AF[2]	x	x	x	x
AC[3]	x	x	x	x

Feature	ESE	SE	SBE	EE
RDB[4]	MariaDB	PostgreSQL	MariaDB	PostgreSQL
KDB[5]	Redis	Redis	Redis	Redis
PRD[6]	ex disk	x	x	x
Services	B	B	B	B, 1

5.2.1 Editions

- **ESE** - Embedded Starter Edition
- **SE** - Standard Edition
- **SBE** - Small Business Edition
- **EE** - Enterprise Edition

5.2.2 RBD options

- (1) PostgreSQL
- (2) MariaDB

5.2.3 KDB options

- (1) - Redis
- (2) - MongoDB - available only as user option that should have a valid license or assume the trial installed one

5.2.4 Services options

- (B) - represent the minimal set of services as defined in Services section
- (1) - training for:
  - Administration (6 hours)
  - Development Introduction (2 hours)
  - Development IN channels (1 hour)

5.2.5 ARINT products options

- (ex disk) - only with external disks (or network mapped disks)

5.2.6 Metrics options

- *pcs* - at number of pieces
- *station* - at number of stations (physical or virtual computing units)
- *role* - at number of business roles that use system in making requests to
- *n.usr* - at number of nominal users that use system in making requests to

5.3 Pricing

Standard prices are available in EUR (or USD) equivalent on RENware Software Systems catalog section (http://www.renware.eu) . Prices are for each edition and for optional services (including training).

Our recommendation is to discuss for a "project level" agreement, to be able to set some objectives ref product implementation and in-production usage.

5.4 Notes and remarks

1. LMe = License Metric ↩
2. AF = ARINT Framework component ↩
3. AC = ARINT CORE component ↩
4. RDB = Relational Database (see also RDB options) ↩
5. KDB = NoSQL Database (see also KDB options) ↩
6. PRD = specify if can accommodate ARINT products (see also ARINT products options) ↩

ALPHAREN CORE-Integrator (ARINT) System

6 Services

Table of contents:

UPCOMING...

ALPHAREN CORE-Integrator (ARINT) System

7 Training Programmes

Table of contents:

UPCOMING...

II. Documentation

II.I End User procedures

ALPHAREN CORE-Integrator (ARINT) System

8 End user documentation catalog

Table of contents:

8.1 Basic work procedures

- wip... Navigation in system
- wip... Procedure A
- wip... Procedure B
- ...

8.2 Advanced work procedures

- wip... Calling a repetitive task

II.II System administration

ALPHAREN CORE-Integrator (ARINT) System

9 System administration documentation catalog

Table of contents:

9.1 Basic procedures

- Installation
- wip... Basic configuration
- wip... Maintain system users
- wip... Expose static sites
- wip... System backup & restore

9.2 Advanced procedures

- wip... Advanced configuration
- wip... Additional database systems installation
- System Data and Objects
- wip... Configure an ad-hoc data API
- wip... Configure a callback route (for example as return from a 3rd party electronic payment system)

ALPHAREN CORE-Integrator (ARINT) System

10 System Installation

Document control:

* last update date: 230605

* last updated by: petre iordanescu

Table of contents:

10.1 Install helpers

```
sudo apt-get install apt-transport-https curl
sudo apt-get install software-properties-common
sudo add-apt-repository universe
sudo apt-get install tzdata
```

10.1.1 Package key

```
curl -s https://zato.io/repo/zato-3.2-48849AAD40BCBB0E.pgp.txt | sudo apt-key add -
```

10.1.2 Add apt repository

```
sudo add-apt-repository \
   "deb [arch=amd64] https://zato.io/repo/stable/3.2/ubuntu $(lsb_release -cs) main"
```

10.1.3 Install zato

```
sudo apt-get install zato
```

10.1.4 Apply latest updates

```
sudo su - zato
cd /opt/zato/current && ./update.sh
```

10.1.5 Check & confirm

```
zato --version
```

10.2 Create a quick cluster environment

10.2.1 Create the cluster

This will create a new cluster ARCLST named **`arclst`** in directory `/opt/zato/env/arclst`.

```
sudo su - zato
mkdir -p ~/env/arclst
cd ~/env/arclst
zato quickstart create . sqlite localhost 6379
```

Response should looks like that:

```
[1/8] Certificate authority created
[2/8] ODB schema created
[3/8] ODB initial data created
[4/8] server1 created
[5/8] Load-balancer created
Superuser created successfully.
[6/8] Dashboard created
[7/8] Scheduler created
[8/8] Management scripts created
Quickstart cluster quickstart-904765 created
Dashboard user:[admin], password:[F7qCOiabas5ToQ7EWupLrHOn9iVHzyBv]
Visit https://zato.io/support for more information and support options
```

10.2.2 Change web console password

The username web administraton console is **admin**. To change the password in **admin**, do:

```
cd ~/env/arclst/web-admin/
zato update password . admin
```

10.3 Configuration & access

10.3.1 Machine general configuration

- Test machine ren-cluster, 192.168.1.190
- Admin console port 8183
- Credentials admin / admin
- public access http://90.84.237.32:8183 user admin pswd admin

10.3.2 Machine environment configuration

- User `sudo su - zato`
- Path `/opt/env/arclst`
- web admin console path `/opt/env/arclst/web-admin`
- ZATO Server `arclst`

10.3.3 System launch scripts

- `zato-qs-start.sh`
- `zato-qs-restart.sh`
- `zato-qs-stop.sh`

Server start in background mode, NOT as daemon.

10.4 Installation notes

None. Everything works as documented. ATTN open 8183 port

ALPHAREN CORE-Integrator (ARINT) System

11 System Data and Objects

product version: `0.1`

Table of contents:

11.1 Introductory

11.1.1 Naming rules

- Objects `code-name` is not case sensitive and is a good rule to use always uppercase. The system will convert the data objects `code-name` to uppercase, even from the save step.

- New "user or client or implementation" level business objects `code-name` should start with character `Z`. Any other ARINT objects do not use `Z` as first character in code-names, so this can make an *acceptable separation of those objects that are subject of change by system updates* and is a *guarantee that objects starting with `Z` in code-name will not be altered* by any system update.

11.1.2 Object categories and models

Any data object must be in one of these categories:

- **entity** - This is a relational data entity, more exactly a *table* to use the specific relational modeling concept. Such kind of objects are stored in relational system database.
- **json** - This is an object can contain any data model and type that respect `JSON` standard. This kind of object can cover almost all kind `NoSQL document type` variants. Such kind of object can be stored in any system database (relational or kv store) at your option. Default store place is kv system database.
- **crypto_key** - This kind of object contains keys used for cryptographic purposes. This kind of object is represented as a limited `json` type (strict one level, no arrays, only string type keys). A `crypto_key` object usually has one or two entries corresponding to one (in symmetrical cryptography) or two (in asymmetrical cryptography) keys, both as strings of `UTF-8` characters (bytes).
- **file** - This kind of object is designed to keep files or directories references. This references are stored can be stored in any system database (relational or kv store) and must respect all *Linux* conventions regarding file names (directories are files too represented as convention with a `/` character at end if is not obvious from context). So as a minimum strings of `UTF-8` characters (bytes) no more than 256 characters (bytes)

As definition models the `OO` definition is preferred and for relational objects the *SQL Alchemy (https://www.sqlalchemy.org/)* is preferred option. For `kv` data the Python normal `dictionaries` can be used.

11.1.3 Namespaces catalog

Here are listed the main *area of objects* defined in *System database*. These are the equivalent of a `namespace` (or `schema` in relational databases terminology) and they will become in a way[1] a `prefix` for all entities / objects contained in.

- **URRM** user roles and rights management
- **ODEF** business objects definition
- **MADA** global master data (cross any systems, applications, nodes, ie global for an **ARINT instance**)
- -#NOTE future reserved ...

11.2 Namespace URRM

11.2.1 URRM catalog

- -#NOTE future reserved ...

11.2.2 URRM_new_object_x

- -#NOTE future reserved ...

11.3 Namespace ODEF

11.3.1 ODEF catalog

- -#NOTE future reserved ...

11.3.2 ODEF_new_object_x

- -#NOTE future reserved ...

11.4 Namespace MADA

11.4.1 MADA catalog

- `OBTYP` object types and models. This contains values described of "Object types and models" section
- -#NOTE future reserved ...

11.4.2 MADA_new_object_x

- -#NOTE future reserved ...

11.5 Notes and remarks

---

1. (`PROPOSAL @ 230823 by piu`): The way that concrete entities / objects are prefixed depends of database, ORM instrument used, etc. To keep as agnostic as possible this prefix will be used as *object name prefix* followed by `_` character. **All `code-name` s for a prefix will be kept in 4 characters**. ↩

II.III Development manuals

ALPHAREN CORE-Integrator (ARINT) System

12 Development documentation catalog

Table of contents:

12.1 Developer manuals

12.2 Development information

ALPHAREN CORE-Integrator (ARINT) System

13 Development Overview

Table of contents:

13.1 Preliminaries

Development process over ARSRV implies basically the following components:

- **SRV** - service
- **CHN** - channel

Fundamentally and very high level, a service (SRV) use a channel (CHN) to communicate with external environment.

13.2 CHN - channel

A channel must be defined in **ARSRV** management interface before use. The channel can be:

- **IN channel** which establish and endpoint route through an ARINT service can be invoked (called)
- **OUT channel** which establish a "place whwere ARINT can write (send)" information

The CHN establish:

- an own name which uniquely identifies it
- the endpoint address
- the protocol used
- data formats in messages exchanged thru the channel
- auth and other security parameters

13.3 SRV - service

A service must be written in Python then deployed to **ARSRV** in order to be used.

A service has the following high level flow:

- defines a handler in order to be accessed by ARSRV
- obtain any required parameters in order to properly do its job
- connects to a channel to read required input
- make the necessary transformation over obtained data
- connects to a channel to write computed output
- log any process details for future references and errors debugging

13.4 File names

Development documents (except the current one) will be named as follows:

- `06.DEV` as prefix
- *optional* a code which specify (only if is case) at which subcomponent or pritocol, and so on
- name of the document

13.5 Services names

The producer reserve a name space for its services (as built in AR Integrator or as future updates) starting with characters **AR**.

The users are free to name how they wants their own developed services, but not start with AR characters. Respecting this rule will allow producer future updates to overwrite *client own developed services*.

This rule should apply as general validity for any components names, for example channel names.

Anyway the customer must be aware that names starting with `AR` characters are reserved and are subject of future changes without any notice or change log.

ALPHAREN CORE-Integrator (ARINT) System

14 Service anatomy

Product 0000-0156 0.0 document control:

- 210728 me new doc
- 230817 me last update

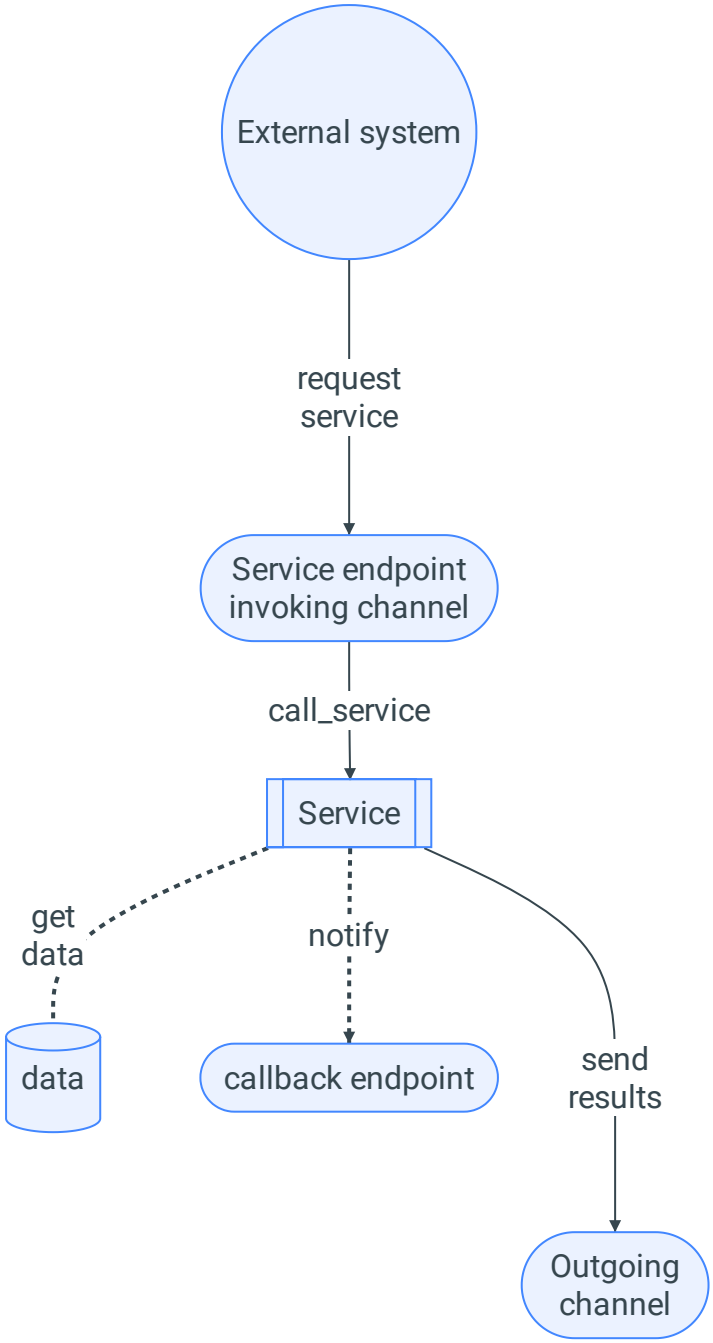
Table of contents:

A service must be written in Python then deployed to **ARSRV** in order to be used.

14.1 Service skeleton

A service has the following high level flow:

- defines a handler in order to be accessed by ARSRV
- it is invoked through a channel
- obtain any required parameters in order to properly do its job
- connects to another channel to read required input, or directly read it, or obtain it from other service, etc (here we are in Python)
- make the necessary transformation over obtained data
- connects to an outgoing channel to write computed output
- log any process details for future references and errors debugging



14.1.1 Detailed operations

A service consists of a class which gives **its name**. This class must contain a method named `handler` each is called by ARSRV to execute the service.

```
# -*- coding: utf-8 -*-
# zato: ide-deploy=True

from zato.server.service import Service

class GetUserDetails(Service):
    """ Returns details of a user by the person's ID.
    """
    name = 'api.user.get-details'

    def handle(self):

        # For now, return static data only
        self.response.payload = {
            'user_name': 'John Doe',
            'user_type': 'SRT'
        }
```

The above example contains:

- first line is a comment for Python but will give important information to ARSRV ref service code serialization, useful to duplicate / copy the service on all servers (for load balancing and fail safe purposes).
- second line is a comment too but for Visual Code IDE add on to know that service should be automatically deployed at save.
- next is a Zato (part of ARSRV) library for right using services
- `self.response.payload` is the property where response must be returned from service processing; this property will be used by ARSRV as response of the service
- `name` will be the name of this service ad used by ARSRV
- the long comment (standard Pyyhon style for a multi line long string) will be used by ARSRV as service description

NOTE. The response format could be anything you want, but for a better serial, serialization and conversion to output channel format, IT IS RECOMMENDED TO USE A DICTIONARY for response payload.

14.2 Deployment

In order to deploy this service the following methods could be used:

- directly from IDE if the corresponding extension was installed - this depends by IDE platform - VS Code has an already written extension
- putting it in directory `~/env/qs-1/server1/pickup/incoming/services` and will be loaded automatically by an ARSRV, server1 shown in path (recommend for automate deployment)
- upload from ARSRV administration console (Services > List > Upload...)

In all cases the deployment ARSRV will distribute the service on all cluster's servers.

14.3 Using in real cases

In most cases will want to access this service by a request from other system. Therefore will be needed a channel (as endpoint) where to invoke the service and sending it data (pls remember that ***anything that is outside ARSRV is 'linked' to ARSRV thru channel***).

There could be cases when want that the service to run automatically driven by a scheduler. As long as ARSRV has its own scheduler, there is not need a channel to invoke the service.

And finally, the service can be invoked by other external event, like a new file in a directory, an updated file, a change in a database, a new message in a queue, a mail, etc. These aspects are ***subject to channels*** and will be treated there.

To produce an usable result, of course, the service must be linked to a channel which will receive response.

ALPHAREN CORE-Integrator (ARINT) System

15 Request and Response Objects

Table of contents:

UPCOMING...

15.1 Introductory in HTTP requests and responses

15.1.1 HTTP protocol

tbd...

15.1.2 HTTP header

tbd...

15.1.3 HTTP data carrying

tbd...

15.2 Request object

tbd...

15.3 Response object

tbd...

ALPHAREN CORE-Integrator (ARINT) System

16 Basic concepts - in channels and calling a service

Table of contents:

16.1 In Channels overview

An **IN channel** is a communication channel defined for *calling (invoking) a service* and act as *request endpoint* seen from outside world.

Channel term

The *IN channel* is also named simple *Channel* meaning that if no other details / hints are given, a "channel" shuld be understood as "IN channel".

Channels can use multiple standard protocols, such as: REST, AMQP, HL7, IBM MQ, JSON RPC, SOAP, Web Sockets, File Transfer protocols, and others.

A channel at request will invoke an existing service.

16.2 REST channel definition

For a *REST* channel, the following parameters must be provided:

- Name
- URL path
- Data format
- Service
- Security definition

Name is the ARCLST name of the channel.

URL path is the address of channel endpoint. This is part of ARCLST route, ie `ARCLST_path.../URL_path`.

Data format is the format of data that will be exchanged through this channel. Usual (for REST channels at least) is to specify here *JSON*.

Service is the name of the service that will be called when channel is invoked.

Security represents the security domain that will be applied to this channel.

Other parameters could also be specified here, for example if there are supplementary parameters (like those with ? after the route), header info (for out channels) and so on.

16.2.1 Invoking the channel

General form of invoking path will be: `http://<user>:<password>@ARCLST_path:11223/URL_path`.

The request is normally made thru load balancer (port 11223). The password is those defined at security domain definition.

NOTE: *The first slash (/) from URL path is part of was entered in definition and not is automatically appended. This will allow for combining channels*.

ALPHAREN CORE-Integrator (ARINT) System

17 Outgoing (channels)

Product 0000-0156 0.0 to current version

- 210731 me new doc
- 210801 me last update

Table of contents:

17.1 Outgoing channels overview

An outgoing channel is a **output endpoint** from a service. It will act as an endpoint usable by a service to access an external system. They will be named as short *OUTGOING* or *OUTCONNS*.

Outgoings can use multiple standard destinations, SAP queues (ex AMQPz IBM), databases, mail and so on.

Outconns are typically invoked (by a service) using attributes from self.out, e.g. self.out.rest, self.out.amqp, self.out.sap and so on, maintaining a connection pool internally when needed so that services can just focus on the invocation part.

17.2 REST outgoing definition

For an **outgoing**, the following parameters must be provided:

- Name
- Host
- URL path
- Data format
- Service
- Security definition

(*NOTE:* it is important to retain the default HEAD ping method, because it will be used to check the endpoint availability)

Name is the ARCLST name of the channel.

URL path is the address of channel endpoint. This is part of ARCLST route, ie `ARCLST_path.../URL_path`.

Data format is the format of data that will be exchanged through this channel. Usual (for REST channels at least) is to specify here *JSON*.

Service is the name of the service that will be called when channel is invoked.

Security represents the security domain that will be applied to this channel.

Other parameters could also be specified here, for example if there are supplementary parameters (like those with ? after the route), header info (for out channels) and so on.

ALPHAREN CORE-Integrator (ARINT) System

18 Channel security

Table of contents:

18.1 Security overview

Mainly security is used to secure channels. As long as a service can interact with external world thru channels, this is clearly enough for all normal operations.

System allow for multiple security models, types and protocols. There ca be active more security rules, each one applicable as needed in various circumstances.

18.2 Define basic security rules

By **basic security** is understood a rule based on requesting explicitly an user and a password.

Basic security allowed **types** are:

- HTTP Basic auth
- JWT
- NTLM
- RBAC
- SSL / TLS
- API keys
- AWS
- Vault
- WS-Security
- X-Path

-#TODO this section START HERE should be reviewed and updated / dropped ---------- Basic security rules can be defined from administration console, *Security* menu, *Basic auth* entry. A security rule means:

- a name for the rule
- an username
- a domain in which rule is applicable (think domain as a kind of grouping more rules in a set usable for a purpose, for example channels); thid approach allows for many to many relationships between security rules and channels or other objects

Password will be generated automatically as uuid4 (guid) and This can be modified latter. -#TODO to review section END HERE ----------

19 Under construction page

UPCOMING...

III. RENware support