Table of Contents

- II Documentation

- II.I User manuals

- II.II Developer manuals

I. System overview

ALPHAREN CORE-Integrator (ARINT) System

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

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

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

2 Core-Integrator System Overview

Table of contents:

2.1 What is ARINT Core

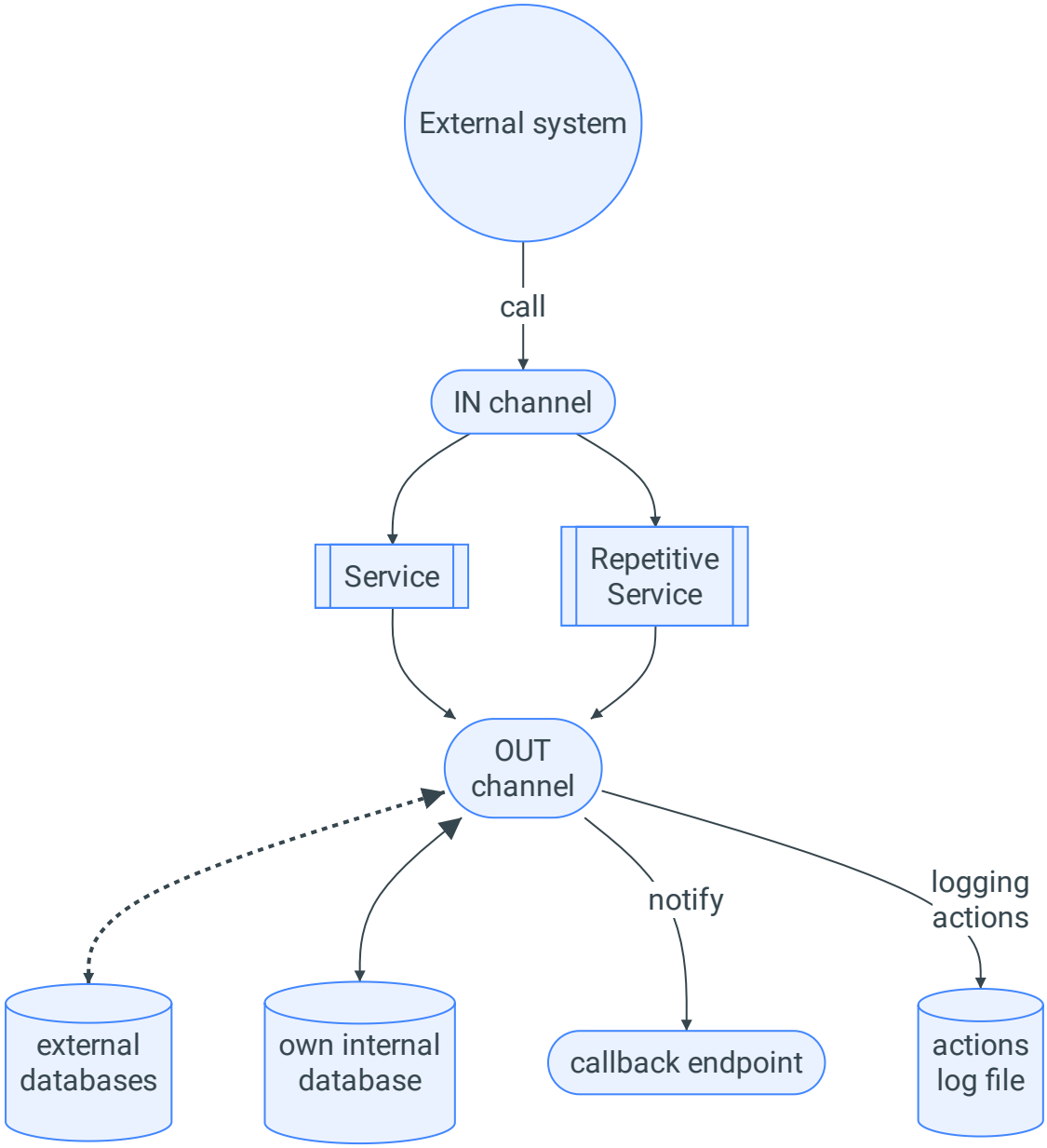
ALPHAREN Core Integrator (aka `ARINT` or `arint`) system is a framework product for automation, integration and interoperability between *distributed systems* or *data sources* basically aimed to build *API oriented*, *middleware*, *frontend* and *backend* applications.

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* 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

- practically an `IN channel` establish a way to address the ARINT system, how to call it
- 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 (cluster containing one or more servers) 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

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

3 Product features

Table of contents:

3.1 General features

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

For an overview of rhe product please see.

3.2 Standards compliance

The following list presents the most known 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 written in 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 so that it is repeatable across environments (ie, 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 elsewhere and how can I better myself?
- How do I manage the complexity of having to integrate at least several systems and dozens or hundreds of APIs?

All these questions have answers 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 | Security*

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

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

4 System Landscape

Document control:

* last update date: 230607

* last updated by: petre iordanescu

Table of contents:

4.1 Basic components

Basic logical components of this system are:

- **(ARCLST)** Integrator Cluster subsystem
  - **(ARSRV)** Physical or virtual Server
  - **(ARLDB)** High Availability assurance service
  - **(ARWADM)** Web admin console interface
  - **(ARSCHED)** Scheduler
  - **(ARKVD)** Key-Value Data store

- **(ARPuSuB)** Publisher Subscriber Queues
- **(ARVPN)** Integrator VPN access

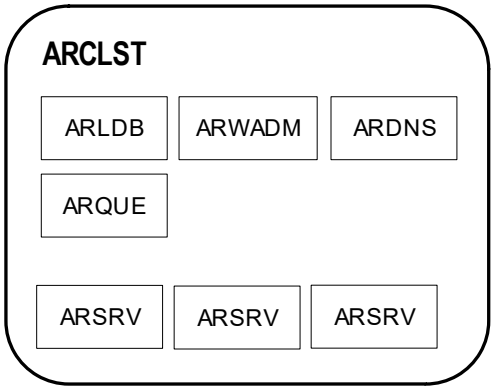
- **(ARDPX)** Discovery Service, Distribution proxy (dynamic DNS)
- **(ARMAIL)** Integrator mail
- **(#TODO)** Configuration portal #NOTE: not yet assigned code

4.2 System Blueprints

4.2.1 ARint blueprint

-#TODO a high level blueprint

4.2.2 ARCLST blueprint



-#TODO start and make new descriptions (based on existing) for each component

-#TODO - from here continue review

4.3 ARCLST. Integrator Cluster

This component creates a local cluster formed by one or more **ARSRV** machines. Particularly can stand on one single machine with **ARSRV**.

This is not recommended because **ARCSLT** is a *network-bounded* system and **ARSRV** is a *cpu-bounded* one, and a *cluster to cluster* integration will have to suffer.

This component can run **1 per LAN machine**.

4.4 ARSRV. Integrator Server

This is the core / heart of each machine. It will assure information getting, processing and sending or streing.

Other functionalities (in cooperation with **ARCLST**) cover scheduling, asynchronous processing and retrying in case of un-availability of an external system.

This component can run **n per cluster**.

4.5 ARLDB. Integrator High Availability assurance subsystem

This component assure:

- load balancing,
- failure detection,
- service availability,
- RTT ordering access to in case of multiple **ARSRV** modules.

All **ARSRV** components work *ACTIVE ACTIVE* inside any **ARCLST**. Of course, clusters work independently each of the others.

Also, each **ARLDB** keeps a dynamic trace of any **ARCLST** from the system, so a new cluster can be added without the need of any downtime.

This thing is also applicable inside a cluster where at any time, with any downtime, a new **ARSRV** can be added. If is right configured then will be automatically discovered and made part of cluster.

This component can run **1 per cluster**.

4.6 ARWADM Web amin console

This will assure cluster administration, for all its servers and other components.

This component can run **n per cluster**. The reason for more ARWADM is to secure each of them.

4.7 ARDPX. Access and distribution proxy

This module is useful when an **ARCLST** is buit on **ARSRV**s physically implemented as a set of small virtual machines on a single server, having their LAN. Sure, ALPHA-REN hardware will assure that, but if you're using other hardware it will be needed.

Thiz module will stay in own LAN DMZ being directly exposed on **ARCLST** IP external access.

This module is responsible for:

- access the system outside its LAN without the need of a router with port forwarding.
- assurance of all reverse proxy operations.
- access on the **ARCLST** and **ARSRV**s outside cluster LAN.

This component can run **1 per machine**.

4.8 ARMAIL Integrator mail

This module is responsible for sending administrative and notification mails from **ARCLST** cluster.

This component can run **1 per machine**.

4.9 ARVPN. Integrator VPN access

This module assure VPN access into the **ARCLST** cluster.

4.10 Deployment over multiple LAN environments

In an environment with multiple LANs, in deployment architecture and process should consider the following aspects:

- every LAN should have at least its own **ARSRV** in order to communicate with other LANs
- an **ARCLST** can assure balancing and failure services inside LAN
- in order to assure balancing and failure services over LANs, each one must have its own **ARCLST** (wirh all other required components to assure corresponding services) which communicate with the others.
- a queue service is strictly required both to assure messages transport inside LAN, but also between LANs; for this reason cannot be used any queuing system but one with remote (over LANs) capabilities (aka named broker system)

ALPHAREN CORE-Integrator (ARINT) System

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

5 Licensing Editions and Pricing

Table of contents:

UPCOMING...

ALPHAREN CORE-Integrator (ARINT) System

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

6 Services

Table of contents:

UPCOMING...

ALPHAREN CORE-Integrator (ARINT) System

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

7 Training Programmes

Table of contents:

UPCOMING...

II. Documentation

II.I User manuals

ALPHAREN CORE-Integrator (ARINT) System

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

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

ALPHAREN CORE-Integrator (ARINT) System

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

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)

II.II Developer manuals

ALPHAREN CORE-Integrator (ARINT) System

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

10 Development Overview

Table of contents:

10.1 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.

10.2 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

10.3 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

10.4 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

10.5 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

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

11 Service anatomy

Product 0000-0156 0.0 document control:

- 210728 me new doc
- 230817 me last update

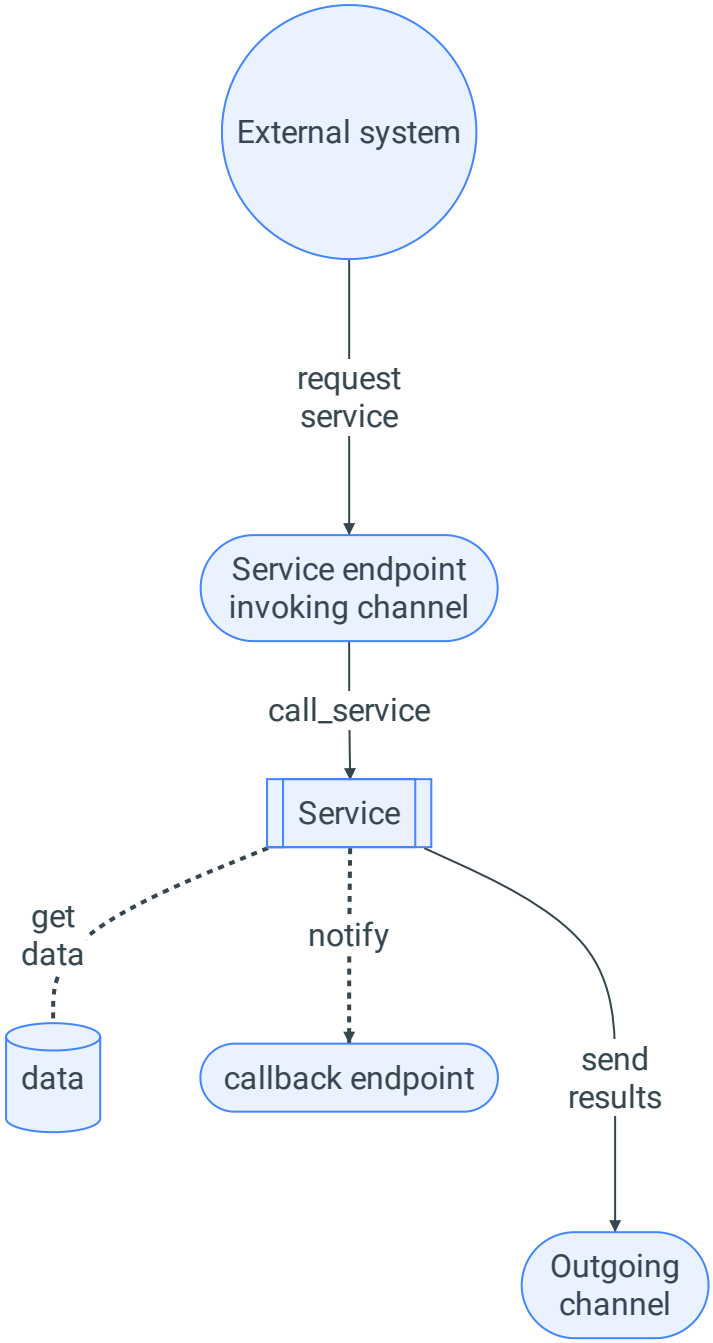
Table of contents:

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

11.1 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



11.1.1 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.

11.2 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.

11.3 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

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

12 Request and Response Objects

Table of contents:

UPCOMING...

12.1 Introductory in HTTP requests and responses

12.1.1 HTTP protocol

tbd...

12.1.2 HTTP header

tbd...

12.1.3 HTTP data carrying

tbd...

12.2 Request object

tbd...

12.3 Response object

tbd...

ALPHAREN CORE-Integrator (ARINT) System

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

13 Basic concepts - in channels and calling a service

Table of contents:

13.1 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.

13.2 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.

13.2.1 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

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

14 Outgoing (channels)

Product 0000-0156 0.0 to current version

- 210731 me new doc
- 210801 me last update

Table of contents:

14.1 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.

14.2 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

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

15 API Security

Table of contents:

15.1 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.

15.2 Define basic security rules

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

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.

ALPHAREN CORE-Integrator (ARINT) System

(c) 2021 RENware Software Systems. RESTRICTED only for project internal use

16 System Data and Objects

Table of contents:

UPCOMING...

17 Under construction page

UPCOMING...

III. RENware support