Table of Contents

1 ALPHAREN Integrator

2 Product information

- p/n code: **0000-6151**
- product short name: **arint**
- initial start: 2021

3 Documentation type

This is product design documentation (directory `810-DSGN`) and contains the following topics:

- a general design documentation: **Design documentation** area
- a set of basic development "how to(s)": **Development how to(s)** section

I. Design documentation

ALPHAREN ARINT System

4 Overview

Document control:

* last update date: 210712

* last updated by: petre iordanescu

5 Integrator System Overview

Integrator system is a model of ALPHA-REN *appliance* which assure integration at services level between different systems.

It acts as high level *"ESB, ESOA"* to connect different services and to make them to work ***as one*** without the human intervention, ie, is automated.

6 Basic features

- **anywhere**, can work even the systems that must be integrated are in different non routable LANs
- **anyhow**, is agnostic to form and look 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 standard (ie, defined as *RFC*) Internet security

7 Detailed 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
- Protocols, industry standards and data formats - REST, SOAP, Odoo, AMQP, HL7, MongoDB, Redis, SAP, IBM MQ, SQL, OpenAPI, ZeroMQ, WebSockets, Cassandra, Amazon S3, Swift, LDAP, Active Directory, Kafka, SMTP, IMAP,

FTP, SFTP, ElasticSearch, Solr, Memcached, Twilio, Vault, Slack, Telegram, RBAC, JMS, integration patterns, cryptography, security and more

8 Typical use cases

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

Real-world, production ***ALPHA-REN 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

ALPHAREN ARINT System

9 Landscape

Document control:

* last update date: 230607

* last updated by: petre iordanescu

10 Basic components

Basic logical components of this system are:

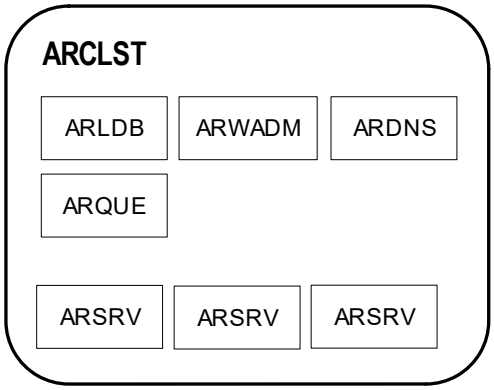
- **(ARCLST)** Integrator Cluster subsystem
  - **(ARSRV)** Physical or virtual Server
  - **(ARLDB)** High Availability assurance service
  - **(ARWADM)** Web admin console interface
  - **(ARSCHED)** Scheduler
  - **(ARKVD)** Key-Value Data store
  - **(ARPuSuB)** Publisher Subscriber Queues
  - **(ARVPN)** Integrator VPN access
- **(ARDPX)** Discovery Service, Distribution proxy (dynamic DNS)
- **(ARMAIL)** Integrator mail
- **(#TODO)** Configuration portal #NOTE: not yet assigned code

11 System Blueprints

11.1 ARint blueprint

-#TODO a high level blueprint

11.2 ARCLST blueprint



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

-#TODO - from here continue review

12 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**.

13 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**.

14 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**.

15 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.

16 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**.

17 ARMAIL Integrator mail

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

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

18 ARVPN. Integrator VPN access

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

19 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)

II. System administration

ALPHAREN ARINT System

20 Installation

Document control:

* last update date: 230605

* last updated by: petre iordanescu

21 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
```

21.1 Package key

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

21.2 Add apt repository

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

21.3 Install zato

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

21.4 Apply latest updates

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

21.5 Check & confirm

```
zato --version
```

22 Create a quick cluster environment

22.1 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
```

22.2 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
```

23 Configuration & access

23.1 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

23.2 Machine environment configuration

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

23.3 System launch scripts

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

Server start in background mode, NOT as daemon.

24 Installation notes

None. Everything works as documented. ATTN open 8183 port

III. Development how to(s)

RENware ALPHA-REN System

25 Development Overview

Product 0000-0156 0.0 to current version

- 210728 me new doc

25.1 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.

25.2 CHN - channel

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

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

25.3 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

---

25.4 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

25.5 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.

RENware ALPHA-REN System

Product 0000-0156 0.0 document control:

- 210728 me new doc
- 230817 me last update

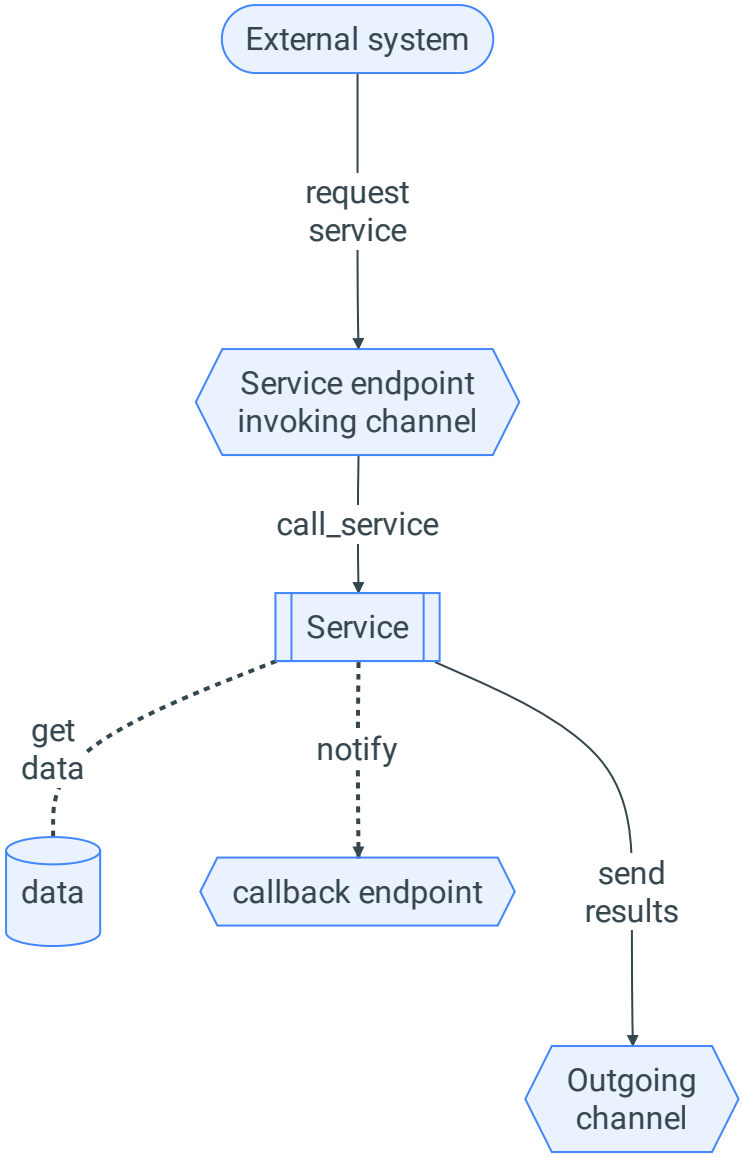
26 Service anatomy

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

26.1 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



26.1.1 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.

26.2 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.

26.3 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.

RENware ALPHA-REN System

27 Channels

Product 0000-0156 0.0 to current version

- 210730 me new doc
- 210731 me last update

27.1 Channels overview

A channel is a **request endpoint** from outside world.

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.

27.2 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.

27.2.1 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*.

RENware ALPHA-REN System

28 Outgoing (channels)

Product 0000-0156 0.0 to current version

- 210731 me new doc
- 210801 me last update

28.1 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.

28.2 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.

RENware ALPHA-REN System

29 API Security

Product 0000-0156 0.0 to current version

- 210730 me new doc

29.1 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.

29.2 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.