ADLINK

VORTEX LINK

User Guide

Release 3.0.1

Contents

1

What is Vortex Link ?

Vortex Link is the ubiquitous and universally accessible 'Internet Service' for sharing data between Vortex-enabled applications. Vortex Link can be deployed on public or private clouds, it is elastic, fault-tolerant and highly scalable. Vortex Link can be deployed in any subsystem (typically a LAN) to allow all DDS applications to share data with other DDS applications in the cloud or in any other subsystem.

Thanks to its flexible architecture, Vortex Link supports several different deployment and connectivity scenarios, including Device to Cloud and System to Cloud, as well as system federation. Additionally, Vortex Link supports multiple protocol transports, such as TCP/IP and UDP/IP (unicast and multicast) and can even mediate between transports. Security is supported in Vortex Link, taking advantage of internet standards such as TLS.

This User Guide will get you started with Vortex Link.

If you have any questions, please contact us at ADLINK support .

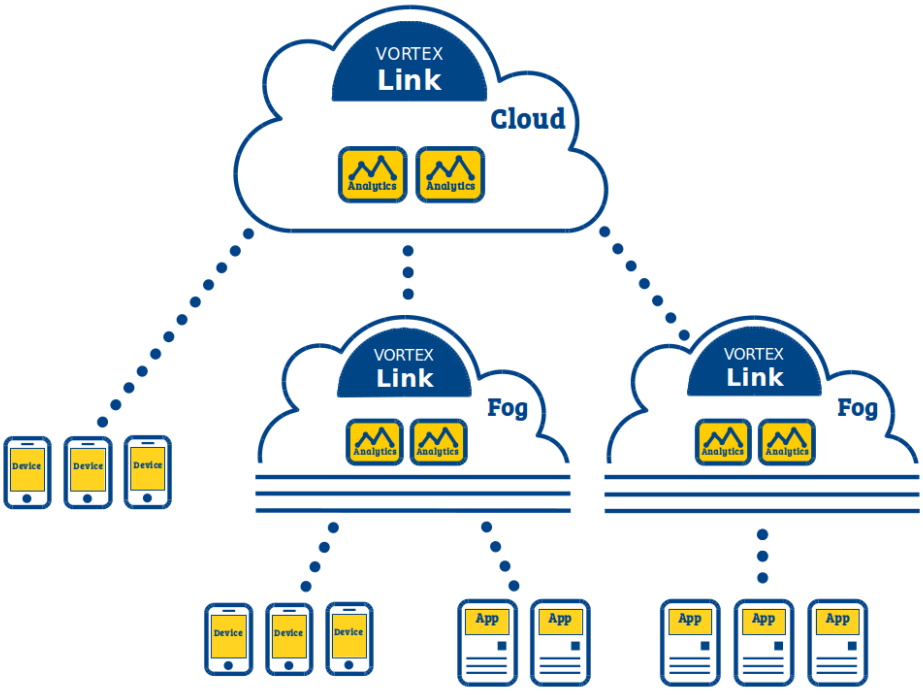


Fig. 1.1: **Vortex Link**

2

How do Vortex Link works?

Discovery

In usual DDS deployments (single LAN), the discovery between DDS applications is achieved peer-to-peer with the help of UDP multicast. But in systems where UDP multicast is partially or not available (typically in WAN), Vortex Link provides another way for DDS applications to discover each other.

The Vortex Link services will discover the end-user DDS applications that are running locally (in the same LAN) through UDP multicast and applications that connect to them through TCP. The Vortex Link services will collect informations about end-user applications DomainParticipants and sub-entities (Writers and Readers) and propagate those informations to the other Vortex Link services in the system when needed. Finally, Vortex Link services will propagate Writers and Readers informations to all end-user applications that need to communicate with them.

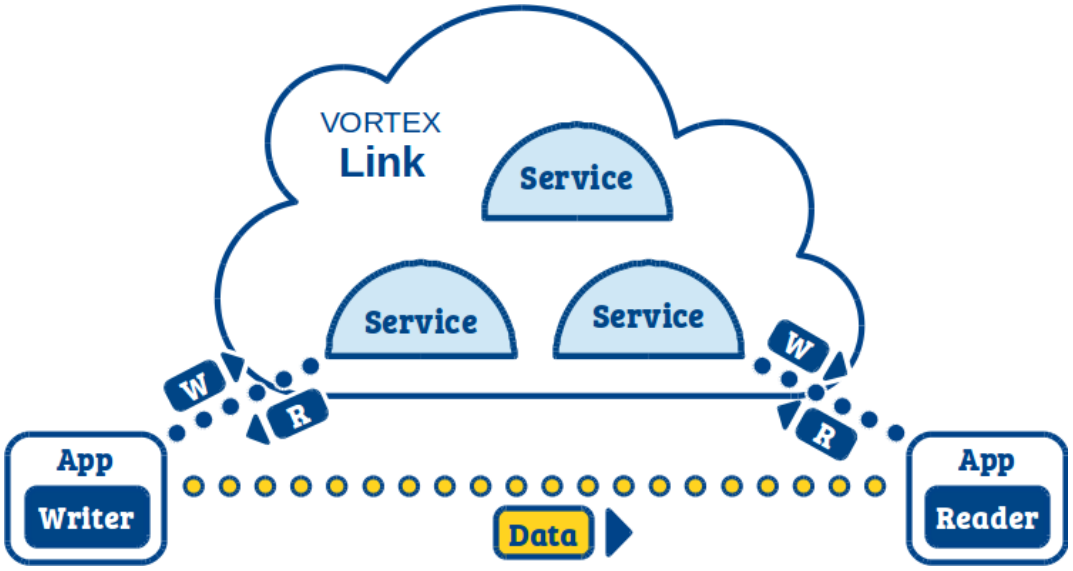


Fig. 2.1: **Vortex Link discovery**

Routing

In the types of environments Vortex Link is designed to address, several circumstances may lead two user applications to be unable to communicate directly with each other:

- Both user applications are using TCP transport, but both are deployed behind a different NAT (on a host that does not have a public address).
- Both user applications are using UDP-multicast transport but are deployed in two different LANs (or private clouds).

- Both user applications use a different transport. One is deployed in a LAN and uses UDP-multicast transport, the other is deployed in a WAN and uses TCP transport.

In such cases, even if they discovered each other with the help of Vortex Link, user applications are not able to communicate with each other and exchange data. Vortex Link services are able to establish a route for data exchanges between the two applications : they will act as bridges between end-user applications, so that two matching end-user applications can communicate with them and exchange data through them.

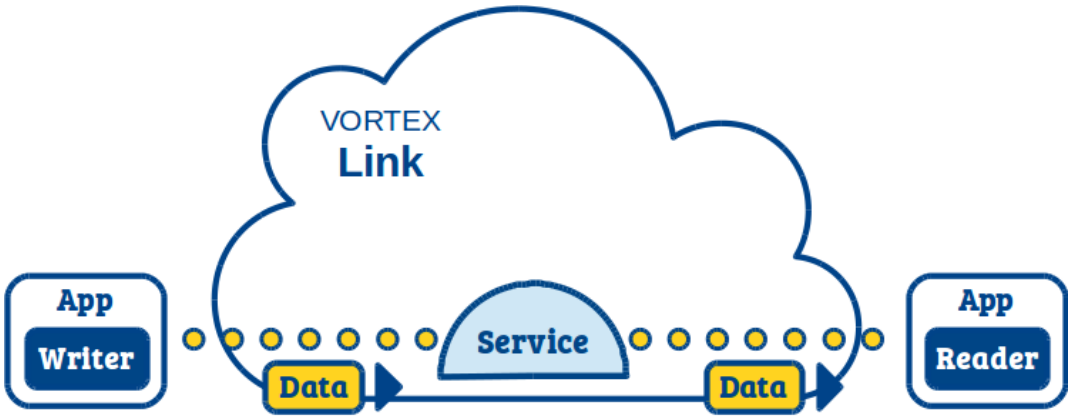


Fig. 2.2: **Vortex Link routing**

3

How to deploy and configure Vortex Link?

The first thing to define when deploying and configuring Vortex Link is the kind of deployment that should be used to satisfy the needs of the system you want to set up.

Device-to-Device & Device-to-Link

If you simply need to make several applications using TCP transport and spread over internet (devices) to communicate with each other using DDS then you typically need a Device-to-Device deployment.

If you also need some applications deployed in the cloud (typically analytics) and using TCP or UDP transport to participate to the DDS domain with the device applications then you typically need a Device-to-Device & Device-to-Cloud deployment.

For such deployment you simply need to deploy a single Vortex Link service in a cloud or any host that has a public IP address or is publicly accessible.

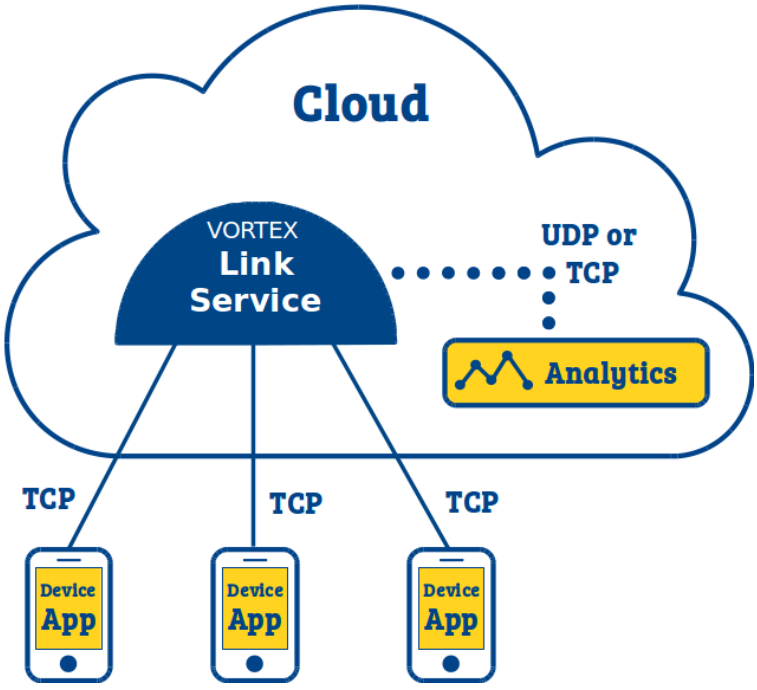


Fig. 3.1: **Device-to-Device & Device-to-Link**

Configure the Vortex Link service

There is nothing mandatory to configure here. You can simply run a Vortex Link service somewhere in the cloud.

```
java -jar link-service.jar
```

You may anyway want to configure :

- The tcp port the service will listen on for incomming TCP connections : std,std-reflink.tcp.port (default 7400).
- The domainid that should be used to discover local applications running in the cloud and using UDP multicast : std,std-reflink.domainid (default 0).
- The public address of the host running the service. If the host running the service has a private address but is reachable from the rest of the world through a public address, then you need to specify this public address using the std,std-reflink.externalNetworkAddresses property.

Configure the device applications

When running end-user applications, we need to configure them to use the TCP transport and to point them to the Vortex Link service. Here is an example of such a configuration starting a Vortex Café application:

```
java
     -Ddds.network.udp.enabled=false
     -Ddds.discovery.tcp.peers=ip_service:7400
     -jar ishapes.jar
```

Configure the cloud applications (analytics)

If the cloud supports multicast communications, then you have nothing particular to configure here. You only need to make sure that the domain id used by the application is the same than the one configured in the Vortex Link service (std,std-reflink.domainid).

If the cloud does not support multicast communications, then you need to configure cloud applications to use the TCP transport and to point them to the Vortex Link service, as done for any device application.

LAN to LAN

If you want to interconnect two or more sub systems (potentially legacy systems) each one running a DDS domain in a LAN using UDP multicast then you typically need a LAN-to-LAN deployment.

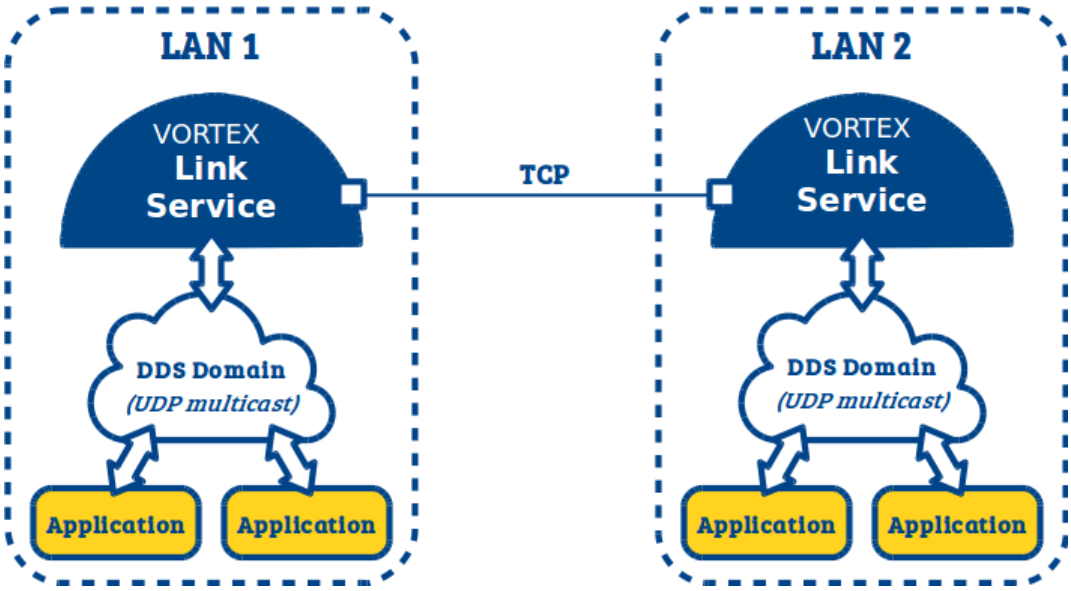


Fig. 3.2: **LAN to LAN**

For such deployment, you need to deploy a single Vortex Link service in each sub system (LAN) on a host that is publically accessible.

You will have to make sure that :

- Each deployed Vortex Link service is configured with the proper domainid to discover local applications through UDP multicast
- Each deployed Vortex Link service is configured to connect to the other services.

If the vortex Link services are deployed on hosts that cannot directly connect to each other (behind a symetric NAT) then you need an Indirect LAN to LAN deployment.

Vortex Link services deployed on the NAT hosts

Each sub system is generally deployed behind a NAT host. A NAT host typically has two network interfaces : one interface with a private IP address that communicates with the sub system hosts and one interface with a public address accessible from the rest of the world.

If you can deploy your Vortex Link services in those NAT hosts, you will have to :

- Configure the Vortex Link services to use the public interface for TCP communications.
- Configure the Vortex Link services to use the private interface for UDP communications.
- Configure the Vortex Link services to connect to each other using their public addresses.

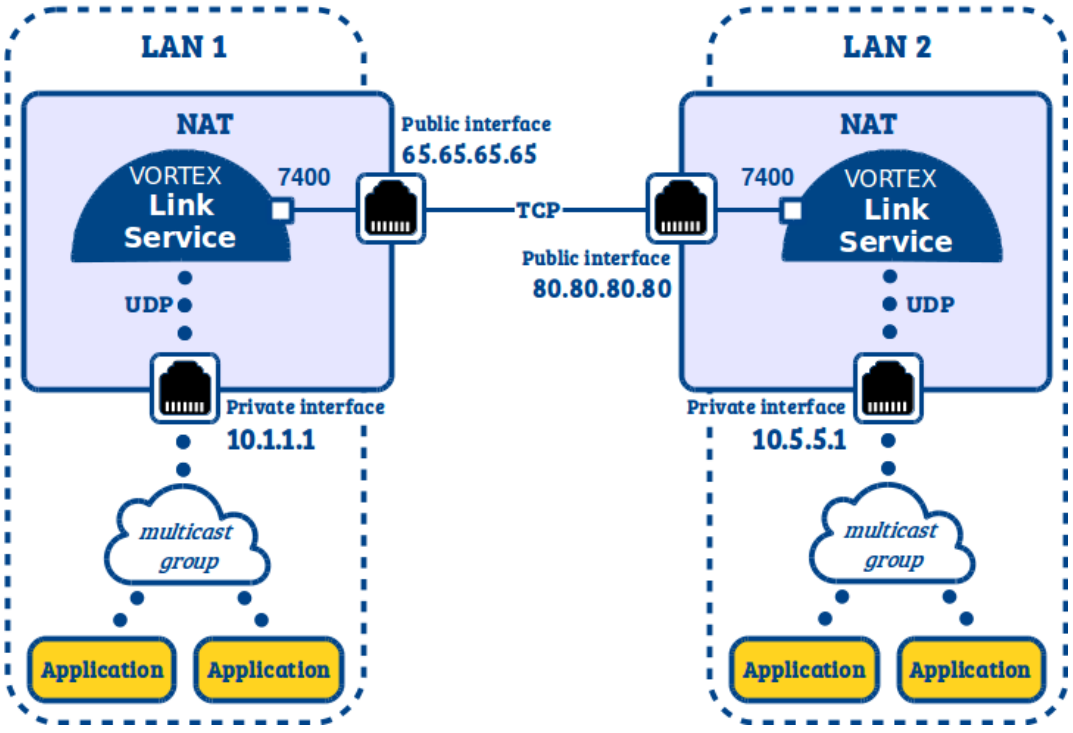


Fig. 3.3: **LAN to LAN (Link on NAT)**

Vortex Link service configuration for LAN 1

```
java
     -Dlink.domainid=0
     -Dlink.tcp.interface=65.65.65.65
     -Dlink.udp.interface=10.1.1.1
     -Dlink.tcp.peers=80.80.80.80:7400
     -jar link-service.jar
```

Vortex Link service configuration for LAN 2

```
java
     -Dlink.domainid=0
     -Dlink.tcp.interface=80.80.80.80
     -Dlink.udp.interface=10.5.5.1
     -Dlink.tcp.peers=65.65.65.65:7400
     -jar link-service.jar
```

NAT hosts configured for ports redirection

If you can't deploy your Vortex Link services in the NAT hosts, maybe you can configure your NAT to redirect ports to a given host in the sub system (LAN), so that this host is accessible from the outside world (using the NAT public address).

If so, you can deploy your Vortex Link services on those publicly accessible hosts and make sure that they can connect to each other. You will have to :

- Configure the Vortex Link services with the public IP address of the NAT host of their sub system with the help of the std,std-reflink.externalNetworkAddresses option.
- Configure the Vortex Link services to connect to each other using the public IP addresses of their respective NAT hosts.

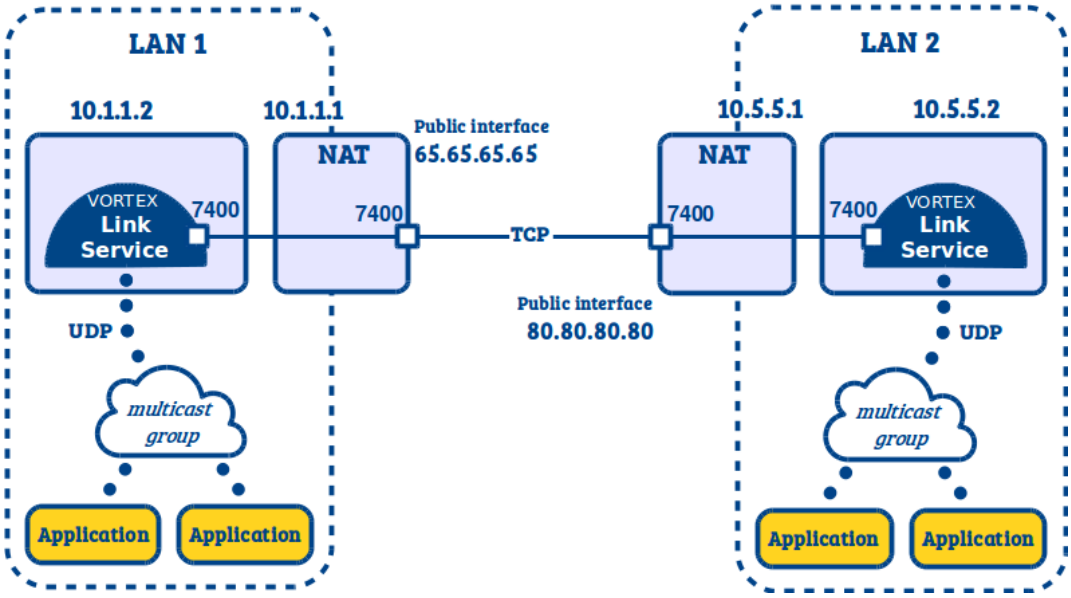


Fig. 3.4: **LAN to LAN (ports redirection)**

Vortex Link service configuration for LAN 1

```
     java
          -Dlink.domainid=0
          -Dlink.tcp.peers=80.80.80.80:7400
          -Dlink.externalNetworkAddresses=65.65.65.65
          -jar link-service.jar
```

Vortex Link service configuration for LAN 2

```
     java
          -Dlink.domainid=0
          -Dlink.tcp.peers=65.65.65.65:7400
          -Dlink.externalNetworkAddresses=80.80.80.80
          -jar link-service.jar
```

Indirect LAN to LAN

If you want to interconnect two or more sub systems (LAN-to-LAN deployment) but need to deploy your Vortex Link services on hosts that have no public address and are not publicly accessible, then you need to deploy a single Vortex Link service in each sub system (LAN) but you also need to deploy a Vortex Link service on a public host, typically in a cloud. This extra Vortex Link service will act as an intermediary between the sub systems and route data between them.

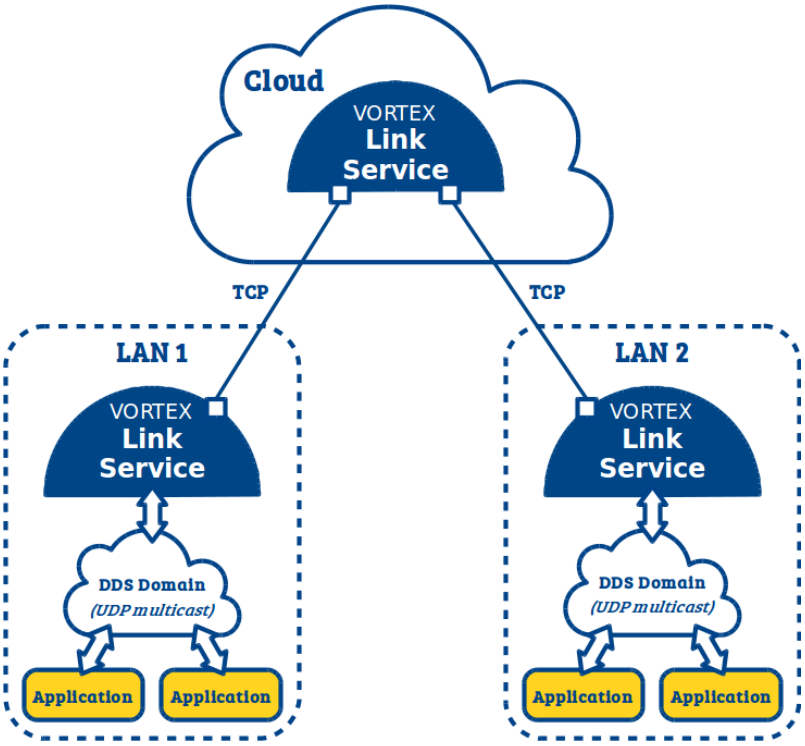


Fig. 3.5: **Indirect LAN to LAN**

You will have to make sure that :

- Each Vortex Link service deployed in a sub system is configured with the domainid used by the UDP applications deployed in the sub system (LAN).
- Each Vortex Link service deployed in a sub system is configured to connect to the Vortex Link service deployed in the cloud.
- The Vortex Link service deployed in the cloud is configured with a service level (std,std-reflink.serviceLevel) greater than the service level configured on the Vortex Link services deployed in the sub systems (default 0).

Configure the Vortex Link service running in the cloud

```
java
     -Dlink.serviceLevel=1
     -jar link-service.jar
```

Configure the Vortex Link services running in the sub systems

```
java
     -Dlink.domainid=0
     -Dlink.tcp.peers=link_host_ip:7400
     -jar link-service.jar
```

Configure the sub systems applications

There is nothing particular to configure for the sub systems applications. By default they will use UDP multicast and be discovered and routed by the Vortex Link services.

4

The Vortex Link service

Vortex Link capabilities

Vortex Link is able to:

- discover DDS applications over UDP multicast, UDP unicast and TCP
- route the DDS traffic between DDS applications it discovered and that can't communicate directly with each other
- route the DDS traffic between a DDS application and another Vortex Link service, that will itself route the traffic to another DDS application.
- act as an intermediate routing point between 2 Vortex Link services

Running a Vortex Link service

The Vortex Link service is a single executable jar:

link-service.jar

You can simply run it like this:

```
java -jar link-service.jar
```

Configuring the Vortex Link service

The Vortex Link service is configured through java properties. You may define those java properties on the command line with the help of `-D` flags. You may also use a java properties file somewhere in the `classpath`.

By default the Vortex Link service will look for a file named `vortex_link.properties`, but you can point to any file like this:

```
-Dvortex_link.properties=myfile
```

Note that the value of this configuration property can be any file location as specified in std,std-refFiles locations.

The most usual configuration properties that may be configured for a Vortex Link service (depending on the use case) are the following:

- link.domainid : Specifies the DDS domain on which the Vortex Link service should discover UDP multicast user applications.
- link.tcp.port : Specifies the TCP port on which the Vortex Link service should listen for incomming connections from user applications and other services.
- link.tcp.peers : Specifies the list of Vortex Link services that this service must try to connect to. Note that with discovery propagation each service does not need to be configured with all remote services as peers. Only some bootstrap peers are sufficient.

- link.serviceLevel : In hierarchical deployments this property specifies the level of the configured service in the services hierarchy. This property is also necessary when using Vortex Link in Indirect LAN to LAN deployments.
- link.externalNetworkAddresses : When the Vortex Link service is deployed behind a NAT, this property should be set with the public IP of the NAT, or to 'none' in case of symetric NAT (*i.e.* the host can't be contacted from outside).

Vortex Link service configuration properties

link.domainid

Valid values: `0 - 230`

Default value: `0`

Purpose: This configuration property indicates in which DDS domain the Vortex Link service should discover user applications. This property has an impact only when discovering user applications using UDP-multicast transport.

- When discovering UDP-multicast based user applications, only applications participating in the configured DDS domain will be discovered.
- When discovering TCP-based user applications, any application that connects to the Vortex Link service will be discovered whichever DDS domain they are participating in.

UDP multicast can also be used for communications between Vortex Link services when possible. In such situations, the several services that are supposed to communicate with each other through UDP multicast must be configured with the same domainid.

link.tcp.port

Valid values: `0 - 65535`

Default value: `7400`

Purpose: This property indicates which TCP port should be opened to listen for connections from user applications and from other services.

This port number should be used in user applications configuration so that they contact this Vortex Link service.

link.tcp.peers

Valid values: *list of locators*

Default value: *not defined*

Purpose: This configuration property indicates the locator, or list of locators to which this service should try to connect to in order to communicate with the other services.

If discovery propagation is disabled, each service need to be configured with the locators of all the other services of the system.

If discovery propagation is enabled (default) then each service may be configured with only one or a part of the other services in the system. It is recommended that more than one locator is defined here for fault tolerance purposes.

See std,std-refLocators & locator list, for more details about locators.

link.externalNetworkAddresses

Valid values: *a list of locators (optionally with ports) or 'local' or 'none'*

Default value: `local`

Purpose: This configuration property specifies which locator(s) (*address:port*) the Vortex Link service will advertise to applications and to other Vortex Link services in its discovery protocol exchanges. The port is optional (if not specified, the actual opened port will be advertised).

When the value `local` is configured, then the Vortex Link service will use the local address of the network interface it is bound to.

So if the Vortex Link service is deployed on a host that is reachable from any host of the system then `local` should be used.

If the Vortex Link service is deployed on a host behind a NAT with port redirection (the local ip address is different from the public ip address), then the public address should be configured here.

See std,std-refLocators & locator list, for more details about locators.

link.tcp.externalNetworkAddresses

Valid values: *a list of locators (optionally with ports) or 'local' or 'none'*

Default value: `local`

Purpose: This configuration property specifies which TCP locator(s) (*address:port*) the Vortex Link service will advertise to applications and to other Vortex Link services through TCP in its discovery protocol exchanges. The port is optional (if not specified, the actual opened port will be advertised).

When the value `local` is configured, then the Vortex Link service will use the local address of the network interface it is bound to.

So if the Vortex Link service is deployed on a host that is reachable from any host of the system then `local` should be used.

If the Vortex Link service is deployed on a host behind a NAT with port redirection (the local ip address is different from the public ip address), then the public address should be configured here.

See std,std-refLocators & locator list, for more details about locators.

link.udp.externalNetworkAddresses

Valid values: *a list of locators (optionally with ports) or 'local' or 'none'*

Default value: `local`

Purpose: This configuration property specifies which UDP locator(s) (*address:port*) the Vortex Link service will advertise to applications and to other Vortex Link services through UDP in its discovery protocol exchanges. The port is optional (if not specified, the actual opened port will be advertised).

When the value `local` is configured, then the Vortex Link service will use the local address of the network interface it is bound to.

So if the Vortex Link service is deployed on a host that is reachable from any host of the system then `local` should be used.

If the Vortex Link service is deployed on a host behind a NAT with port redirection (the local ip address is different from the public ip address), then the public address should be configured here.

See std,std-refLocators & locator list, for more details about locators.

link.network.interface

Valid values: *any string*

Default value: `auto`

Purpose: This configuration property indicates which network interface should be used to listen for end user applications and other services and to communicate with them for both UDP and TCP transports.

If link.tcp.interface is set, it will override this property for the TCP transport.

If link.udp.interface is set, it will override this property for the UDP transport.

When the default value `auto` is used, the Vortex Link service will choose the most suitable interface among the available ones. To determine the most suitable network interface, the Vortex Link service ranks the eligible interfaces by quality, and then selects the interface with the highest quality. If multiple interfaces are of the highest quality, it will select the first enumerated one. Only interfaces that are up and have an IPv4 address family are eligible.

Quality is then determined as follows:

- interfaces with a non-link-local address are preferred over those with a link-local one;
- multicast-capable is preferred, or if none is available
- non-multicast capable but neither point-to-point, or if none is available
- point-to-point, or if none is available
- loopback

link.tcp.interface

Valid values: *any string*

Default value: `auto`

Purpose: This configuration property indicates which network interface should be used to listen for end user applications and other services and to communicate with them on TCP transport.

When set, this property will override the value of property link.network.interface for TCP transport.

When the default value `auto` is used, the behavior of this property is equivalent to the behavior of link.network.interface property.

link.udp.interface

Valid values: *any string*

Default value: `auto`

Purpose: This configuration property indicates which network interface should be used to listen for end user applications and other services and to communicate with them on UDP transport.

When set, this property will override the value of property link.network.interface for UDP transport.

When the default value `auto` is used, the behavior of this property is equivalent to the behavior of link.network.interface property.

link.serviceLevel

Valid values: `0 - 2147483647`

Default value: `0`

Purpose: In hierarchical deployments this property specifies the level of the configured service in the services hierarchy. Each discovered service with a higher level will be considered as a 'parent' service and each discovered service with a lower level will be considered as a 'child' service.

This property is also necessary when using Vortex Link in Indirect LAN to LAN deployments (that actually are hierarchical deployments with only 2 levels). In such deployments, the Vortex Link service is deployed outside of any LAN (typically in a cloud) and thus on the top of all Vortex Link services in LANs. Therefore it must be configured with a higher level value than the Link services.

A 'parent' service will route data between it's 'child' services if needed. Note that Vortex Link has the capacity to route between services and can also be deployed at the bottom of hierarchical systems with the lower level values.

link.cluster.id

Valid values: *any string* , `"auto"`

Default value: `""`

Purpose: For fault tolerance and load balancing purposes, it is possible to deploy several replicas of a service and to group them in a cluster. Each cluster must be identified with a cluster id and all replicas in the cluster must be configured with the same cluster id (see std,std-refClusters & Cluster ID).

When set to `"auto"`, Vortex Link will configure the clusters automatically by setting the same cluster id to all services that are deployed in a same IP subnetwork (see std,std-refAutomatic cluster id).

link.loadBalancing.pluginClass

Valid values: *any string*

Default value: `vortex.lb.plugins.PerParticipantHashPlugin`

Purpose: This property defines which plugin implementation should be used to balance data flows on the several replicas of the cluster. Two implementations are available :

- `vortex.lb.plugins.PerParticipantHashPlugin`
- `vortex.lb.plugins.PerWriterHashPlugin`

See std,std-refLoad balancing plugins for more details on the plugins and their behavior.

link.directConnection

Valid values: `true` , `false`

Default value: `false`

Purpose: As discussed in std,std-refRouting chapter, routing may not be necessary in all cases but only when concerned end-user applications cannot directly communicate with each other. But the current version of the Vorex Link service is not able to detect if end-user applications can directly communicate or not (except when both applications are deployed on the same LAN and use UDP multicast). So it has two deployment modes:

- If the property is set to `true` (direct connection mode), all end-user applications will always communicate directly with each other.
- If the property is set to `false` (routing mode), all end-user applications will always communicate with each other through a Vortex Link service.

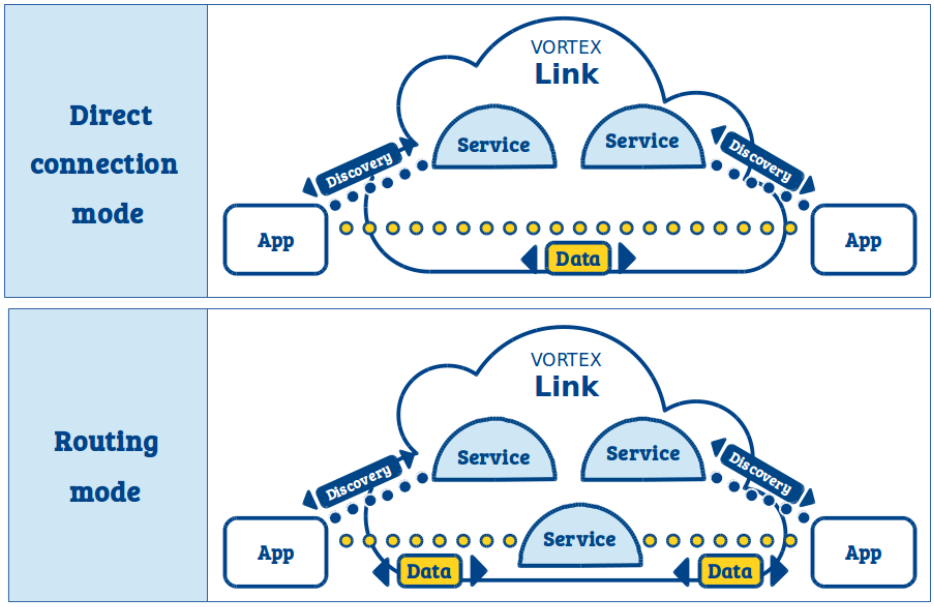


Fig. 4.1: **Direct Connection and Routing Modes**

link.participantsAdvertisment

Valid values: `never` , `always`, `otherVendors`

Default value: `otherVendors`

Purpose: When active and when the service advertises a Participant of the existence of a matching DataReader or DataWriter (sending a SEDP message), it will also advertise this Participant of the existence of the parent Participant of the matching DataReader or DataWriter, sending a SPDP message before the SEDP message. This SPDP message will contain the newly discovered Participant's GUID, but the service's locators and an infinite duration lease (to avoid the sending of periodical SPDP to all Participants, that will affect scalability).

Note that the discovery of DataReaders and DataWriters is usually sufficient for the Vortex DDS implementations, without the discovery of their parent Participant. But some other implementations require to discover the parent Participant before to consider the discovery of DataReaders and DataWriters.

Valid values are :

- `never`: the service will never advertise the other Participants when discovering a new Participant
- `always`: the service will always advertise the other Participants when discovering a new Participant
- `otherVendors`: the service will advertise only the non-Vortex Participants when discovering a new Participant

An invalid value will be equivalent to `never`.

link.servicesDiscoveryPropagation

Valid values: `true` , `false`

Default value: `true`

Purpose: The discovery propagation facilitates the discovery of other services when using TCP transport between services. When activated, each time the service discovers a new remote service, it propagates the information about this new service to all the other services it already knows. So it is not necessary to configure each service with the locators of all the other services, which makes the system more flexible.

link.blockBuiltinPartition

Valid values: `true` , `false`

Default value: `false`

Purpose: This property enables the filtering out ('not routing') of all entities using the `_BUILT-IN PARTITION_`. The `_BUILT-IN PARTITION_` is a partition use by Vortex OpenSplice for it's buit-in entities. When blocking this partition, you will loose some functionalities in Vortex OpenSplice. But those buit-in entities may cause problems in large scale systems. That's why it may be necessary to block this `_BUILT-IN PARTITION_` in some cases.

link.blockDurabilityPartition

Valid values: `true` , `false`

Default value: `false`

Purpose: This property enables the filtering out ('not routing') of all entities using the `durabilityPartition`. The `durabilityPartition` is a partition use by all Vortex products to provide the durability functionality. When blocking this partition, durability (TRANSIENT, PERSISTENT) will not work any more. But it may be necessary to block this `durabilityPartition` in large scale systems.

link.blacklist.file

Valid values: *any string*

Default value: *not defined*

Purpose: Identifies a black list file location. This file contains a list of *partition.topic* items which will not be routed. Together with the white list file it is used for selective routing filtering. See std,std-refSelective Routing.

The value of this configuration property can be any file location as specified in std,std-refFiles locations.

link.whitelist.file

Valid values: *any string*

Default value: *not defined*

Purpose: Identifies a white list file location. This file contains a list of *partition.topic* items which will be routed. Together with black list file it is used for selective routing filtering. See std,std-refSelective Routing.

The value of this configuration property can be any file location as specified in std,std-refFiles locations.

link.ssl

Valid values: `true` , `false`

Default value: `false`

Purpose: When set to true all the TCP communications will be secured via a SSL (or TLS) protocol. See the other "ssl" configuration properties below to configure the SSL protocol.

link.ssl.algorithm

Valid values: *any string*

Default value: `TLS`

Purpose: Specifies SSL protocol to use. This must be a standard SSL protocol name. See the SSLContext section in the Java Cryptography Architecture Standard Algorithm Name Documentation for information about standard protocol names.

link.ssl.keystore.file

Valid values: *any string*

Default value: *not defined*

Purpose: Indicates the keystore file providing credential (private/public keys and certificates) to be used for encryption and authentication. See the Java Secure Socket Extension Reference Guide for more information.

The value of this configuration property can be any file location as specified in std,std-refFiles locations.

link.ssl.keystore.format

Valid values: *any string*

Default value: `JKS`

Purpose: Indicates the type of the keystore file.

See the KeyStore section in the Java Cryptography Architecture Standard Algorithm Name Documentation for information about standard keystore types.

link.ssl.keystore.password

Valid values: *any string*

Default value: *not defined*

Purpose: Indicates the password of the keystore file.

link.ssl.keymanager.algorithm

Valid values: *any string*

Default value: `PKIX`

Purpose: Specifies the name of the requested SSL Key Manager algorithm. This must be a standard algorithm name.

See the Java Secure Socket Extension Reference Guide for information about standard algorithm names.

link.ssl.truststore.file

Valid values: *any string*

Default value: *not defined*

Purpose: Indicates the truststore file providing trusted certificates which are used to verify the identity of remote entities. See the Java Secure Socket Extension Reference Guide for more information. Note that if this property is not defined and a keystore file is defined, the keystore will be used as truststore.

The value of this configuration property can be any file location as specified in std,std-refFiles locations.

link.ssl.truststore.format

Valid values: *any string*

Default value: `JKS`

Purpose: Indicates the type of the truststore file.

See the KeyStore section in the Java Cryptography Architecture Standard Algorithm Name Documentation for information about standard keystore types.

link.ssl.truststore.password

Valid values: *any string*

Default value: *not defined*

Purpose: Indicates the password of the truststore file.

link.ssl.trustmanager.algorithm

Valid values: *any string*

Default value: `PKIX`

Purpose: Specifies the name of the requested SSL Trust Manager algorithm. This must be a standard algorithm name.

See the Java Secure Socket Extension Reference Guide for information about standard algorithm names.

link.ssl.crl.file

Valid values: *any string*

Default value: *not defined*

Purpose: Indicates the CRL (Certificate Revocation List) file providing the list of revoked certificates. This file can be PEM-encoded or DER-encoded.

The value of this configuration property can be any file location as specified in std,std-refFiles locations.

Files locations

Some configuration properties define a file location (*e.g.* link.blacklist.file). Vortex Link accepts the file location to be specifeid as a path (absolute or relative), a Java resource or an URL. It will try to open the file treating the location parameter in the following order:

- as a path to a file on local file system:
  - as an absolute path
  - as a relative path from *"user.dir"* system property (*i.e.* working directory)
  - as a relative path from *"user.home"* system property (*i.e.* home directory)
- as a Java resource:
  - loaded by the ClassLoader which loaded Link
  - loaded by the SystemClassLoader (see `java.lang.ClassLoader.getSystemClassLoader()`)
- as an `java.net.URL` accepting the following protocols:
  - http, https, ftp, file and jar

- any other protocol if the `java.net.URLStreamHandlerFactory` was previously set *via* the `java.net.URL.setURLStreamHandlerFactory(java.net.URLStreamHandlerFactory)` operation.

Vortex Link service advanced configuration

The Vortex Link service is implemented on top of the Vortex Café ddsi stack. For performance or interoperability reasons, it may be necessary to fine-tune the ddsi stack. So, if needed, any Vortex Café ddsi configuration property (properties that start with *ddsi*) can be set to the Vortex Link service. For more details about Vortex Café ddsi configuration properties, please refer to the Vortex Café documentation.

Note that some of the Vortex Link configuration properties directly impact the ddsi configuration. For example, the Vortex Link link.tcp.port property impacts the Vortex Café *ddsi.discovery.tcp.port* property. For such configuration properties, if both Vortex Link and Vortex Café properties are set, the Vortex Café property will have precedence over the Vortex Link property and will apply.

5

Selective Routing

The selective routing functionality allows to define which kind of data (which topics and which partitions) should be routed or not by Vortex Link. This is defined with the help of a black list and/or a white list. Each list is defined in a different file containing a *partition.topic* combination on each line. *partition.topic* combinations may contain wild-card '*' characters.

Example :

```
partitionA.topicA
*.topicB
partitionB.*
A*.C
```

Each Vortex Link service can be configured to point to a black list file and/or a white list file with the help of std,std-reflink.blacklist.file and std,std-reflink.whitelist.file configuration properties.

When a white list is defined, *partition.topic* combinations not present in the white list will not be routed by Vortex Link. When a black list is defined, *partition.topic* combinations present in the black list will not be routed by Vortex Link. When both white list and black list are defined, black list has precedence over white list: *partition.topic* combinations present in both lists will not be routed by Vortex Link.

Different black lists and white lists can be defined in each different Vortex Link service of the system. This means that some *partition.topic* combinations may be routed from/to some sub systems/applications but not from/to some other sub systems/applications in the system.

WARNING Partitions containing a wild-card '*' character are never blocked. So user entities configured with partitions containing a wild-card '*' character will always be able to send receive data on those partitions whatever is defined in the black and white lists.

Specific case of Vortex Link as an intermediate routing point

Note that Vortex Link only applies Selective Routing to the subsystem or devices it routes directly (i.e. only to DDS applications it discovered directly via UDP multicast or TCP). If Vortex Link is used as an intermediate routing point between 2 (or more) services, it wont apply Selective Routing to the data it routes between those services.

For instance, with an Indirect-LAN-to-LAN deployment as such:

Vortex Link applies the blacklisting of "topicA" only to DDS applications it directly discovered. But it still routes data on "topicA" between the two Vortex Link services deployed in subsystems.

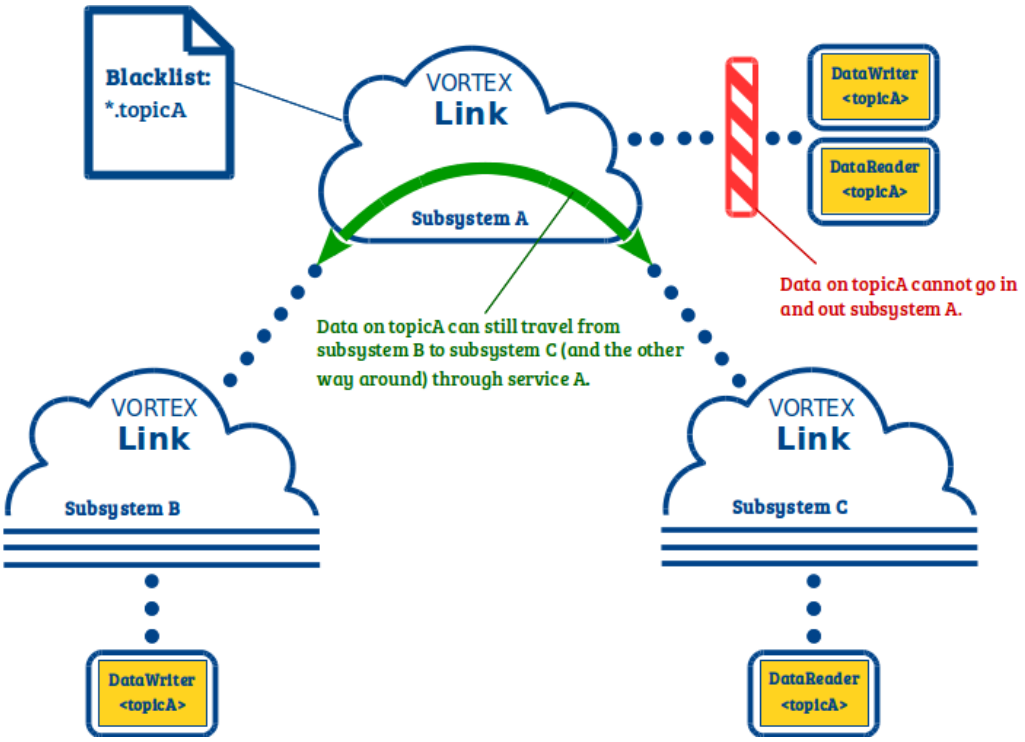


Fig. 5.1: **Selective Routing**

6

DDS-Security

Vortex Link is able to route secured traffic between applications using standard DDS Security. Vortex Link will not try to authenticate applications nor decode encrypted messages. It will simply route authentication messages, crypto tokens and encrypted messages between secured applications so that they can authenticate each other and exchange encrypted messages end-to-end.

So Vortex Link does not need to be configured with any certificate or security plugin and does not need to be trusted (it will not decode encrypted messages).

Limitations

- Vortex Link can only route data between participants configured with *allow_unauthenticated_participants* set to *false*.
- Vortex Link can only route data between participants configured with *rtps_protection_kind* set to *NONE*.
- If the participants are configured with *discovery_protection_kind* set to *ENCRYPT*, *SIGN*, *ENCRYPT_WITH_ORIGIN_AUTHENTICATION* or *SIGN_WITH_ORIGIN_AUTHENTICATION*, then Vortex Link can only route data on topics configured with *enable_discovery_protection* set to *false*.
- Vortex Link can route data on topics configured with any *data_protection_kind*.
- Vortex Link can route data on topics configured with any *metadata_protection_kind*. But, if *metadata_protection_kind* is set to *ENCRYPT* or *ENCRYPT_WITH_ORIGIN_AUTHENTICATION*, Vortex Link will forward encrypted messages to all participants that have matching entities with the participant that is source of the message (even if the matching entities are on a different topic than the topic of the encrypted meessage). This will lead to poor scalability and extra resources consumption.

7

Boundary Security

The Concept

Applying security mechanisms (authentication, cryptography, access-control) end-to-end between each single participant in the system may be very complex and not necessarily useful.

“Boundary Security” allows to apply such security mechanisms at key points in the system: at the edge (the boundary) of each subsystem and between subsystems. The communications inside each subsystem will be unsecured but data flowing out and in each subsystem will be controled and communications between subsystems will be secured. Security mechanisms will also apply on each device deployed in the WAN (outside any subsystem) and connected directly to the cloud.

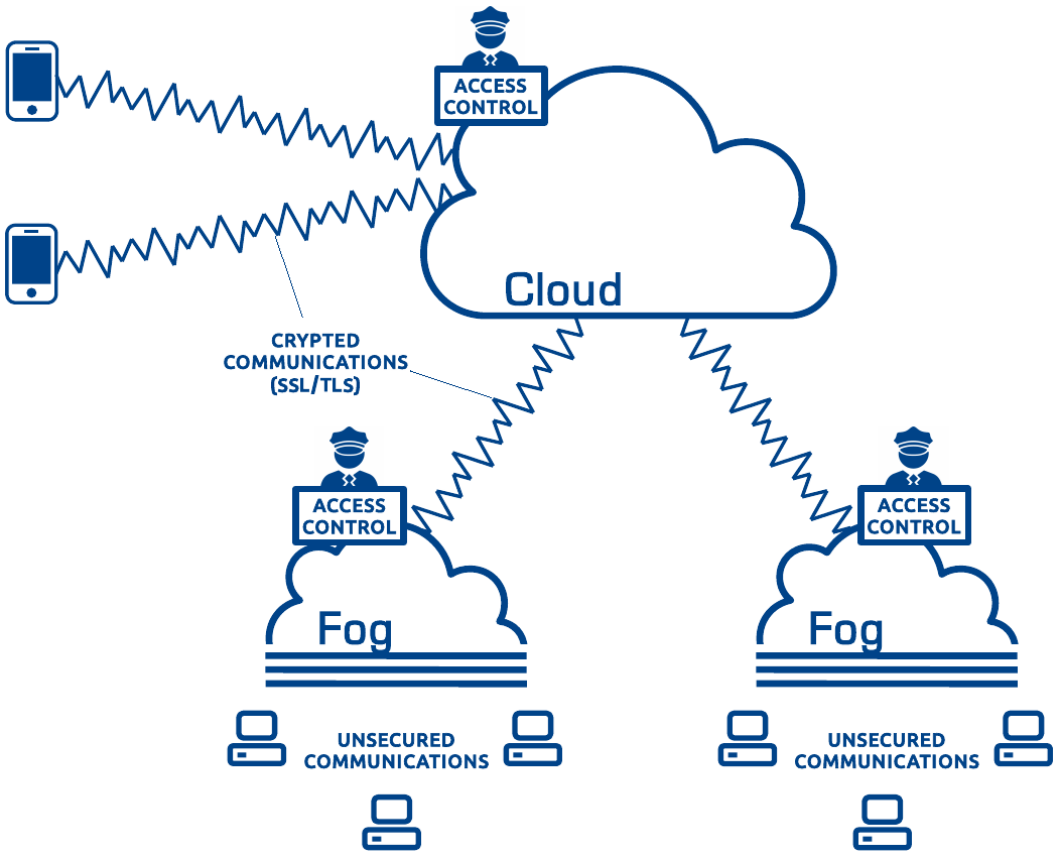


Fig. 7.1: **Boundary Security**

“Boundary Security” can be used with any kind of deployments (Device to Device, Lan to Lan, *etc.*) described in this documentation.

Authentication : Even if communications are unsecured inside each subsystem, each subsystem should authenticate itself to participate to the whole system. Each device using TCP transport should also authenticate. This

authentication will be performed with the help of SSL/TLS and X.509 certificates.

Cryptography : Communications between each subsystems, between subsystems and cloud and between external devices and cloud/subsystems should by crypted. This encryption will also be performed with the help of SSL/TLS.

Access-Control : Access-control rules, defined in a XML file, will be applied by each Vortex Link services and rely on identifiers provided by the SSL/TLS authentication so that different permissions are granted to each subsystem and device. For example, you may forbid a subsystem to access some particular topics or partitions from the rest of the system, or forbid data from some particular topics or partitions to get outside a particular subsystem.

Authentication and Cryptography Configuration

Authentication and cryptography are both provided by SSL/TLS over TCP connections. Thus, to configure authentication and cryptography you will have to generate some pairs of public/private keys and some X.509 certificates and to configure Vortex Link services and devices with those keys and certificates. The keys and certificates will be packaged into keystore files.

Vortex Link support the following KeyStore types:

- **jceks** The proprietary keystore implementation provided by the SunJCE provider.
- **jks** The proprietary keystore implementation provided by the SUN provider.

All the Vortex products use a mutual authentication to establish a TCP connection with SSL/TLS. This means that each DDS device and Vortex Link service must also have a list of trusted certificates (a truststore) that are accepted for TCP connection establishments.

As a truststore is actually a keystore containing only certificates, Vortex Link support the same types than keystore for truststore files.

PKIX is used as default algorithm for both keymanager and trustmanager.

To configure your devices to use TCP with SSL/TLS, please refer to the User Guide of the DDS product used by your device application.

Certificates and public/private keys organization

There are several ways to generate and manage certificates :

- You may generate all certificates for services and devices by yourself (using the java 'keytool' for example), or using your own Public Key Infrastructure (PKI) tool.
- You may use the provided "KeyManagers" tool (see Managing a CA hierarchy with the KeysManager tool) to generate all certificates for services and devices.
- You may use the provided "KeyManagers" tool to generate services certificates and use your own certificates for devices updating the generated services keystores with the devices public keys (or devices certificate authority public key).

Whatever solution is used, the following requirements must be answered :

- Each process (application or service) using the TCP transport must be deployed with a certificate (public and private key pair). The same certificate can be used in several process, but a process (using TCP) cannot be deployed with no certificate.
- All the services certificates must be signed (directly or indirectly) by a certificate authority (let's name it *Services CA*). The public key of this *Services CA* must be found in the services truststores under the alias '**vortex-services-ca**'. Note that :
  - The *Services CA* must be kept secret. If a malicious application is able to sign it's certificate with the services certificate authority, then the security is compromised.

  - The *Services CA* must not be used to sign any device certificate. Otherwise, a DDS application using such certificate could bypass the security.

- Each device must be deployed with :
  - the public key of the *Services CA* in it's truststore.
- Each service must be deployed with :
  - the public key of the *Services CA* in it's truststore with the alias '**vortex-services-ca**'.
  - the public key of all devices certificates or the public key of the certificate authority that signed the devices certificates if a certificate authority is used for devices.

The advised way to generate certificates is to generate a distinct public/private keys pair and certificate for each DDS device and each Vortex Link services.

- This allows granting of different access-control rights to each individual device and subsytem.
- It is also possible to grant the same access-control rights to a set of devices or a set of subsystems with the help of certificate subject names matching (see Access-Control Configuration).
- If a certificate is compromised, there is no need to re-deploy the new certificate to a bunch of devices/subsystems.
- It may become very difficult to manage all the certificates when the system gets bigger.

Another approach that minimizes the number of certificates to manage is to generate a distinct public/private keys pair and certificate for each level of permissions you want to apply in the system. (For example if 1000 devices in the system should be granted with the same access-control rights, then a single public/private keys pair and certificate can be used by those 1000 devices).

- If a certificate is compromised, a new certificate needs to be deployed on all the devices using it.

All the Vortex products accept self-signed certificates. However, using a certificate authority (CA) to sign devices certificates is advised in order to simplify the configuration of the truststores. They would not need to contain the certificates of all the devices that are trusted, but only a root CA or some intermediate CA certificates.

Managing a CA hierarchy with the KeysManager tool

The main disadvantage of self-signed certificates is that each time a new certificate is added to the system, it has to be added to one or more truststores. Using certificates signed by a CA allows to only have the CA certificate in the truststores. Each new certificate signed with this CA will be accepted as soon as the CA is in the truststore.

The KeysManager tool is a java graphical tool delivered with Vortex Link that helps to create a Certificate Authority hierarchy for a Vortex Link system. It can be used to generate keystore files that can directly be used by services and devices:

- a keystore for the Vortex Link service
- a keystore for each device (i.e. a DDS application using TCP/SSL); each device can be:
  - either restricted in its connections to 1 Link service
  - either unrestricted (i.e. it can establish TCP/SSL connection to any Link service)

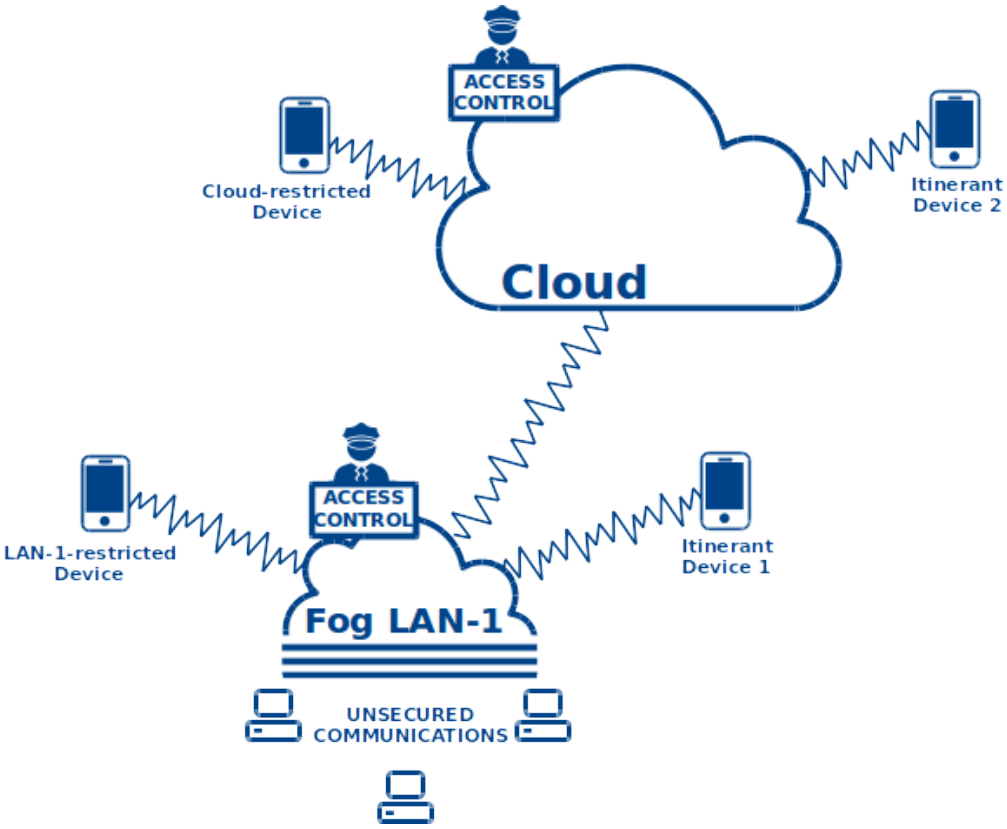
Note that KeysManager requires OpenSSL 1.0.1 to revoke certificates or export them to pem format (for Vortex OpenSplice or Vortex Lite).

The CA hierarchy

The KeysManager creates a CA hierarchy that aims to simplify the management of truststores. For this purpose, we consider 2 classes of process with different authorization of connection to the system:

- **the devices:** user DDS applications using TCP/SSL.
- **the services:** the Vortex Link services.

For instance, in case of such system:



the CA hierarchy will look like this:

For each service and device, the tool will generate a single keystore file (JKS or PEM). Each keystore file contains:

- The private/public key pair and associated certificate the service/device should use to authenticate.
- The certificates the service/device should trust.

With this hierarchy, once a service trusts the servicesCA, it will trust all other services, and once a service trusts the devicesCA, it will trust all unrestricted devices.

For a restricted device, as its private key is signed by the CA of its authorized service (Link), it can only connect to this service. The other services will refuse its TCP connection.

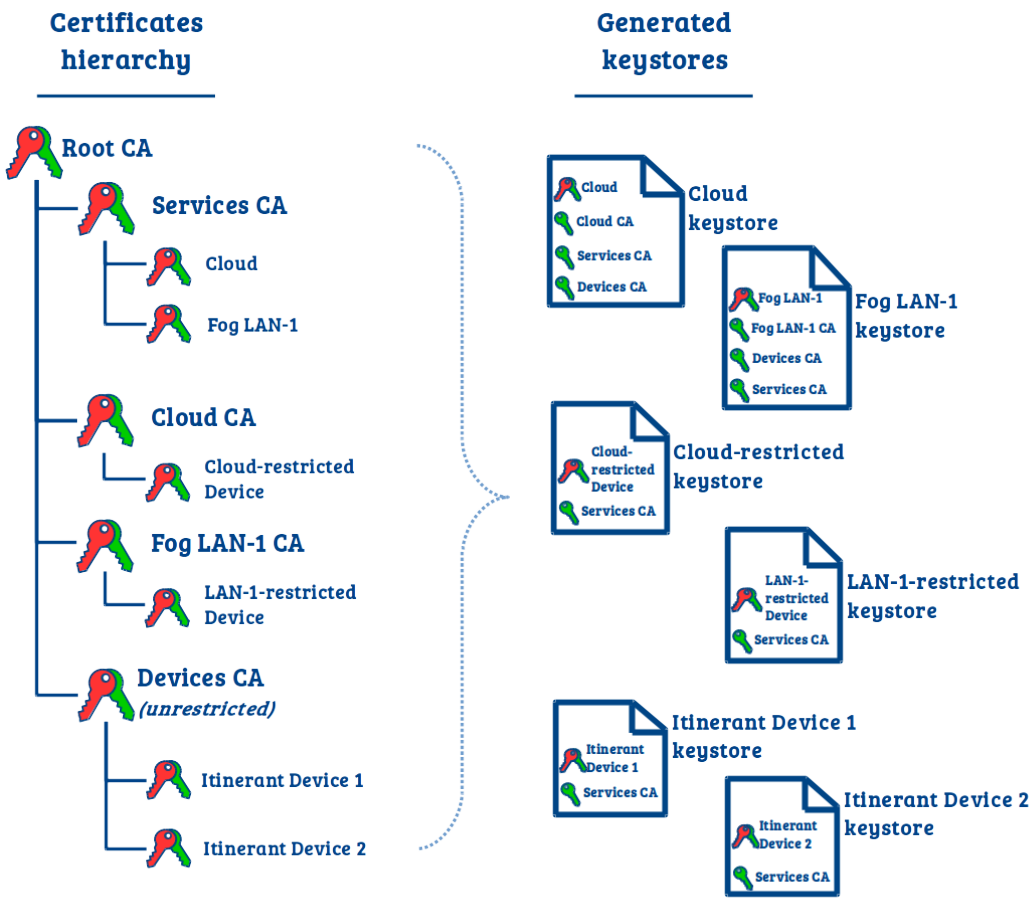


Fig. 7.2: **CA Hierarchy**

KeysManager tool usage

Running the KeysManager tool

The tool is delivered as a single executable jar file named `keys-manager.jar`. You can simply run it by double-clicking on it (on most OS) or by running the following command:

```
java -jar keys-manager.jar
```

Creating a new certificates hierarchy

First, you need to create a new certificate hierarchy. To do so open the *File* menu and select *Create new hierarchy*. You will then have to enter a set of informations and select a new file to save the hierarchy.

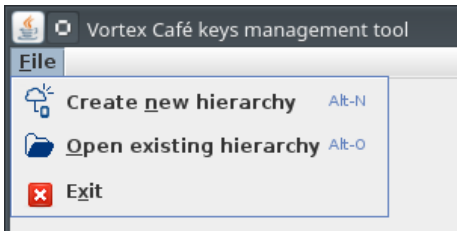


Fig. 7.3: **The keysmanager File menu**

After a few seconds, the newly created hierarchy will be displayed.

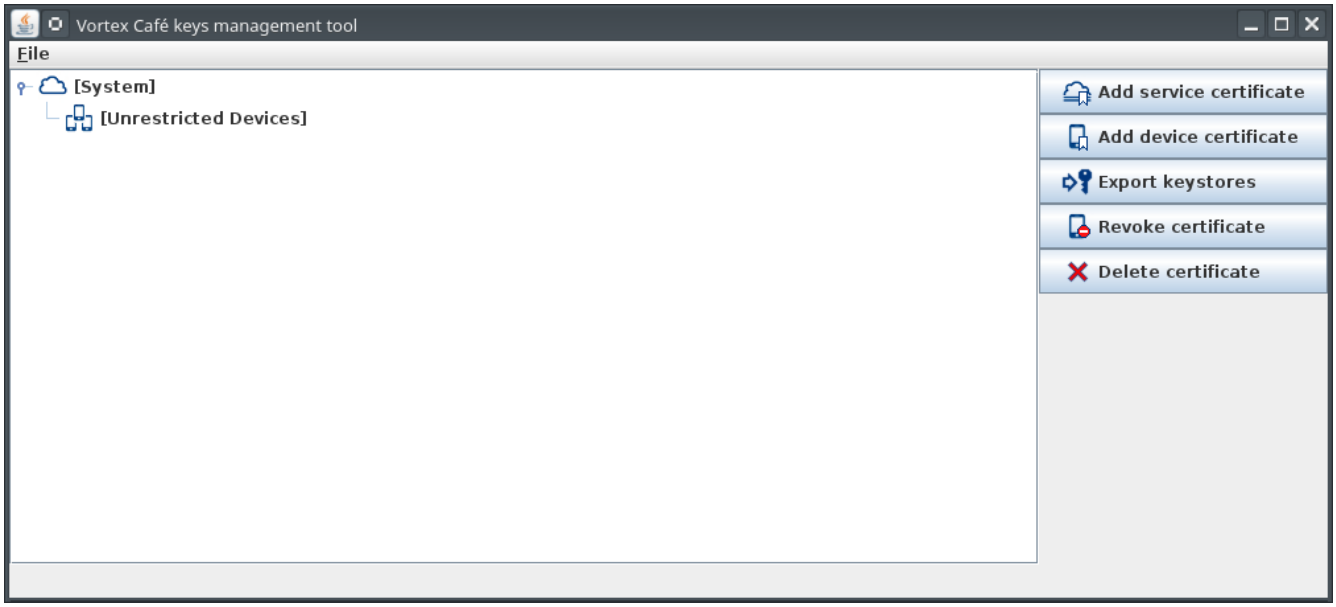


Fig. 7.4: **The keysmanager main window after creation of the CA hierarchy**

You can reload and update any hierarchy later using the *File* > *Open existing hierarchy* menu.

Creating a new certificate

Then you can add to this herarchy:

- A new service certificate
- A new device certificate (for a single device or a set of devices with the same rights)

To do so, click the *Add service certificate* or the *Add device certificate* button.

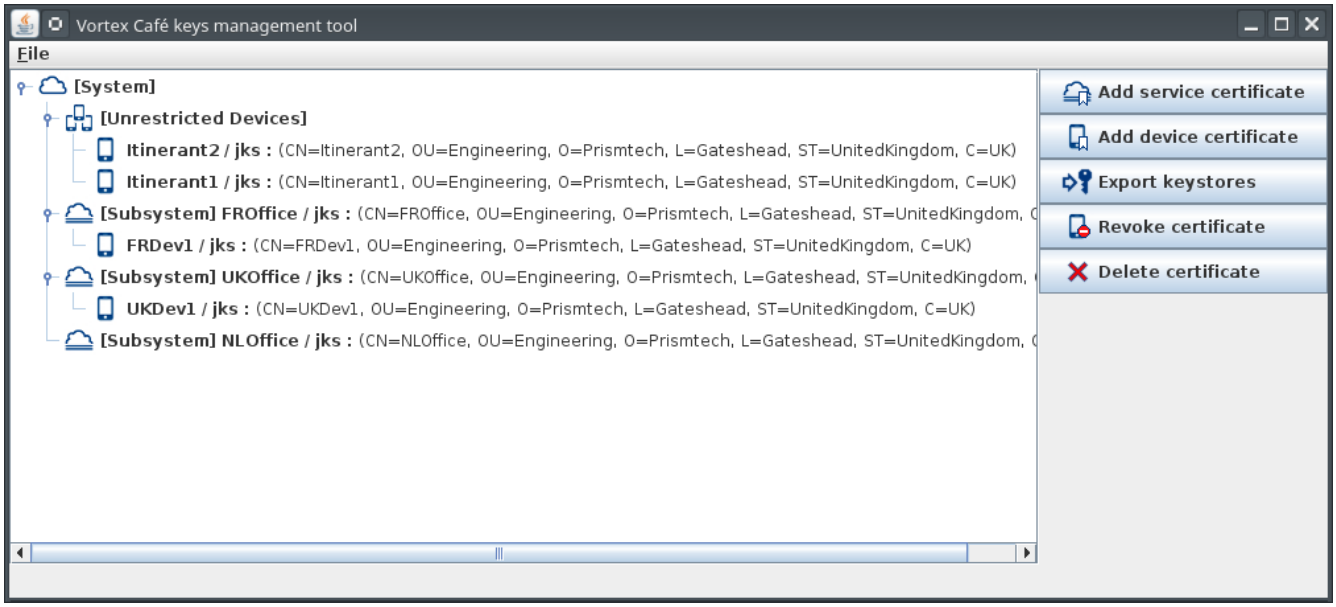


Fig. 7.5: **The keysmanager main window after addition of some certificates**

When creating a device certificate you can choose the export format. It can be jks format for Vortex Café product or pem format for Vortex OpenSplice and Vortex Lite products.

You can create two kinds of devices:

- **Unrestricted**: which can be used within any subsystem.
- **Restricted**: which is restricted to a specific subsystem. When choosing this kind of devices you have to select one of the existing subsystems.

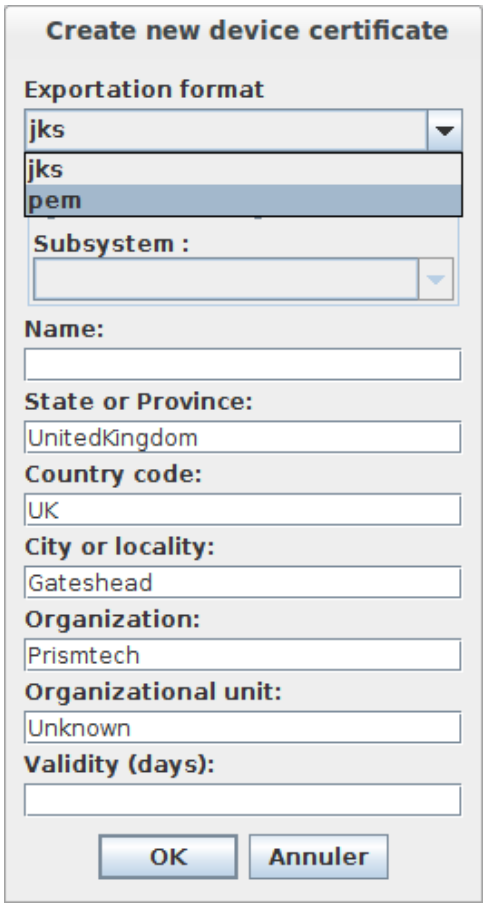


Fig. 7.6: **Exportation format option in the certificate creation window**

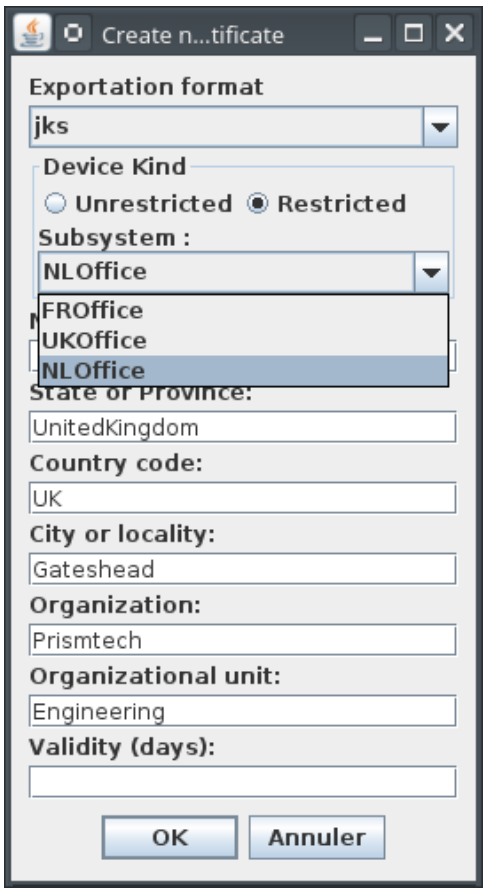
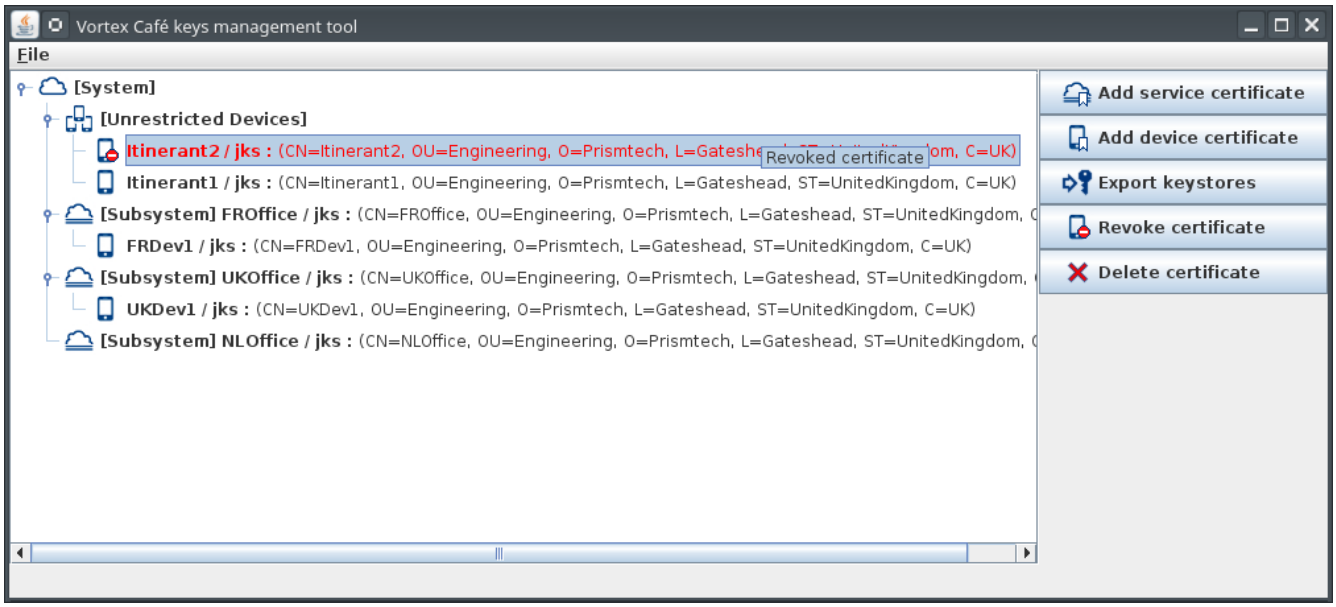


Fig. 7.7: **Kind of the new device**

Certificates revocation

You can choose a certificate to revoke and click on *Revoke certificate*.



A CRL (certificate revocation list) file will be generated *crl.pem* and exported to the folder that you will choose in the next section.

Exporting keystores and CRL

Finally, you can export Java KeyStores and CRL using the *Export keystores* button. This will create the following files in the folder of your choice:

- a JKS store for each service
- a JKS store for each device
- CRL: *crl.pem* if you have revoked any certificate.

For an easier configuration of the Vortex services and devices, the tool creates a single keystore file for each service/device. This single store contains the pair of public/private keys and certificate to be used by the service or device to authenticate, but also the trusted intermediate CA certificate. Therefore, it can be used as both keystore and truststore in the configuration of a service or device.

Configuring Link services

First, you need to activate SSL :

```
link.ssl=true
```

You also need to configure Vortex Link with the keystore generated by the tool for this subsystem or this level of permissions (When the same file is used as both keystore and truststore, only the keystore options need to be defined. The file specified for keystore will also be used as truststore):

```
link.ssl.keystore.file="mysubsystem.jks"
link.ssl.keystore.password="password"
```

Configuring Devices

You need to activate SSL and configure it with the keystore generated by the tool for this device or this level of permissions.

For a device using Vortex Café, the configuration properties would be:

```
ddsi.security.ssl=true
ddsi.security.ssl.keystore.file="/mydevice.jks"
ddsi.security.ssl.keystore.password="password"
```

To configure a device using another DDS product, please refer to this product's documentation.

Access-Control Configuration

Access-control rules are defined in an XML file that should respect the `boundary_security_permissions.xsd` schema file.

A typical permissions file will contain a **permissions** section itself containing several **grant** sections. Each *grant* section will contain:

- a **subject_name** or a **cluster_subject_name** section identifying the *DomainParticipant(s)* or the *cluster(s)* on which the grant should apply.
  - when **subject_name** is used, the grant will apply to all *DomainParticipants* configured with a certificate with a subject_name that matches the one specified in the *subject_name* section.
  - when **cluster_subject_name** is used, the grant will apply to all *DomainParticipants* deployed in a subsystem where the Vortex Link service is configured with a certificate with a *subject_name* that matches the one specified in the *cluster_subject_name* section.

    A single grant may apply to several devices/subsystems by providing an partial subject name that will match several certificate subject names.

    For example:

    ```
    ,O=ADLINK
    ```

    Will match all the certificates defined above:

    ```
    CN=Unknown,OU=Sales,O=ADLINK,L=Unknown,ST=Unknown,C=Unknown
    CN=Unknown,OU=Engineers,O=ADLINK,L=Unknown,ST=Unknown,C=Unknown
    CN=Unknown,OU=UKoffice,O=ADLINK,L=Gateshead,ST=UnitedKingdom,C=UK
    CN=Unknown,OU=FRoffice,O=ADLINK,L=Orsay,ST=France,C=FR
    CN=Unknown,OU=LinkUser,O=ADLINK,L=Unknown,ST=Unknown,C=Unknown
    ```

    Note that wildcards (the * character) may also be used:

    ```
    ,O=ADLINK,C=*,CN=Abc*
    ```

  - If a *DomainParticipant* has both a *subject_name* and a *cluster_subject_name* (TCP DomainParticipant deployed in a subsystem), and the permissions file contains both a grant that matches the *subject_name* and a grant that matches the *cluster_subject_name*, the grant that matches the *subject_name* will apply.
  - If a *DomainParticipant* matches several *cluster_subject_name* grants or several *subject_name* grants, then the first matching grant will be used. A warning trace will be logged by the Vortex Service.
- an optional *validity* section containing the validity time range of the grant.
- some **allow_rule** and/or **deny_rule** section(s). Each rule containing:
  - a **domains** section containing the domain id or set of domain ids on which the rule should apply.

  - some **publish** sections containing the list of topics and partitions on whish publication should be allowed or denied.
  - some **subscribe** sections containing the list of topics and partitions on whish subscription should be allowed or denied.

- a **default** section containing either *ALLOW* or *DENY* which specifies the default action to perform regarding partitions and topics that do not match any specified rules.

Example:

```
<permissions>

  <!-- Rules for all services or devices with OU=OrgUnit in their certificate -->
  <grant name="Grant1">
    <subject_name>,OU=OrgUnit</subject_name>
    <validity>
      <!--  Format is YYYY-MM-DDThh:mm::ss in GMT  -->
      <not_before>2014-03-01T13:00:00</not_before>
      <not_after>2500-03-01T13:00:00</not_after>
    </validity>
    <allow_rule>
      <!-- on domain 0 -->
      <domains>
        <id>0</id>
      </domains>
      <!-- allow subscription on all topics prefixed with "TopicName1" and all partitions -->
      <subscribe>
        <topics>
          <topic>TopicName1*</topic>
        </topics>
        <partitions>
          <partition>*</partition>
        </partitions>
      </subscribe>
      <!-- allow publication on topic "TopicName2" on default partition -->
      <publish>
        <topics>
          <topic>TopicName2</topic>
        </topics>
      </publish>
    </allow_rule>
    <!-- by default deny all other domains and topics -->
    <default>DENY</default>
  </grant>

  <!-- Rules for all sub systems with OU=OrgUnit in their Vortex Link certificate -->
  <grant name="Grant2">
    <cluster_subject_name>,OU=AnotherOrgUnit</cluster_subject_name>
    <validity>
      <!--  Format is YYYY-MM-DDThh:mm::ss in GMT  -->
      <not_before>2014-03-01T13:00:00</not_before>
      <not_after>2500-03-01T13:00:00</not_after>
    </validity>
    <deny_rule>
      <!-- on domain 0 -->
      <domains>
        <id>0</id>
      </domains>
      <!-- deny publication on all topics prefixed with "TopicName3" and all partitions -->
      <publish>
        <topics>
          <topic>TopicName3*</topic>
        </topics>
```

```
        <partitions>
          <partition>*</partition>
        </partitions>
      </publish>
    </deny_rule>
    <!-- by default allow all other domains and topics -->
    <default>ALLOW</default>
  </grant>

</permissions>
```

To avoid security breaches, all Vortex Link services in the system must be configured with the same permissions xml file.

Vortex Link services should be configured with this file:

```
link.accesscontrol.file="permissions.xml"
```

Note that a URL to the permissions file can be specified (*e.g.* http, ftp, *etc.*).

Limitations

All Vortex Link services are communicating with each other to exchange discovery information and to perform routes management. By consequence each service must trust all other services in the system.

The implication of this is that the key material used by the services is very sensitive. If a malicious person get such key material, he could run a malicious service that could :

- Access any data flowing through it
- Publish some data on unauthorized topics
- Publish some fake discovery informations

In summary, all keys and certificates generated for services should only be given to trusted people so each Link service in the system can only be deployed by trusted people.

Note: such limitation does not apply on devices.

Example

In this example, we want to build a system for a company (ADLINK) with:

- **subsystems (offices) with different permissions:**
  - UK office
  - FR office
- **Employees connecting from outside with different level of permissions:**
  - Sales
  - Engineers

Note that we may create a more granular system by giving to each individual employee connecting from outside a single certificate and different permissions but we keep the system simple for the examples.

Create a new certificates hierarchy

Using the KeysManager tool:

- Create a new hierarchy

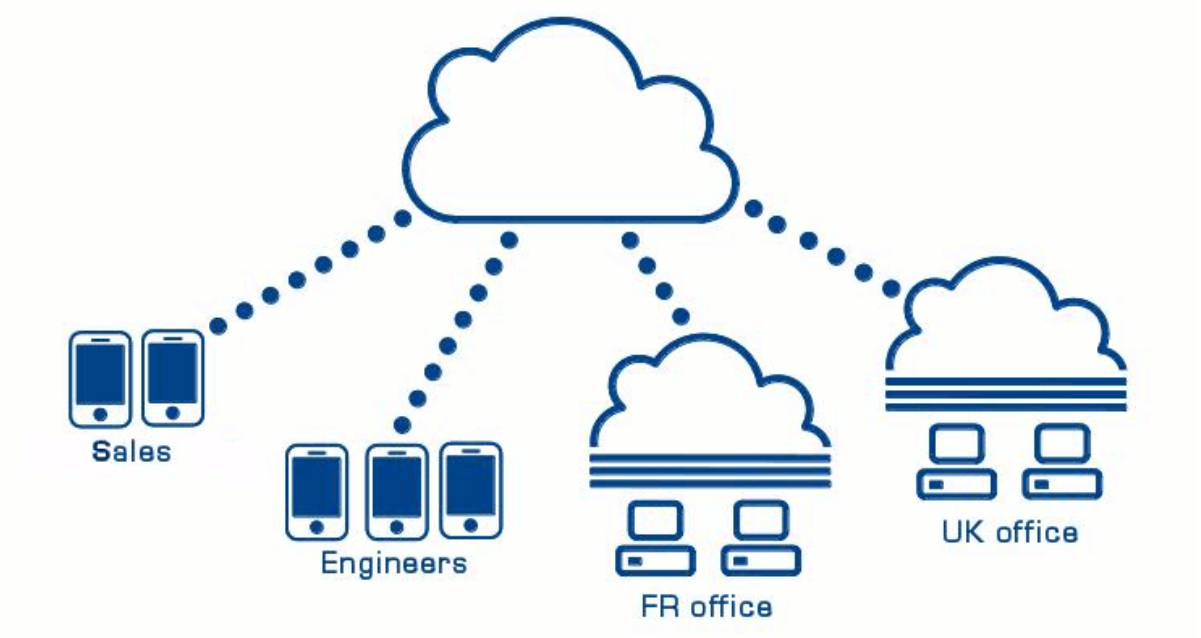


Fig. 7.8: **Boundary Security examples scenario**

- Add a new service certificate for the cloud service (Name=Link)
- Add a new service certificate with an identifyable subject name for UKOffice subsystem (ex: Name=UKOffice)
- Add a new service certificate with an identifyable subject name for FROffice subsystem (ex: Name=FROffice)
- Add a new device certificate with an identifyable subject name for sales (ex: Name=Sales)
- Add a new device certificate with an identifyable subject name for engineers (ex: Name=Engineers)
- Export certificates

This should generate 6 jks stores:

- **link.jks** to be used by the Vortex Link service deployed in the cloud
- **ukoffice.jks** to be used by the Vortex Link service deployed in UK office
- **froffice.jks** to be used by the Vortex Link service deployed in FR office
- **sales.jks** to be used by all Sales' devices
- **engineers.jks** to be used by all Engineer' devices

Configure Link services with the generated keystores

Link service (deployed in cloud) configuration

```
link.ssl=true
link.ssl.keystore.file=/link.jks
link.ssl.keystore.password=secret
link.accesscontrol.file=permissions.xml
```

UK office service configuration

```
link.ssl=true
link.ssl.keystore.file=/ukoffice.jks
```

```
link.ssl.keystore.password=secret
link.accesscontrol.file=permissions.xml
```

FR office service configuration

```
link.ssl=true
link.ssl.keystore.file=/froffice.jks
link.ssl.keystore.password=secret
link.accesscontrol.file=permissions.xml
```

Sales devices configuration (for Vortex Café)

```
ddsi.security.ssl=true
ddsi.security.ssl.keystore.file=/sales.jks
ddsi.security.ssl.keystore.password=secret
```

Engineers devices configuration (for Vortex Café)

```
ddsi.security.ssl=true
ddsi.security.ssl.keystore.file=/engineers.jks
ddsi.security.ssl.keystore.password=secret
```

Access-Control rules example

Here is an example of an Access-Control rules xml file that could be applied to this example:

```
<permissions>
    <grant name="UK office rules">
        <cluster_subject_name>CN=UKoffice</cluster_subject_name>
        <default>ALLOW</default>
    </grant>
    <grant name="FR office rules">
        <cluster_subject_name>CN=FRoffice</cluster_subject_name>
        <deny_rule>
            <domains>
                <id>0</id>
            </domains>
            <publish>
                <topics>
                    <topic>UkAccounting*</topic>
                </topics>
                <partitions>
                    <partition>*</partition>
                </partitions>
            </publish>
        </deny_rule>
        <default>ALLOW</default>
    </grant>
    <grant name="Sales rules">
        <subject_name>CN=Sales</subject_name>
        <allow_rule>
            <domains>
                <id>0</id>
            </domains>
            <publish>
                <topics>
                    <topic>Com*</topic>
                </topics>
            </publish>
            <subscribe>
                <topics>
                    <topic>Com*</topic>
```

```
                </topics>
            </subscribe>
        </allow_rule>
        <default>DENY</default>
    </grant>
    <grant name="Engineers rules">
        <subject_name>CN=Engineers</subject_name>
        <allow_rule>
            <domains>
                <id>0</id>
            </domains>
            <subscribe>
                <topics>
                    <topic>Repository</topic>
                </topics>
            </subscribe>
            <subscribe>
                <topics>
                    <topic>Jira</topic>
                </topics>
            </subscribe>
        </allow_rule>
        <default>DENY</default>
    </grant>
</permissions>
```

8

Load balancing & Fault tolerance

Redundancy

For both fault tolerance and load balancing purposes, we may need to deploy several Vortex Link services replicas in the subsystems. Indeed :

- We don't want the Vortex Link services to be single points of failure. So replicating the services allows the replicas to act as fallbacks if one of the services fails.
- We may need to balance the traffic load across several services and hosts.

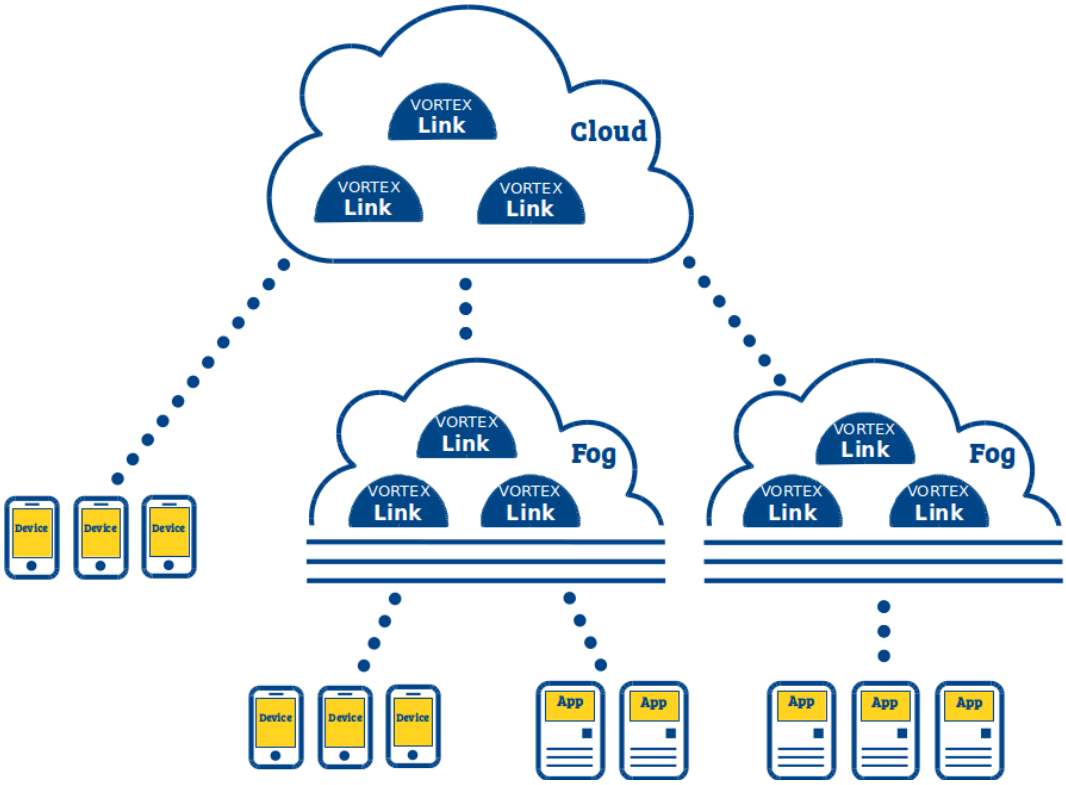


Fig. 8.1: **Services redundancy**

Clusters & Cluster ID

The set of replicated services deployed to serve a subsystem is called a cluster. Services deployed in a cluster need to behave differently than when deployed standalone. They also need to know which other services are part of the same cluster or not. Consequently each cluster must be identified in the system by a unique cluster id string.

By default a service will consider being standalone and not being part of any cluster. To activate redundancy and indicate to the service it is supposed to be in a cluster, the `link.cluster.id` property must be set. This property also indicates to the service to which cluster it belongs. So all services of a same cluster must be configured with the same cluster id.

Example:

```
link.cluster.id="PublicLink"
```

All services in a cluster must be configured with the same service level (see std,std-reflink.serviceLevel).

In a system, it is not mandatory to replicate all the services in all the subsystems. A "standalone" service deployed alone in a subsystem is able to interoperate with replicas deployed in other subsystems.

Note also that a "standalone" service offers a smallest discovery and route establishement time than a cluster service. So if you know that a service will not be redunded it is better to deploy it as "standalone" (i.e. not setting its cluster id).

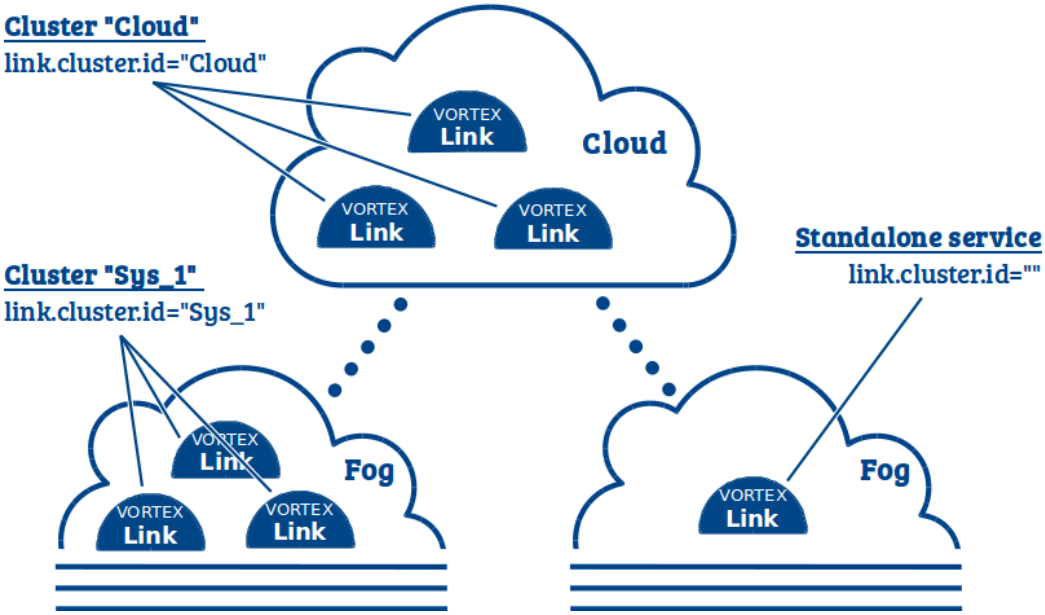


Fig. 8.2: **Clusters**

Automatic cluster id

It is possible to let Vortex Link services automatically determine which services belong to the same cluster. When automatic cluster id is activated, Vortex Link services will consider all services deployed in the same IP subnetwork to belong to the same cluster. To do so, Vortex Link services will use the broadcast address of the network interface it uses as cluster id. If both UDP and TCP transport are activated but configured to use different network interfaces, the UDP network interface will be used to determine the cluster id string.

For example, assume a Vortex service configured to use the following network interface:

```
eth0    Link encap:Ethernet  HWaddr 01:23:45:67:89:AB
        inet addr:192.168.2.2  Bcast:192.168.2.255  Mask:255.255.255.0
        inet6 addr: fe80::21d:92ff:fede:499b/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

If this service is also configured with automatic cluster id, it will use string `"192.168.2.255"` as cluster id.

To activate automatic cluster id, the `link.cluster.id` property must be set to value `"auto"`.

Example:

```
link.cluster.id="auto"
```

WARNING:

when you deploy different clusters on different LANs, you must check that those LANs don't have the same broadcast address. If they have, you must not use the `"auto"` value for cluster ids.

Similarly, if you want to test the deployment of different clusters in a same LAN, you must not use the `"auto"` value for cluster ids.

Load balancing plugins

A lot of different policies could be applied to balance the several data flows across the several service replicas of a cluster. In some use cases, some policies may be more appropriate but in other cases other policies may be more optimal. So the load balancing decisions are made by plugins. The `link.loadBalancing.pluginClass` or `link.loadBalancing.pluginClass` property can be used to configure which plugin implementation should be used by the Vortex service. This property must be set to the fully qualified class name of the plugin implementation that should be used. All services of a same cluster must be configured to use the same plugin implementation.

Example:

```
link.loadBalancing.pluginClass=vortex.lb.plugins.PerParticipantHashPlugin
```

Vortex Link services are delivered with two plugin implementations :

- PerParticipantHashPlugin
- PerWriterHashPlugin

By default, Vortex Link services will use the PerParticipantHashPlugin.

PerParticipantHashPlugin

Fully qualified name : `vortex.lb.plugins.PerParticipantHashPlugin`

This is the default plugin. It will make all data coming from the same user Participant (thus typically the same application) to be routed by the same replica in the cluster. The election of the replica that will route all the data flow from a Participant is made by a Rendezvous Hashing algorithm, based on the Participant identifier (GUID).

Benefits and drawbacks of this plugin:

- When user applications use TCP transport, this implementation induce less TCP connections than the PerWriterHashPlugin, so less resource consumption on both application and services side.
- This implementation offers a less optimal balancing than the PerWriterHashPlugin.

PerWriterHashPlugin

Fully qualified name : `vortex.lb.plugins.PerWriterHashPlugin`

This plugin will make all data coming from the same user DataWriter to be routed by the same replica. The election of the replica that will route all the data flow from a DataWriter is made by a Rendezvous Hashing algorithm, based on the DataWriter identifier (GUID).

Benefits and drawbacks of this plugin:

- This implementation offers a better balancing than the PerParticipantHashPlugin.
- When user applications use TCP transport, this implementation may induce more TCP connections than the PerParticipantHashPlugin, so more resource consumption on both application and services side.

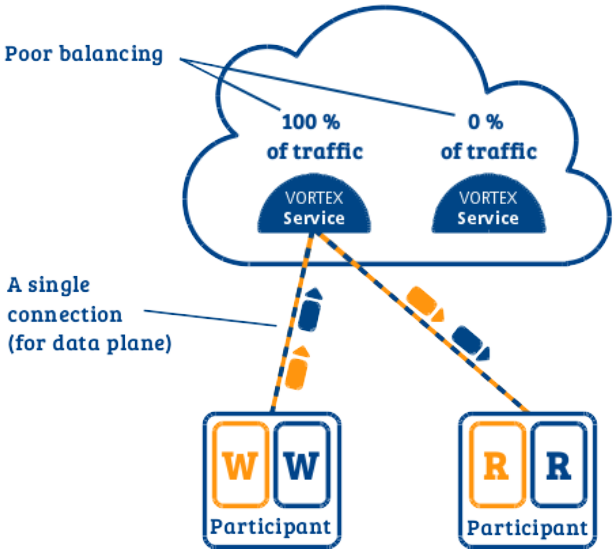


Fig. 8.3: **PerParticipantHashPlugin**

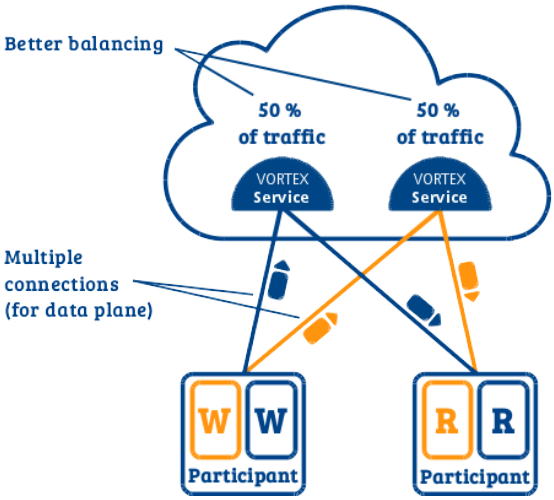


Fig. 8.4: **PerWriterHashPlugin**

Fault tolerance

When a Vortex Link service fails in a cluster (crashes, is terminated by admin, ...), all the data flows that were routed by this service in the cluster will be rebalanced to other services of the cluster.

This switch over will take some time (see Optimizing fault detection time). During this time :

- On best-effort communications, all samples emitted during this switch over will be lost and so not received by the concerned DataReaders.
- On reliable communications, no sample will be lost and the concerned readers will receive all samples after the switch over. The DataWriter may block in its write operation until the switch over is finished (depending if its reliability queue is full).

Peers configuration for services

As described in std,std-refHow to deploy and configure Vortex Link? : when services use unicast communications to communicate with each other, they need to be configured with initial peers. When using fault tolerance, we want the system to work even if some services fail. So we need to make sure that:

- The services can connect to the remote clusters even if some of the services of the cluster failed.
- Whenever a service is connected to one of the replicas of a remote cluster and if this replica fails, the service connects to another replica of the remote cluster.

For this purpose, we need to configure the services with all the locators of the replicas (or at least part of the replicas) of the cluster they should connect to, so that if some of the replicas fail, the service can still connect to one of them. As we don't necessarily want the service to connect to all of the replicas of the remote cluster at the same time, we can use locator groups (see std,std-refLocators group).

Example of connection of Vortex Link to a group of 2 replicas on different hosts:

```
-Dlink.tcp.peers=[65.65.65.65:7400,80.80.80.80:7400]
```

Peers configuration for applications

As described in std,std-refHow to deploy and configure Vortex Link? : when applications use unicast communications to communicate with Vortex Link services, they need to be configured with initial peers. When using fault tolerance, we want the system to work even if some services fail. So we need to make sure that:

- The application can connect to a cluster even if some of the services of the cluster failed.
- Whenever an application is connected to one of the replicas of a cluster and if this replica fails, the application connects to another replica of the cluster.

For this purpose, we need to configure the applications with all the locators of the replicas (or at least part of the replicas) of the cluster they should connect to so that, if some of the replicas failed, the service can still connect to one of them. As we don't necessarily want the application to connect to all of the replicas of the cluster at the same time, we can use locator groups (see std,std-refLocators group).

Example with Vortex Café connecting to a group of 2 replicas on different hosts:

```
-Ddds.discovery.tcp.peers=[65.65.65.65:7400,80.80.80.80:7400]
```

Optimizing fault detection time

During a switch over, what usually takes most time is the fault detection is the time other services of the cluster take to detect that one service in the cluster has failed. This fault detection time can be minimized by configuring the Vortex Link services lease duration with the help of the following properties:

- `ddsi.discovery.participant.leaseDuration` (time in seconds, default 10 seconds)

  This property configures the amount of time other services should wait when they do not receive any message from a given service before considering the service as dead. The smallest this duration is, the fastest the switch over will be.

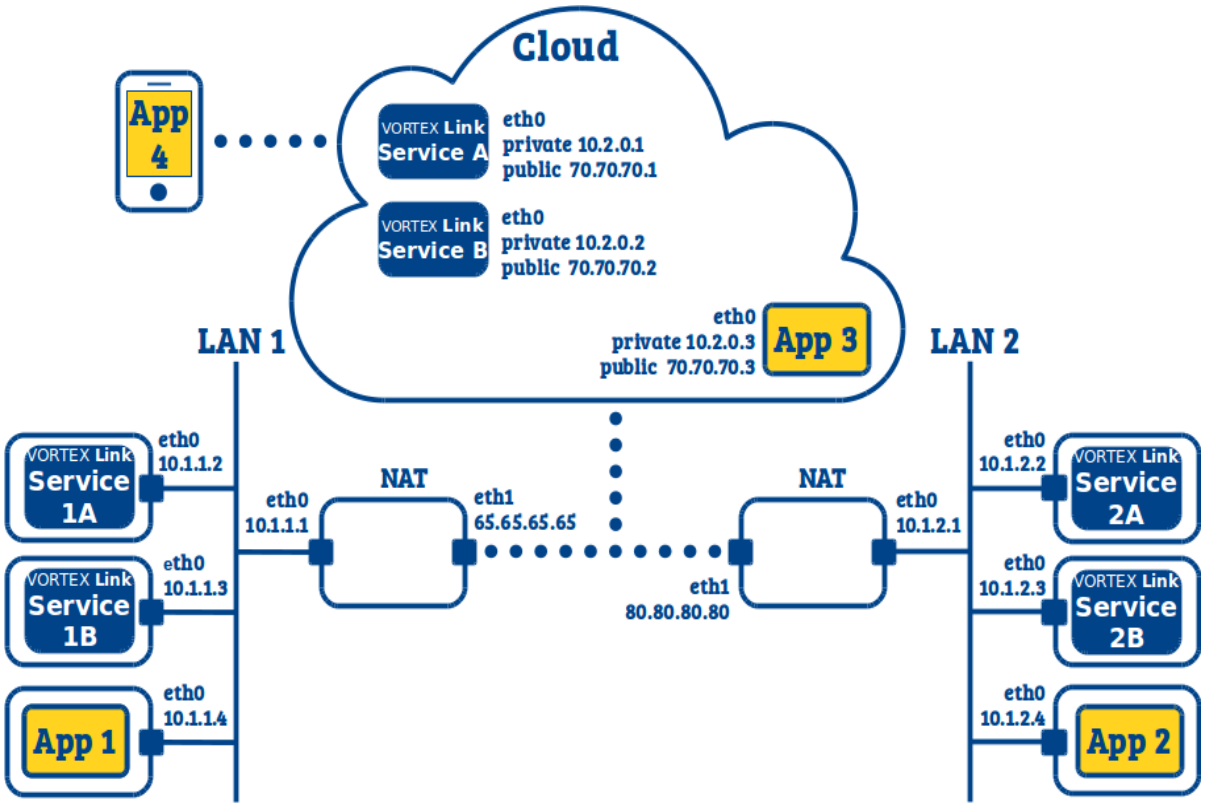
- `ddsi.discovery.participant.advertisePeriod` (time in seconds, default 2.5 seconds)

  This property configures the period used by services to advertise themselves to other services or applications. This property must be set with a duration at least 2 or 3 times shorter than the duration configured in `ddsi.discovery.participant.leaseDuration`.

For more details about those configuration properties, please refer to the Vortex Café documentation.

Fault tolerance configuration examples

Indirect LAN to LAN



Vortex Link service A:

```
link.cluster.id="Link"
link.network.interface=eth0
link.externalNetworkAddresses=70.70.70.1
link.serviceLevel=20
```

Vortex Link service B:

```
link.cluster.id="Link"
link.network.interface=eth0
link.externalNetworkAddresses=70.70.70.2
link.serviceLevel=20
```

Vortex Link services 1A & 1B:

```
link.cluster.id="LAN1"
link.network.interface=eth0
link.tcp.peers=[70.70.70.1:7400,70.70.70.2:7400]
link.externalNetworkAddresses=none
```

Vortex Link services 2A & 2B:

```
link.cluster.id="LAN2"
link.network.interface=eth0
link.tcp.peers=[70.70.70.1:7400,70.70.70.2:7400]
link.externalNetworkAddresses=none
```

App 1 & 2:

(nothing to configure)

App 3:

(nothing to configure)

App 4 (using Vortex Café):

```
ddsi.network.udp.enabled=false
ddsi.discovery.tcp.peers=[70.70.70.1:7400,70.70.70.2:7400]
ddsi.discovery.tcp.port=7400
ddsi.discovery.externalNetworkAddresses=none
```

9

Logging

The Vortex Link logging system is based on **SL4J** (`http://www.slf4j.org`).

SL4J is an abstraction layer for various logging frameworks (such as java.util.logging, Log4J, Logback, *etc.*) allowing the end user to plug in the desired logging framework at deployment time.

The **Logback** framework is embedded in the Vortex Link service jar and so will be used by default. If you want to use any other logging framework, you need to add it in the *CLASSPATH* (before the Vortex Link service jar), as documented in `http://www.slf4j.org/manual.html#swapping`.

Using the Logback framework, you can change the log level with the `log.level` java property. This property accepts the following values:

- ALL
- DEBUG
- ERROR
- INFO
- OFF
- TRACE
- WARN

Example:

```
java -Dlog.level=OFF -jar link-service.jar
```

By default both Vortex Link services have Logback configured to produce log messages in the console (*i.e.* `stdout`). If you want to change this (for instance to have log messages in a file) you may define you own Logback configuration file and run the service with the following command:

```
java -Dlogback.configurationFile=<your_file> -jar link-service.jar
```

For more details on Logback configuration please refer to `http://logback.qos.ch/documentation.html`.

10

Locators

Locators & locator list

A *locator* describes a TCP or UDP endpoint. It is an ip-address/port couple. In the configuration, a single locator is expressed like this:

```
ip-address:port
```

Example:

```
10.1.1.1:7400
```

Configuring a service or an application, we may need to specify several remote locators. This is called a *locator list* and is expressed like this:

```
ip-address:port,ip-address:port,...
```

Example:

```
10.1.1.1:7400,10.1.1.2:7400
```

When configured with a list of locators, the service or the application will try to connect them all.

Locators group

A *locator group* defines the locators of a fault-tolerant group of applications. An application configured with a locator group will try to connect to one and only one of those locators. During its lifetime, if the locator of the group to which it is connected dies, it will try to reconnect to another locator of the group so that it maintains a connection to one and only one of the locators of the group.

This is useful when pointing end-user applications to Vortex Link services as we may don't want the application to be connected to all the services at the same time (unnecessary resources consumption), but we don't want the service to which it is connected to to be a single point of failure.

In the configuration, a locator group is expressed in enclosing square brackets like this:

```
[ip-address:port,ip-address:port,...]
```

Example:

```
[10.1.1.1:7400,10.1.1.2:7400]
```

We may want to configure an application with a *list of locators group*. In the configuration, such a list is expressed like this:

```
[ip-address:port,ip-address:port,...],
[ip-address:port,ip-address:port,...],...
```

Example:

```
[10.1.1.1:7400,10.1.1.2:7400],[10.1.10.1:7405,10.1.10.2:7405]
```

We may also use a list of groups and single locators:

```
[10.1.1.1:7400,10.1.1.2:7400],10.1.10.1:7405,10.1.10.2:7405
```

11

Detailed Examples

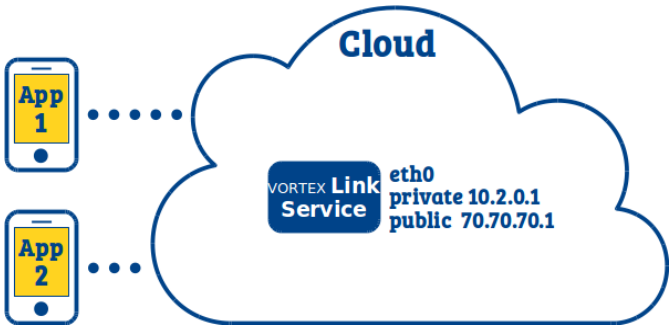
In this chapter we detail in several use cases the configuration of each Vortex Link service, but also the configuration of DDS applications.

Device to Device

In this use case we want to enable DDS discovery and communication between devices running DDS applications over TCP (*e.g.* mobile applications which are usually behind a symmetric NAT, and so have no public IP). For this we need to deploy a Vortex Link service in a cloud on a host that is publicly accessible. In this example, we will consider that the cloud host that will host the service has a private IP address (10.2.0.1) but is accessible through a public Ip address (70.70.70.1).

We will configure the Vortex Link service to:

- use the network interface `eth0` (as cloud hosts might be multi-homed)
- use the public IP of its host (`70.70.70.1`)
- listen for TCP connections on port 8500



Vortex Link service:

```
link.network.interfaces=eth0
link.externalNetworkAddresses=70.70.70.1
link.tcp.port=8500
```

App 1 and 2 (using Vortex Café):

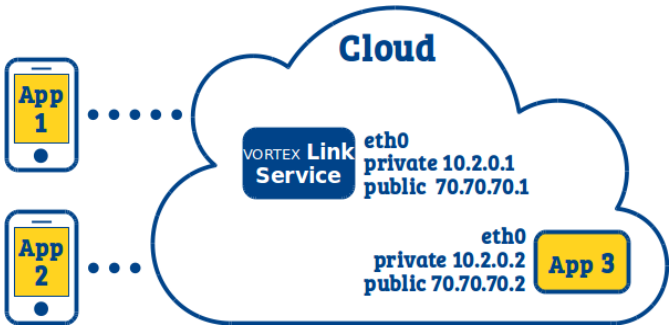
```
ddsi.network.udp.enabled=false
ddsi.discovery.tcp.peers=70.70.70.1:8500
ddsi.discovery.tcp.port=7400
ddsi.discovery.externalNetworkAddresses=none
```

Device to Device & Device to Link

In this use case we want to enable DDS discovery and communication between devices with DDS applications over TCP, but also DDS applications running in the cloud (over UDP multicast or TCP).

We will configure the Vortex Link service to:

- use the network interface `eth0` (as cloud hosts, might be multi-homed)
- use the public IP of its host (`70.70.70.1`)
- listen for TCP connections on port 8500
- discover UDP multicast applications on domain 0



Vortex Link service:

```
link.network.interfaces=eth0
link.externalNetworkAddresses=70.70.70.1
link.tcp.port=8500
link.domainid=0
```

App 1 and 2 (using Vortex Café):

```
ddsi.network.udp.enabled=false
ddsi.discovery.tcp.peers=70.70.70.1:8500
ddsi.discovery.tcp.port=7400
ddsi.discovery.externalNetworkAddresses=none
```

App 3 (using Vortex Café):

If using multicast:

Nothing particular to configure (use domainId 0 by default)

If using tcp:

```
ddsi.network.udp.enabled=false
ddsi.discovery.tcp.peers=10.2.0.1:8500
ddsi.discovery.tcp.port=7400
```

LAN to LAN (I can deploy on my Firewall/NAT)

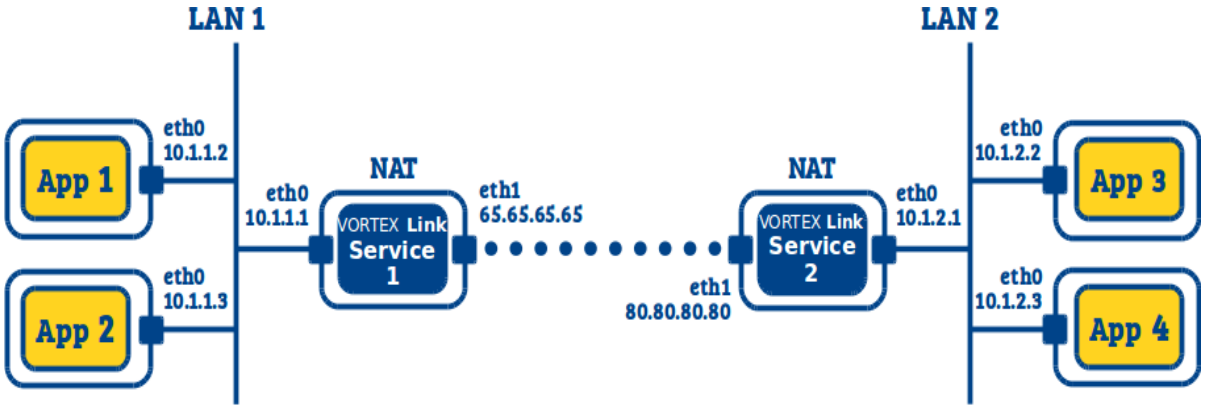
In this use case we want to interconnect two LANs where DDS applications are running using UDP multicast. For this we need to deploy a Vortex Link service in each LAN.

Each Vortex Link service will discover applications in its LAN, but also get informations about the applications discovered by the remote Vortex Link service in the other LAN. The Vortex Link services will then be able to establish a route between applications in distinct LAN.

In this use case we can deploy the services on the NAT host (*i.e.* a host with two network interfaces: one connected to the LAN with a private IP, and one connected to the WAN with a public IP).

We will configure the Vortex Link service in each LAN to:

- use the network interface connected to the LAN to discover UDP user applications and route traffic from user applications
- use the network interface connected to the WAN to route traffic to the remote Vortex Link service through TCP
- establish a TCP connection with the remote Vortex Link service in the other LAN



Vortex Link service 1:

```
link.udp.interface=eth0
link.tcp.interface=eth1
link.tcp.peers=80.80.80.80:7400
```

Vortex Link service 2:

```
link.udp.interface=eth0
link.tcp.interface=eth1
link.tcp.peers=65.65.65.65:7400
```

App 1, 2, 3, 4:

(nothing to configure)

LAN to LAN (I can't deploy on my Firewall/NAT but can configure it)

This use case is similar to the previous one (see LAN to LAN (I can deploy on my Firewall/NAT)) except that here we cannot deploy the Vortex Link services on the NAT hosts. Instead, we can configure each NAT to redirect the TCP port (default 7400) to a host in each LAN.

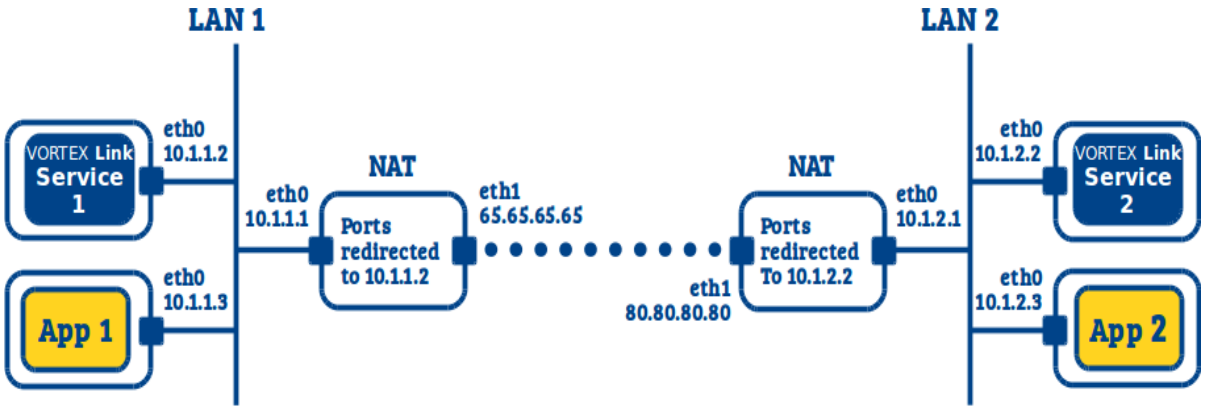
The only differences from the configurations of the previous use case are that the services are now using only one network interface, and that we need to specify the NAT public IP for each service using the *externalNetworkAddresses* property.

Vortex Link service 1:

```
link.network.interface=eth0
link.tcp.peers=80.80.80.80:7400
link.externalNetworkAddresses=65.65.65.65
```

Vortex Link service 2:

```
link.network.interface=eth0
link.tcp.peers=65.65.65.65:7400
link.externalNetworkAddresses=80.80.80.80
```



App 1, 2:

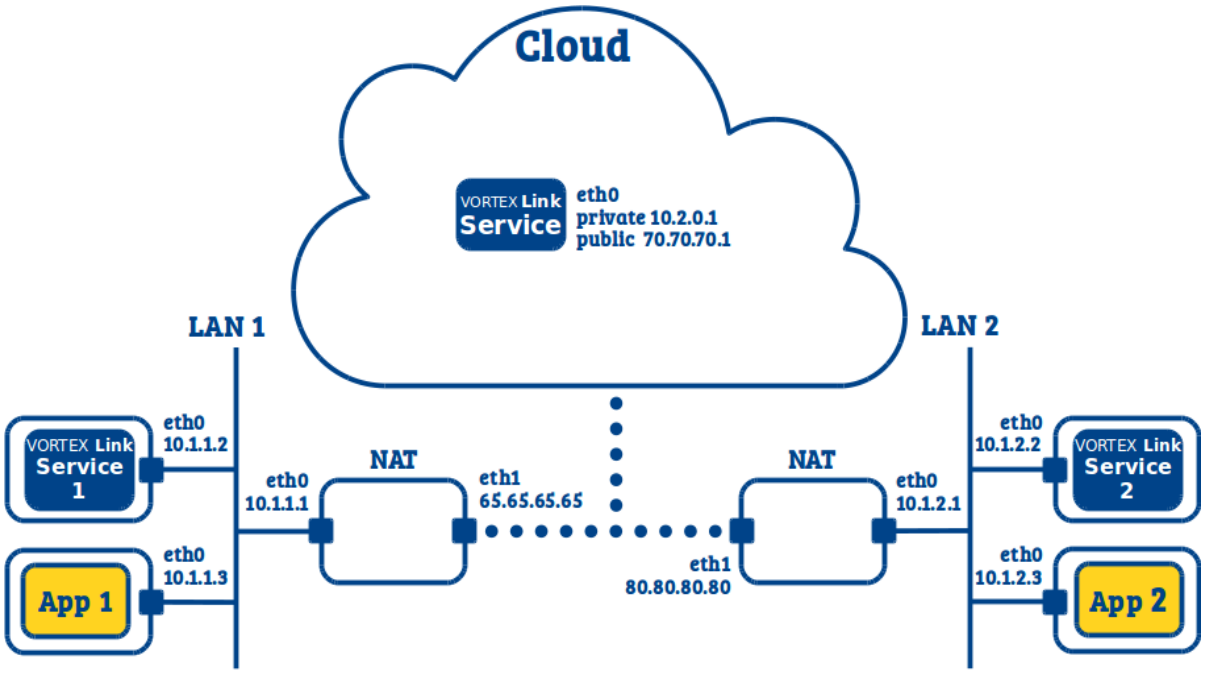
(nothing to configure)

Indirect LAN to LAN (I can't deploy on my Firewall/NAT and can't configure it)

This use case is similar to the previous ones (see LAN to LAN (I can deploy on my Firewall/NAT) and see LAN to LAN (I can't deploy on my Firewall/NAT but can configure it)), but here we can not deploy the Vortex Link services on the NAT hosts, and we can not configure ports redirection on the NAT. The consequence is that services deployed in each LAN cannot establish a direct connection with services in the remote LAN.

We therefore need to use a cloud as an intermediate, and to deploy a Vortex Link service within the cloud. This service will be in charge of routing communications between the Vortex Link services.

Note that this deployement is actually a hierarchical deployment, and thus Vortex Link must be configured with a higher service level than the Vortex Link services (which by default are configured with level 0). See std,std-reflink.serviceLevel configuration.



Vortex Link service:

```
link.network.interface=eth0
link.externalNetworkAddresses=70.70.70.1
link.serviceLevel=1
```

Vortex Link services 1 & 2:

```
link.network.interface=eth0
link.tcp.peers=70.70.70.1:7400
link.externalNetworkAddresses=none
```

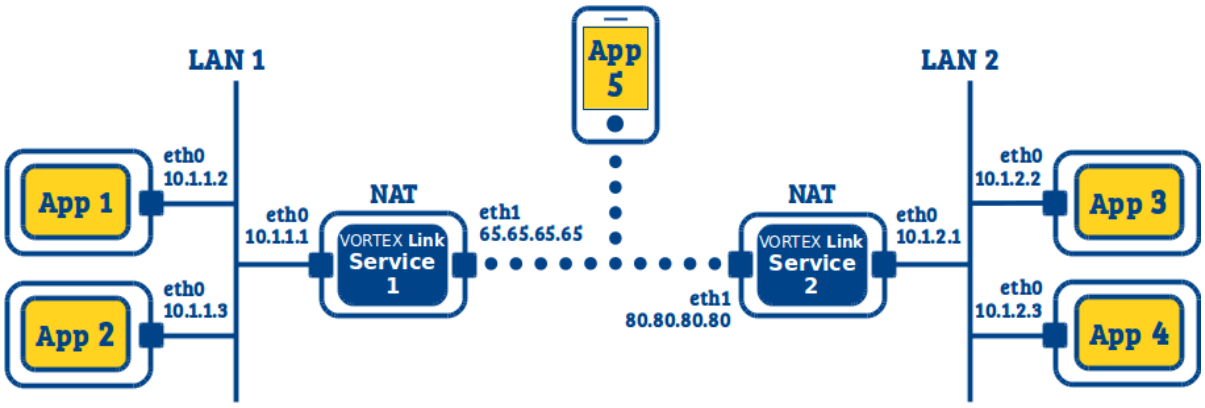
App 1, 2:

(nothing to configure)

LAN to LAN + Internet devices (no cloud)

This use case is similar to the one in LAN to LAN (I can deploy on my Firewall/NAT), but we also want to have external devices using DDS over TCP to communicate with DDS applications in each LAN.

The TCP applications must be configured to connect to one (or both) of the Vortex Link services.



Link 1:

```
link.udp.interface=eth0
link.tcp.interface=eth1
link.tcp.peers=80.80.80.80:7400
```

Link 2:

```
link.udp.interface=eth0
link.tcp.interface=eth1
link.tcp.peers=65.65.65.65:7400
```

App 1, 2, 3, 4:

(nothing to configure)

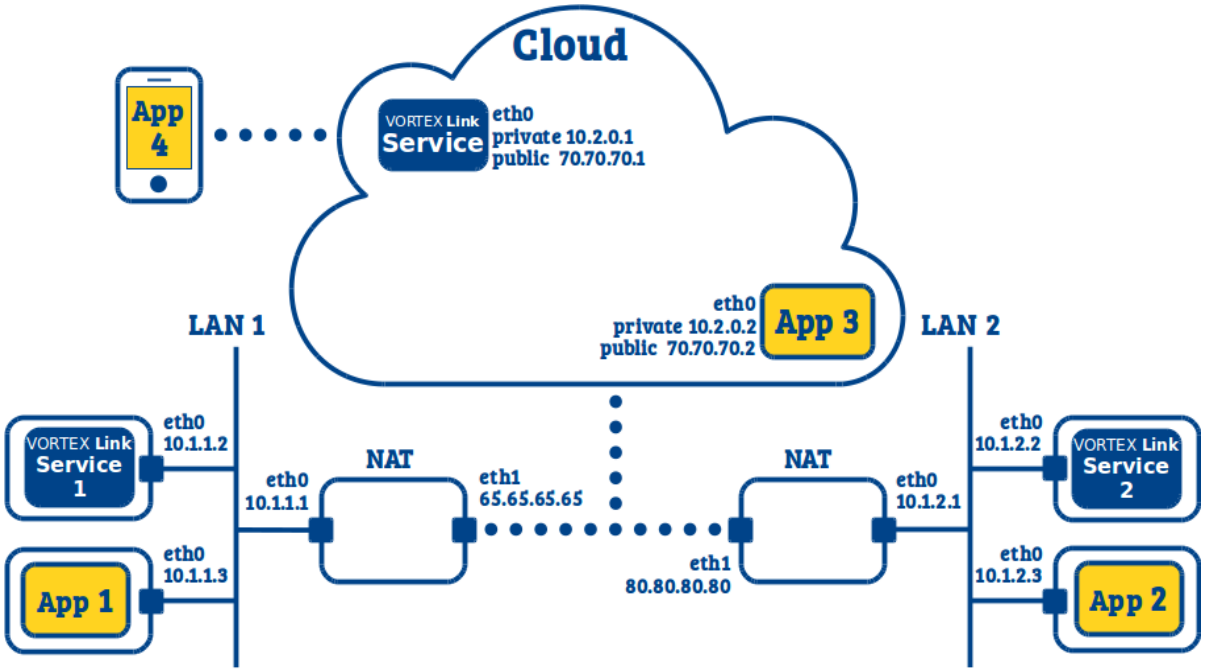
App 5 (using Vortex Café):

```
ddsi.network.udp.enabled=false
ddsi.discovery.tcp.peers=65.65.65.65:7400
ddsi.discovery.tcp.port=7400
ddsi.discovery.externalNetworkAddresses=none
```

LAN to LAN + Internet devices (with public cloud)

This use case is similar to the previous one (see LAN to LAN + Internet devices (no cloud)), but in addition we want applications deployed in a public cloud (without UDP multicast) to communicate with applications in the LAN and applications on devices using TCP.

To support this, we need to deploy a Vortex Link service in the cloud. Note that as the Vortex Link service is deployed in a cloud with a public IP, the TCP applications on devices can connect to it directly. Note also that as the Vortex Link service can mediate communications between the Vortex Link services in the sub systems, the Vortex Link services can be deployed on any host and do not need to be publicly accessible.



Vortex Link service:

```
link.network.interface=eth0
link.externalNetworkAddresses=70.70.70.1
link.serviceLevel=20
```

Vortex Link services 1 & 2:

```
link.network.interface=eth0
link.tcp.peers=70.70.70.1:7400
link.externalNetworkAddresses=none
```

App 1 & 2:

(nothing to configure)

App 3:

(nothing to configure)

App 4 (using Vortex Café):

```
ddsi.network.udp.enabled=false
ddsi.discovery.tcp.peers=70.70.70.1:7400
ddsi.discovery.tcp.port=7400
ddsi.discovery.externalNetworkAddresses=none
```

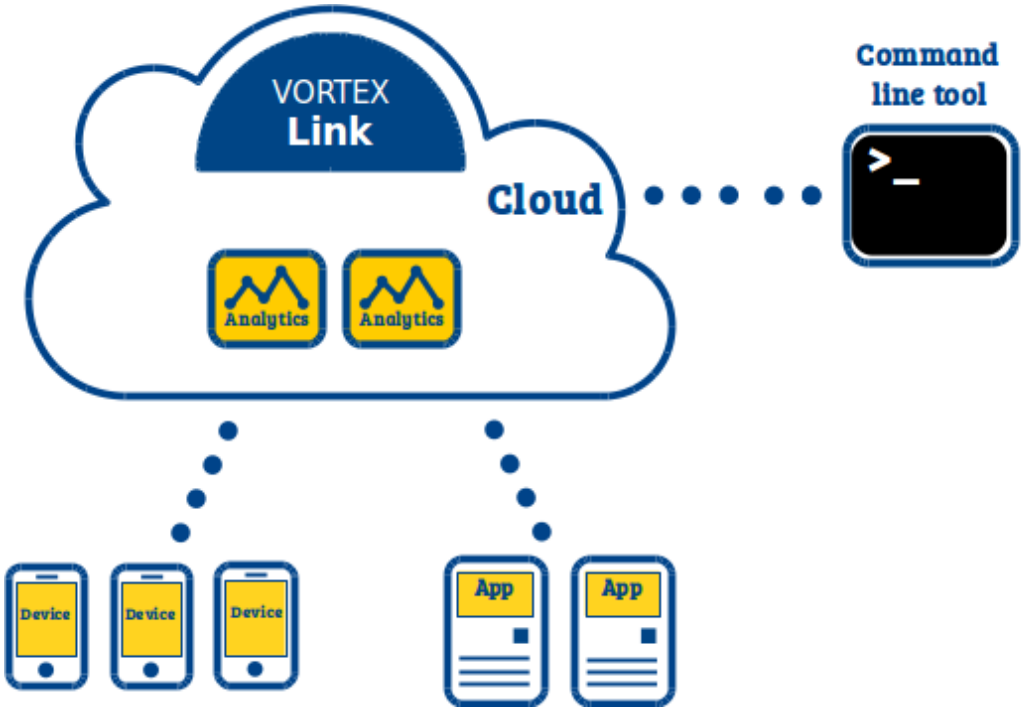
12

The Command Line Tool

The Command Line Tool shows some informations about a Vortex Link system:

- Services
- Participants
- Writers
- Readers

by getting data form one or more of the services of the system.



Commands

All the available commands on the Command Line Tool

service list

The tool prints a list of all the discovered Services and it shows their GUIDs plus some other properties:

- node name
- exec name

service <service_GUID>

The tool prints some informations about the specified Service:

- node name
- exec name

participant list

The tool prints a list of all the discovered Participants and it shows their GUID

writer list

The tool prints a list of all the discovered DataWriters and it shows their GUID plus some other informations:

- topic name
- topic kind

writer <writer_GUID>

The tool prints some informations about the specified DataWriter:

- topic name
- topic kind
- entity name
- QoS
  - GroupData
  - UserData
  - Lifespan
  - Partition
  - LatencyBudget
  - DestinationOrder
  - Deadline
  - Presentation
  - Ownership
  - DurabilityService
  - Durability
  - Liveliness
  - Reliability
  - ResourceLimits
  - History
  - TransportPriority
  - WriterDataLifeCycle
  - OwnershipStrength

reader list

The tool prints a list of all the discovered DataReaders and it shows their GUID plus some other informations:

- topic name
- topic kind

reader <reader_GUID>

The tool prints some informations about the specified DataReader:

- topic name
- topic kind
- entity name
- QoS
  - GroupData
  - UserData
  - Lifespan
  - Partition
  - LatencyBudget
  - DestinationOrder
  - Deadline
  - Ownership
  - DurabilityService
  - Durability
  - Liveliness
  - Reliability
  - ResourceLimits
  - History
  - ReaderDataLifeCycle
  - TimeBasedFilter

route list

The tool prints a list of all the discovered routes and it shows the Services discovered by GUID and all the routes between DataWriter and DataReader related to those Services:

- service GUID_x
  - writer GUID_y
  - reader GUID_z
  - writer GUID_y
  - reader GUID_r
- service GUID_r
  - writer GUID_g

- reader GUID_f
- writer GUID_e
- reader GUID_d
- writer GUID_e
- reader GUID_a

...

route <service_GUID>

The tool prints a list of all the discovered routes related to the specified Service

route <service_GUID> <writer_GUID> <reader_GUID>

The tool prints a list of only the specified route

route nb

The tool prints the total number of the discovered routes

route nb <service_GUID>

The tool prints the total number of the discovered routes releated to the specified Service

exit

This command closes the Tool

quit

This command closes the Tool

help

The list of all the available commands is printed

Examples

Service

Example 1

input:

```
> service list
```

output:

```
service [007f0101.000069f9.00000000 PARTICIP]
   node name : Unknown
   exec name : Unknown
service [004d0101.0000759e.00000000 PARTICIP]
   node name : Unknown
   exec name : Unknown
```

Example 2

input:

```
> service [007f0101.000069f9.00000000 PARTICIP]
```

output:

```
service [007f0101.000069f9.00000000 PARTICIP]
   node name : Unknown
   exec name : Unknown
```

Participant

Example 1

input:

```
> paticipant list
```

output:

```
participant [007f0101.00007607.00000000 PARTICIP]
participant [007f0101.000075ba.00000000 PARTICIP]
```

Writer

Example 1

input:

```
> writer list
```

output:

```
writer [007f0101.000075ba.00000000 00000302]
   topic name : Circle
   topic kind : ShapeType
writer [007f0101.000075ba.00000000 00000402]
   topic name : Square
   topic kind : ShapeType
```

Example 2

input:

```
> writer [007f0101.000075ba.00000000 00000302]
```

output:

```
writer [007f0101.000075ba.00000000 00000302]
   topic name : Circle
   topic kind : ShapeType
   entity name : Circle_DataWriter
   QoS :
          GroupData: Empty
          UserData: Empty
```

```
          Lifespan: Duration = INFINITE
          Partition: []
          LatencyBudget: Duration = 0
          DestinationOrder: Kind = BY_RECEPTION_TIMESTAMP
          Deadline: Period = INFINITE
          Presentation: AccessScope = INSTANCE
          Ownership: Kind =  SHARED
          DurabilityService:
                 HistoryDepth = 1, HistoryKind = KEEP_LAST
                 MaxInstances = -1, MaxSamples = -1
                 MaxSamplesPerInstance = -1
          Durability: Kind = VOLATILE
          Liveliness: Kind = AUTOMATIC, LeaseDuration = INFINITE
          Reliability: Kind = BEST_EFFORT, MaxBlockingTime = 0.099999998
          ResourceLimits:
                 MaxInstances = -1, MaxSamples = -1
                 MaxSamplesPerInstance = -1
          History: Kind = KEEP_LAST, Depth = 1
          TransportPriority: 50
          WriterDataLifeCycle:
                 AutoDisposeUnregisteredInstances = true
                 AutoPurgeSuspendedSamplesDelay = INFINITE
                 AutoUnregisterInstanceDelay = INFINITE
          OwnershipStrength: 50
```

Reader

Example 1

input:

```
> reader list
```

output:

```
reader [007f0101.00007607.00000000 00000407]
   topic name : Triangle
   topic kind : ShapeType
reader [007f0101.00007607.00000000 00000307]
   topic name : Circle
   topic kind : ShapeType
```

Example 2

input:

```
> reader [007f0101.00007607.00000000 00000307]
```

output:

```
reader [007f0101.00007607.00000000 00000307]
   topic name : Circle
   topic kind : ShapeType
   entity name : Circle_DataReader
   QoS :
          GroupData: Empty
          UserData: Empty
          Partition: []
          LatencyBudget: Duration = INFINITE
          DestinationOrder: Kind = BY_RECEPTION_TIMESTAMP
          Deadline: Period = INFINITE
          Ownership: Kind =  SHARED
          Durability: Kind = VOLATILE
```

```
          Liveliness: Kind = AUTOMATIC, LeaseDuration = INFINITE
          Reliability: Kind = BEST_EFFORT, MaxBlockingTime = 0.099999998
          ResourceLimits:
                 MaxInstances = -1, MaxSamples = -1
                 MaxSamplesPerInstance = -1
          History: Kind = KEEP_LAST, Depth = 1
          ReaderDataLifeCycle:
                 AutoPurgeDisposedSamplesDelay = INFINITE
                 AutoPurgeNoWriterSamplesDelay = INFINITE
          TimeBasedFilter:  Minimum Separation = 0
```

Route

Example 1

input:

```
> route list
```

output:

```
service [007f0101.0000759e.00000000 PARTICIP]
   writer [007f0101.000075ba.00000000 00000302]
      Unicast Locators List:
         TCP 10.100.1.227:8000
      No Multicast Locators
   reader [007f0101.00007607.00000000 00000307]
      Unicast Locators List:
         TCP 10.100.1.227:8100
      No Multicast Locators

   writer [007f0101.000075ba.00000000 00000302]
      Unicast Locators List:
         TCP 10.100.1.227:8000
      No Multicast Locators
   reader [007f0101.00007a74.00000000 00000307]
      Unicast Locators List:
         UDP 10.100.1.227:7431
      Multicast Locators List:
         UDP 239.255.0.1:7401

service [004d0101.000069f8.00000000 PARTICIP]
   writer [004d0101.000042ca.00000000 00000402]
      Unicast Locators List:
         TCP 10.100.1.70:8200
      No Multicast Locators
   reader [004d0101.00003407.00000000 00000507]
      Unicast Locators List:
         TCP 10.100.1.70:8100
      No Multicast Locators
```

Example 2

input:

```
> route list [007f0101.0000759e.00000000 PARTICIP]
```

output:

```
service [007f0101.0000759e.00000000 PARTICIP]
   writer [007f0101.000075ba.00000000 00000302]
      Unicast Locators List:
         TCP 10.100.1.227:8000
```

```
      No Multicast Locators
   reader [007f0101.00007607.00000000 00000307]
      Unicast Locators List:
         TCP 10.100.1.227:8100
      No Multicast Locators

   writer [007f0101.000075ba.00000000 00000302]
      Unicast Locators List:
         TCP 10.100.1.227:8000
      No Multicast Locators
   reader [007f0101.00007a74.00000000 00000307]
      Unicast Locators List:
         UDP 10.100.1.227:7431
      Multicast Locators List:
         UDP 239.255.0.1:7401
```

Example 3

input:

```
> route nb
```

output:

```
3
```

Example 5

input:

```
> route nb [007f0101.0000759e.00000000 PARTICIP]
```

output:

```
2
```

13

Troubleshooting

If you experience any problems with Vortex Link installation or usage then you can contact ADLINK support.

Please provide a full description of your platform, including the versions of tools you are using (such as JDK, Ant, Maven, *etc.*).

14

Contacts & Notices

Contacts

ADLINK Technology Corporation
400 TradeCenter
Suite 5900
Woburn, MA
01801
USA
Tel: +1 781 569 5819

ADLINK Technology Limited
The Edge
5th Avenue
Team Valley
Gateshead
NE11 0XA
UK
Tel: +44 (0)191 497 9900

ADLINK Technology SARL
28 rue Jean Rostand
91400 Orsay
France
Tel: +33 (1) 69 015354

Web: `http://ist.adlinktech.com`

E-mail: ist_info@adlinktech.com

Notices