Tigase Properties Guide

Tigase Properties Guide

Table of Contents

Chapter 1. --admins

Default value: none

Example: --admins = admin@domain.com

Possible values: user1@domain,user2@domain2

Description: 'Specifies a list of administrator accounts.'

Possible values: list of admin accounts: user1@domain,user2@domain2

Available since: 2.0.0

Chapter 2. --auth-db

Default value: tigase-custom

Example: --auth-db = db-type

Possible values: mysql|pgsql|ldap|drupal|tigase-auth|tigase-custom|class name

Description: Specifies authentication repository, where db-type can be one of possible values: mysql, pgsql, drupal, wp, tigase-auth and tigase-custom (if omitted: user-db is used in versions up to 5.0 and tigase-custom is the new default value starting from version 5.1) or the class name. For SQL database this is normally: tigase.db.jdbc.JDBCRepository.

Available since: 2.0.0

Chapter 3. --auth-db-uri

Default value: jdbc:mysql://localhost/tigasedb?user=tigase_user&password=mypass

Example: --auth-db-uri = jdbc:mysql://localhost/tigasedb?user=tigase_user&password=mypass

Possible values: db connection-uri.

Description: connection-uri is a full resource uri for user repository data source. (If omitted user-db-uri settings are used.)

Available since: 2.0.0

Chapter 4. --auth-domain-repo-pool

Default value: tigase.db.AuthRepositoryMDImpl

Example: --auth-domain-repo-pool = tigase.db.AuthRepositoryMDImpl

Possible values: 'class implementing AuthRepository.'

Description: Allows to specify an implementation for per-domain AuthRepository implementation. This is used only if the implementation provided by a default Tigase server package is not sufficient in the particular deployment. The implementation provides a DB (AuthRepository to be more specific) connection pool where each connection (AuthRepository) handles data for a different DNS domain (VHost). This allows for database (user data) sharing to improve the system performance and better balance the load.

Available since: 5.1.0

Chapter 5. --auth-repo-pool

Default value: tigase.db.AuthRepositoryPool

Example: --auth-repo-pool = tigase.db.AuthRepositoryPool

Possible values: 'class implementing AuthRepository.'

Description: Allows you to specify an implementation for the authentication repository connection pool. This is used only if the implementation provided by a default Tigase server package is not sufficient in a deployment. The implementation provides a DB (AuthRepository to be more specific) connection pool to improve the data access performance. The repository pool can offer data caching for improved performance or any other features necessary.

Available since: 5.1.0

Chapter 6. --auth-repo-pool-size

Default value: 10

Example: --auth-repo-pool-size = 25

Possible values: 'Number of db connections as integer.'

Description: The property allows to set the database connections pool size for the AuthRepository. **Please note** if this value is not specified, there are some cases where instead of a default for this property, the setting for --data-repo-pool-size can be used. This depends on the repository implementation and the way it is initialized.

Available since: 4.0.0

Chapter 7. --bind-ext-hostnames

Default value: none

Example: --bind-ext-hostnames = pubsub.host.domain

Possible values: 'comma separated list of domains.'

Description: This property enables setting a list of domains to be bound to the external component connection. Let's say we have a Tigase instance with only MUC and PubSub components loaded and we want to connect this instance to the main server via external component protocol. Using --external property we can define a domain (perhaps muc.devel.tigase.org), password, TCP/IP port, remote host address, connection type, etc… This would make one of our components (MUC) visible on the remote server.

To make the second component (PubSub) visible we would need to open another connection with the domain name (pubsub.devel.tigase.org) for the other component. Of course the second connection is redundant as all communication could go through a single connection. This is what this property is used. In our example with 2 components you can just put the 'pubsub.devel.tigase.org' domain as a value to this property and it will bind the second domain to a single connection on top of the domain which has been authenticated during protocol handshaking.

Available since: 5.0.0

Chapter 8. --bosh-close-connection

Default value: false

Example: --bosh-close-connection = true

Possible values: true|false

Description: This property globally disables Bosh keep-alive support for Tigase server. It causes the Bosh connection manager to force close the HTTP connection each time data is sent to the Bosh client. To continue communication the client must open a new HTTP connection.

This setting is rarely required but on installations where the client cannot control/disable keep-alive Bosh connections and keep-alive does not work correctly for some reason.

Available since: 5.1.0

Chapter 9. --bosh-extra-headers-file

Default value: etc/bosh-extra-headers.txt

Example: --bosh-extra-headers-file = /path/to/file.txt

Possible values: 'path to a file on the filesystem.'

Description: This property allows you to specify a path to a text file with additional HTTP headers which will be sent to a Bosh client with each request. This gives some extra flexibility for Bosh clients running on some systems with special requirements for the HTTP headers and some additional settings.

By default a file distributed with the installation contains following content:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Allow-Headers: Content-Type
Access-Control-Max-Age: 86400
```

This can be modified, removed or replaced with a different content on your installation.

Available since: 5.1.0

Chapter 10. --cl-conn-repo-class

Default value: tigase.cluster.repo.ClConSQLRepository

Example: --cl-conn-repo-class = tigase.cluster.repo.ClConDirRepository

Possible values: 'class implementing ComponentRepository.'

Description: This property allows setting of the class controlling cluster connections repository. The cluster connections repository is responsible for discovering cluster nodes which are part of the installation. Tigase in cluster mode establishes TCP/IP connections between cluster nodes to allow for user communication and exchanging cluster state metadata.

From Tigase XMPP Server version 5.2.0 the server supports cluster auto-configuration so no action, configuring, or any maintenance action is required to add a new cluster node.

As everything in Tigase, it is also a pluggable system so it is possible to implement/add new ways to synchronize information about cluster nodes on the system. Currently following cluster connection repositories are implemented:

1. **ClConSQLRepository** a default implementation which synchronizes cluster nodes information through SQL database. By default it uses the same database as the main Tigase DB, that is the User-Repository database. All the cluster nodes need an access to the same database for cluster nodes information synchronization.

For backward compatibility this mode reads the list of the cluster nodes from a configuration file (init.properties) as well. However, this is used only as an initial setup and, after startup time the cluster nodes are synchronized through the database.

It is recommended however, that the --cluster-nodes property is not used (should be removed or commented out) when the automatic reconfiguration mode is used. This is because we found out that when the network configuration and DNS names are not perfect than automatic mode may conflict with manual settings.

By default the same database as for the user repository is used for the cluster automatic cluster mode but, a different, separate database can be used as well. A DB URI for a different SQL database can be set using following configuration property:

```
cl-comp/repo-uri=jdbc:mysql://localhost/tigasedb?user=user&password=mypass
```

1. **ClConDirRepository** an alternative way to synchronize information about cluster nodes on the installation through filesystem. This might be used in case where DB is not accessible by the clustering code for some reason or synchronization through DB is not desired. To make use of the filesystem based cluster nodes synchronization you need to mount a directory via some network filesystem mechanism (like NFS for example) and point Tigase to the directory. The rest works the same way as through the DB. However, instead of writing cluster node metadata to some DB table, each node writes it's metadata to a separate file in the given directory. This mode is also compatible with the manual nodes configuration through --cluster-nodes but the same precautions and suggestions as for DB based automatic node hold.

   A default location of the directory is probably not very useful, as it points to etc/ directory, therefore, normally a correct location has to be set through the repo-uri property in a following way:

   ```
   cl-comp/repo-uri=/mount/tigase-cluster-repo
   ```

2. **ClConConfigRepository** is an implementation which allows you to revert back to the previous and manual cluster configuration through cluster nodes. However, since version **5.2.0** --cluster-nodes has been extended with ability to set password and port number for each cluster node.

Available since: 5.2.0

Chapter 11. --client-access-policy-file

Default value: etc/client-access-policy.xml

Example: --client-access-policy-file = /path/to/access-policy-file.xml

Possible values: 'path to a file on the filesystem.'

Description: The --client-access-policy-file property allows control of the cross domain access policy for Silverlight based web applications. The cross domain policy is controlled via XML file which contains the policy and rules.

By default Tigase is distributed with an example policy file which allows for full access from all sources to the whole installation. This is generally okay for most Bosh server installations. The configuration through the property and XML files allows for a very easy and flexible modification of the policy on any installation.

Available since: 5.2.0

Chapter 12. --cluster-connect-all

Default value: false

Example: --cluster-connect-all = true

Possible values: true|false

Description: The --cluster-connect-all property is used to open active connections to all nodes listed in the --cluster-nodes configuration property. This property should be used only on the node which is added to the live cluster at later time. Normally this new cluster node is not listed in the configuration of the existing cluster nodes. This is why they can not open connections the new node. The new node opens connection to all existing nodes instead. False is the default value and you can skip this option if you want to have it switched off.

Available since: 4.3.0

Chapter 13. --cluster-mode

Default value: false

Example: --cluster-mode = true

Possible values: true|false

Description: The property is used to switch cluster mode on. The default value is 'false' so you can normally skip the parameter if you don't want the server to run in the cluster mode. You can run the server in the cluster mode even if there is only one node running. The performance impact is insignificant and you will have the opportunity to connect mode cluster nodes at any time without restarting the server.

Available since: 4.0.0

Chapter 14. --cluster-nodes

Default value: none

Example: --cluster-nodes = node1.domain,node2.domain,node3.domain

Possible values: 'a comma separated list of hostnames.'

Description: The property is used to specify a list of cluster nodes running on your installation. The node is the full DNS name of the machine running the node. Please note the proper DNS configuration is critical for the cluster to work correctly. Make sure the 'hostname' command returns a full DNS name on each cluster node. Nodes don't have to be in the same network although good network connectivity is also a critical element for an effective cluster performance.

```
--cluster-nodes=host-a.domain.com:pass:port,host-b.domain.com:pass:port,host-c.dom
```

All cluster nodes must be connected with each other to maintain user session synchronization and exchange packets between users connected to different nodes. Therefore each cluster node opens a 'cluster port' on which it is listening for connections from different cluster nodes. As there is only one connection between each two nodes Tigase server has to decide which nodes to connect to and which has to accept the connection. If you put the same list of cluster nodes in the configuration for all nodes this is not a problem. Tigase server has a way to find and void any conflicts that are found. If you however want to add a new node later on, without restarting and changing configuration on old nodes, there is no way the old nodes will try to establish a connection to the new node they don't know them. To solve this particular case the next parameter is used.

Available since: 4.0.0

Chapter 15. --cm-ht-traffic-throttling

Default value: xmpp:25k:0:disc,bin:200m:0:disc

Example: --cm-ht-traffic-throttling = xmpp:25k:0:disc,bin:200m:0:disc

Possible values: 'comma separated list of traffic limits settings.'

Description: This property is used to specify traffic limit of non-user connections, that is s2s, external components and other high traffic server connections. The meaning of the property and values encoded are in the same way as for the --cm-traffic-throttling property.

Available since: 5.1.3

Chapter 16. --cm-see-other-host

Default value: tigase.server.xmppclient.SeeOtherHostHashed

Example: --cm-see-other-host=tigase.server.xmppclient.SeeOtherHostHashed

Possible values: "none" 'or class implementing SeeOtherHostIfc.'

Description: Allows you to specify a load balancing mechanism by specifying SeeOtherHostIfc implementation. More details about functionality and implementation details can be found in Tigase Load Balancing documentation.

Available since: 5.2.0

Chapter 17. --cm-traffic-throttling

Default value: xmpp:2500:0:disc,bin:20m:0:disc

Example: --cm-traffic-throttling = xmpp:2500:0:disc,bin:20m:0:disc

Possible values: 'comma separated list of traffic limits settings.'

Description: The --cm-traffic-throttling property allows you to limit traffic on user connections. These limits are applied to each user connection and if a limit is exceeded then a specified action is applied.

The property value is a comma separated list of traffic limits settings. For example the first part: xmpp:2500:0:disc specifies traffic limits for XMPP data to 2,500 packets allowed within last minute either sent to or received from a user and unlimited (0) total traffic on the user connection, in case any limit is exceeded the action is to **disconnect** the user.

- **[xmpp|bin]** traffic type, xmpp - XMPP traffic, that is limits refer to a number of XMPP packets transmitted, bin - binary traffic, that is limits refer to a number of bytes transmitted.

- **2500** maximum traffic allowed within 1 minute. 0 means unlimited, or no limits.

- **0** maximum traffic allowed for the life span of the connection. 0 means unlimited or no limits.

- **[disc|drop]** action performed on the connection if limits are exceeded. disc - means disconnect, drop - means drop data.

Available since: 5.1.3

Chapter 18. --cmpname-ports

Default value: 'depends on component.'

Example: --s2s-ports=5269,5270,5271

Possible values: 'comma separate list of TCP/IP port numbers.'

Description: The --cmpname-ports property is used to set a ports list for a connection manager. 'cmpname' is a component name of the connection manager you want to change ports numbers for. 'list of ports' is a comma separated list of ports numbers. For example for the server to server connection manager named s2s the property would like like the example below:

```
--s2s-ports=5269,5270,5271
```

Available since: 3.0.0

Chapter 19. --comp-class

Default value: 'depends on component.'

Example: --comp-class-1 = tigase.muc.MUCComponent

Possible values: 'class name.'

Description: This property is used to load an extra component to the server. Normally this parameter is used if you want to load a component which is not included in the config-type you use. You can, of course, load more than just one component using parameters: --comp-class-2, --comp-class-3 and so on…. Let's say you want to load the MUC component and the class name for the component is: tigase.muc.MUCService. The line in the properties file should look like:

```
--comp-class-1 = tigase.muc.MUCComponent
```

Available since: 3.0.0

Chapter 20. --comp-name

Default value: 'custom string.'

Example: --comp-name-1 = muc

Possible values: 'XMPP localnode string.'

Description: The --comp-name property is used to assign name 'name' to the non-standard component which is loaded by the server. It is normally used when you want to load components which are not loaded by the config-type you use. Together with --comp-class-1 it allows you to load any extra component to your server configuration. Of course you can load more than just one component, just use --comp-name-2, --comp-name-3 and so on… Let's say you want to load the MUC component. You can then put give it a name: muc and the setting would look like:

```
--comp-name-1 = muc
```

Available since: 3.0.0

Chapter 21. --cross-domain-policy-file

Default value: etc/cross-domain-policy.xml

Example: --cross-domain-policy-file = /path/to/cross-domain-policy.xml

Possible values: 'path to a file on the filesystem.'

Description: This property allows you to set a path to a file with cross domain access policy for flash based clients. This is a standard XML file which is sent to the flash client upon request.

A default file distributed with Tigase installations allows for full access for all. This is good enough for most use cases but it can be changed by simply editing the file.

Available since: 5.1.0

Chapter 22. --data-repo-pool-size

Default value: 10

Example: --data-repo-pool-size = 25

Possible values: 'Number of db connections as integer.'

Description: DataRepository is an abstraction layer between any higher level data access repositories such as UserRepository or AuthRepository and SQL database or JDBC driver to be more specific. Many implementations use DataRepository for DB connections and in fact on many installations they also share the same DataRepository instance if they connect to the same DB. In this case it is desired to use a specific connection pool on this level to an avoid excessive number of connections to the database.

It recommended to control the number of DB connections using this property rather than --user-repo-pool-size or --auth-repo-pool-size.

Available since: 5.1.0

Chapter 23. --debug

Default value: 'none'

Example: --debug = server,xmpp.impl

Possible values: 'comma separated list of Tigase's package names.'

Description: The --debug property is used to turn on the debug log for the selected Tigase package. For example if you want to turn debug logs on for the tigase.server package, then you have to use the --debug server parameter. If you have any problems with your server the best way to get help from the Tigase team is to generate configuration with '--debug = server' and run the server. Then from the logs/tigase-console.log log file I can provide the best information for us to provide assistance. More details about server logging and adjusting logging level is described in the Debugging Tigase article in the admin guide.

Available since: 2.0.0

Chapter 24. --debug-packages

Default value: 'none'

Example: --debug-packages = com.company.CustomPlugin,com.company.custom

Possible values: 'comma separated list of Java packages or classes.'

Description: This property is used to turn debugging on for any package not located within the default Tigase packages.

Available since: 5.0.0

Chapter 25. --domain-filter-policy

Default value: ALL

Example: --domain-filter-policy = LOCAL

Possible   values: ALL|LOCAL|OWN|BLOCK|LIST=domain1;domain2|
BLACKLIST=domain1;domain2

Description: The --domain-filter-policy property is a global setting for setting communication filtering
for vhosts. This function is kind of an extension of the same property which could be set on a single user
level. However, in many cases it is desired to control users communication not on per user-level but on
the domain level. Domain filtering (communication filtering) allows you to specify with whom users can
communicate for a particular domain. It enables restriction of communications for a selected domain or
for the entire installation. A default value ALL renders users for the domain (by default for all domains)
able to communicate with any user on any other domains. Other possible values are:

1. `ALL` a default value allowing users to communicate with anybody on any other domain, including
   external servers.

2. `LOCAL` allows users to communicate with all users on the same installation on any domain. It only
   blocks communication with external servers.

3. `OWN` allows users to communicate with all other users on the same domain. Plus it allows users to
   communicate with subdomains such as **muc.domain**, **pubsub.domain**, etc…

4. `BLOCK` value completely blocks communication for the domain or for the user with anybody else. This
   could be used as a means to temporarily disable account or domain.

5. `LIST` property allows to set a list of domains (users' JIDs) with which users on the domain can com-
   municate (i.e. *whitelist*).

6. `BLACKLIST` - user can communicate with everybody (like `ALL`), except contacts on listed domains.

This is a global property which is overridden by settings for particular vhosts and settings for particular
users.

A default settings for all virtual hosts for which the configuration is not defined. This settings is useful
mostly for installations with many virtual hosts listed in the init.property file for which there is no indi-
vidual settings specified. It allows default value for all of servers, instead of having to provide individual
configuration for each vhost.

ALL is also applied as a default value for all new vhosts added at run-time.

Available since: 5.2.0

Chapter 26. --elements-number-limit

Default value: 1000

Example: --elements-number-limit=20000

Possible values: 'any integer.'

Description: elements-number-limit configuration property allows configuring a Denial of Service protection mechanism which limits number of elements sent in stanza. It can be configured on a per ConnectionManager basis with the following configuration:

```
<ConnectionManager component>/elements-number-limit[I]=integer_number
```

for example (for ClusterConnectionManager):

```
cl-comp/elements-number-limit[I]=100000
```

Available since: 5.2.0

Chapter 27. --ext-comp

Default value: 'none'

Example: --ext-comp = localdomain,remotedomain,port,passwd,plain,accept

Possible   values: 'external   connection   definition   string.'   connection   string
localdomain,remotedomain,port,passwd,(plain|ssl),(accept|connect),routing

Description: Deprecated in favor of --external. *Support for this property is no longer maintained, please update your installation for the new way to connect external components.*

The --ext-comp property creates a connection over an external component protocol - XEP-0114 [http://xmpp.org/extensions/xep-0114.html]. The connection can be made to/from any XEP-0114 application such as IM transort, MUC, PubSub and others. It is also possible to separate Tigase internal components onto separate instances connected via XEP-0114 connections.

Note: It is also possible to generate a configuration for many external components. To do so use --ext-comp_1 parameters, --ext-comp_2 parameters and so on…

Available since: 2.0.0 **Discontiuned since** 4.3.0

Chapter 28. --extcomp-repo-class

Default value: tigase.server.ext.CompConfigRepository

Example: --extcomp-repo-class = tigase.server.ext.CompConfigRepository

Possible values: 'a class name implementing tigase.db.comp.ComponentRepository.'

Description: This can be used when --external is used to connect external components. The component responsible for maintaining external components connections can be reconfigured during run-time and can store configuration in a separate place, configuration file, database or any other data source. This property specifies implementation for this data source.

The property sets a class managing the component repository. There are 3 available now and you can implement and use your own. They are:

- 'tigase.server.ext.CompConfigRepository' which reads settings for external components from configuration only. It works well with ad-hoc commands, you can add, remove and update external component settings but your changes are not saved to any permanent storage.

- 'tigase.server.ext.CompDBRepository' which reads settings for external components from configuration and from a database. As a database backend it uses UserRepository. It works well with ad-hoc commands and your changes are stored in DB and are loaded after server restart. All data are cached in memory so it doesn't cause a significant load on database. As it uses UserRepository data are stored in a format hard for direct modifications in database and because of caching in memory all the data it is also not a good choice in cluster environment as changes made on one node are not quickly propagated to other nodes - a reload ad-hoc command must be called on all nodes.

- 'tigase.server.ext.CompSQLRepository' which reads initial settings from configuration and also stores data in SQL database. It automatically creates a dedicated DB table with a simple structure suitable for direct modification with SQL commands. It also doesn't cache any data. External component details are read from DB on demand when the component tries to authenticate. As it doesn't cache any data, each authentication request requires DB access therefore it may put some load on the DB. On the other hand it allows for easy external component management from a separate application connecting directly to DB and all changes are instantly picked up by all cluster nodes.

Available since: 4.3.0

Chapter 29. --external

Default value: 'none'

Example: --external = muc1.devel.tigase.org:passwd1:connect:5270:devel.tigase.org:accept:lb-class
Example: --external = muc1.devel.tigase.org:passwd1:(connect|listen):5270:devel.tigase.org:(accept|client):lb-class:(plain|ssl)

Possible values: 'external domains parameters list.'

Description: This property defines parameters for external component connections.

The component is loaded the same way as all other Tigase components. In your init.properties file you need to add 2 lines:

```
--comp-name-1 = ext
--comp-class-1 = tigase.server.ext.ComponentProtocol
```

This will load the component with an empty configuration and is practically useless. You have to tell the component on what port to listen to (or on what port to connect to) and external domains list with passwords.

As a value you have to put comma separated list of external domains settings. Each domain settings consist of a few, colon separated parameters. For example:

```
--external = muc1.devel.tigase.org:passwd1,muc2.devel.tigase.org:passwd2
```

This sets passwords for 2 external domains but does not say anything about port number or connection. The above list is a simplified syntax. The full syntax looks like this:

```
--external = muc1.devel.tigase.org:passwd1:listen:5270
```

or

```
--external = muc1.devel.tigase.org:passwd1:connect:5270:devel.tigase.org:accept:lb
```

Definition of each colon separated part:

1. external component **domain**;

2. **password** for this domain;

3. **connection type** - 'listen' for incoming connections or 'connect' for the remote server;

4. **port number** for the TCP/IP connection (listening on or connecting to);

5. **remote hostname address(es)** - if the connection type is 'connect' then the the remote hostname address should be here. For 'listen' connection type this parameter can be skipped. The first item on the list is always the remote domain name, if there are more entries, the rest is just addresses to connect to for this domain separated by a semicolon (;)

6. **protocol** - if the connection type is 'connect' then the protorol is defined here - 'accept' for XEP-0114, 'client' for XEP-0225, possibly others in the future. It can be skipped for 'listen' connection types.

7. **lb-class** - is a class name for a load-balancer plugin. This is used only where there are multiple connections from the external component and you want to spread the resource load among them. More details and examples are in this guide.

8. **socket** - it's possible to specify whether this should be a `plain` socket (accepting unencrypted or TLS connections), `ssl` socket (accepting SSL connections) or `tls` (enforcing TLS connection)

Only 2 first parts are mandatory, the rest is optional. The simplified form is used to provide a list of domains:password elements for external components connections.

The settings on the server side may most likely looks like this:

```
--external=muc1.devel.tigase.org:passwd1:listen:5270,muc2.devel.tigase.org:passwd2
```

It specifies 3 domains with passwords and one TCP/IP port to listen to. On the other hand you can specify a configuration which would establish the connection to the server:

```
--external = muc1.devel.tigase.org:passwd1:connect:5270:devel.tigase.org:accept
```

We use one of the domains configured on the server side, the same port number and the server address. (Assuming the main server works at devel.tigase.org address).

Available since: 4.3.0

Chapter 30. --hardened-mode

Default value: false

Example: --hardened-mode=true

Possible values: true|false

Description: Enabling hardened mode affects handling of security aspects within Tigase. It turns off workarounds for SSL issues, turns off SSLv2 and forces enabling more secure ciphers suites. It also forces requirement of StartTLS.

Enabling it requires UnlimitedJCEPolicyJDK [http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html] installed. We prefer to use OracleJDK as our tests revealed that using OpenJDK in hardened mode may cause issues with some clients on some platforms.

Available since: 5.2.0

Chapter 31. --max-queue-size

Default value: 'depends on RAM size.'

Example: --max-queue-size = 10000

Possible values: 'integer number.'

Description: The --max-queue-size property sets internal queues maximum size to a specified value. By default Tigase sets the queue size depending on the maximum available memory to the Tigase server process. It set's 1000 for each 100MB memory assigned for JVM. This is enough for most cases. If you have however, an extremely busy service with Pubsub or MUC component generating huge number of packets (presence or messages) this size should be equal or bigger to the maximum expected number of packets generated by the component in a single request. Otherwise Tigase may drop packets that it is unable to process.

Available since: 5.1.0

Chapter 32. --monitoring

Default value: 'none'

Example: --monitoring = jmx:9050,http:9080,snmp:9060

Possible values: 'list of monitoring protocols with port numbers.'

Description: This property activates monitoring interfaces through selected protocols on selected TCP/IP port numbers. For more details please refer to the monitoring guide in the user guide for details.

Available since: 4.0.0

Chapter 33. --net-buff-high-throughput

Default value: 64k

Example: --net-buff-high-throughput = 256k

Possible values: 'network buffer size as integer.'

Description: The --net-buff-high-throughput property sets the network buffer for high traffic connections like s2s or connections between cluster nodes. The default is 64k and is optimal for medium traffic websites. If your cluster installation can not cope with traffic between nodes try to increase this number.

Available since: 4.3.0

Chapter 34. --net-buff-standard

Default value: 2k

Example: --net-buff-standard = 16k

Possible values: 'network buffer size as integer.'

Description: This property sets the network buffer for standard (usually c2s) connections, default value is 2k and is optimal for most installations.

Available since: 4.3.0

Chapter 35. --new-connections-throttling

Default value: 5222:200,5223:50,5269:100,5280:1000

Example: --new-connections-throttling = 5222:100

Possible values: 'a list of port numbers with throttling thresholds as integer.'

Description: The property allows you to limit how many new users' connection per second the server accepts on a particular port. Connections established within the limit are processed normally, all others are simply disconnected. This allows you to avoid server overload in case there is a huge number of users trying to connect at the same time. Mostly this happens after a server restart.

The property value is a list of comma separated port settings. Each port setting is formatted in a following way: PORT_NO:LIMIT_VAL

Available since: 5.1.0

Chapter 36. --nonpriority-queue

Default value: false

Example: --nonpriority-queue = true

Possible values: true|false

Description: The --nonpriority property can be used to switch to non-priority queues usage in Tigase server (value set to 'true'). Using non-priority queues prevents packets reordering. By default Tigase uses priority queues which means that packets with highest priority may take over packets with lower priority (presence updates) which may result in packets arriving out of order.

This may happen however only for packets of different types. That is, messages may take over presence packets. However, one message never takes over another message for the same user. Therefore, out of order packet delivery is not an issue for the most part.

Available since: 5.0.0

Chapter 37. --queue-implementation

Default value: tigase.util.PriorityQueueRelaxed

Example: --queue-implementation = tigase.util.PriorityQueueStrict

Possible values: 'class name extending tigase.util.PriorityQueueAbstract.'

Description: The --queue-implementation property sets Tigase's internal queue implementation. You can choose between already available and ready to use or you can create own queue implementation and let Tigase load it instead of the default one. Currently following queue implementations are available:

1. **tigase.util.PriorityQueueRelaxed** - specialized priority queue designed to efficiently handle very high load and prevent packets loss for higher priority queues. This means that sometimes, under the system overload packets may arrive out of order in cases when they could have been dropped. Packets loss (drops) can typically happen for the lowest priority packets (presences) under a very high load.

2. **tigase.util.PriorityQueueStrict** - specialized priority queue designed to efficiently handle very high load but prefers packet loss over packet reordering. It is suitable for systems with a very high load where the packets order is the critical to proper system functioning. This means that the packets of the same priority with the same source and destination address are never reordered. Packets loss (drops) can typically happen for all packets with the same probability, depending which priority queue is overloaded.

3. **tigase.util.NonpriorityQueue** - specialized non-priority queue. All packets are stored in a single physical collection, hence they are never reordered. Packets are not prioritized, hence system critical packets may have to wait for low priority packets to be processed. This may impact the server functioning and performance in many cases. Therefore this queue type should be choosen very carefully. Packets of the same type are never reordered. Packets loss (drops) can typically happen for all packets which do not fit into the single queue.

Please note! *Since the packets are processed by plugins in the SessionManager component and each plugin has own thread-pool with own queues packet reordering may happen regardless what queue type you set. The reordering may only happen, however between different packet types. That is 'message' may take over 'iq' packet or 'iq' packet may take over 'presence' packet and so on... This is unpredictable.*

Available since: 5.1.0

Chapter 38. --roster-implementation

Default value: RosterFlat.class.getCanonicalName()

Example: --roster-implementation=my.pack.CustomRosterImpl

Possible values: 'Class extending tigase.xmpp.impl.roster.RosterAbstract.'

Description: The --roster-implementation property allows you to specify a different RosterAbstract implementation. This might be useful for a customized roster storage, extended roster content, or in some cases for some custom logic for certain roster elements.

Available since: 5.2.0

Chapter 39. --s2s-ejabberd-bug-workaround-active

Default value: false

Example: --s2s-ejabberd-bug-workaround-active = true

Possible values: true|false

Description: This property activates a workaround for a bug in EJabberd in it's s2s implementation. EJabberd does not send dialback in stream features after TLS handshaking even if the dialback is expected/needed. This results in unusable connection as EJabberd does not accept any packets on this connection either. The workaround is enabled by default right now until the EJabberd version without the bug is popular enough. A disadvantage of the workaround is that dialback is always performed even if the SSL certificate is fully trusted and in theory this dialback could be avoided.

Available since: 5.1.0

Chapter 40. --s2s-secret

Default value: <null>

Example: --s2s-secret = some-s2s-secret

Possible values: 'ascii string.'

Description: This property is a global setting for s2s secrets to generate dialback keys on the Tigase installation. By default it is null, which means the secret is automatically generated for each s2s connection and handshake.

This is a global property which is overridden by settings for each vhost.

A default settings for all virtual hosts for which the configuration is not defined. This settings is useful mostly for installations with many virtual hosts listed in the init.property file for which there is no individual settings specified. It allows to configure a default values for all of them, instead of having to provide individual configuration for each vhost.

It is also applied as a default value for all new vhosts added at run-time.

Available since: 5.2.0

Chapter 41. --s2s-skip-tls-hostnames

Default value: 'none'

Example: --s2s-skip-tls-hostnames = domain1,domain2

Possible values: 'comma separated list of domains.'

Description: The --s2s-skip-tls-hostnames property disables TLS handshaking for s2s connections to selected remote domains. Unfortunately some servers (certain versions of Openfire - [1 [http:// community.igniterealtime.org/thread/36206]] or [2 [http://community.igniterealtime.org/thread/30578]]) have problems with TLS handshaking over s2s which prevents establishing a usable connection. This completely blocks any communication to these servers. As a workaround you can disable TLS for these domains to get communication back.

Available since: 5.1.0

Chapter 42. --script-dir

Default value: scripts/admin

Example: --script-dir = /opt/admin-scripts

Possible values: 'path to a directory on the filesystem.'

Description: This property sets the directory where all administrator scripts for ad-hoc command are stored.

Available since: 4.3.0

Chapter 43. --sm-cluster-strategy-class

Default value: tigase.cluster.strategy.SMNonCachingAllNodes

Example: --sm-cluster-strategy-class=tigase.cluster.strategy.SMNonCachingAllNodes

Possible values: 'any class implementing tigase.cluster.strategy.ClusteringStrategyIfc interface.'

Description: The --sm-cluster-strategy-class property allows users to specify Clustering Strategy class which should be used for handling clustering environment; by default SMNonCachingAllNodes is used.

Available since: 4.0.0

Chapter 44. --sm-plugins

Default value: 'none'

Example: --sm-plugins = -jabber:iq:register, your-plugin

Possible values: 'comma separated list of plugins IDs.'

Description: The --sm-plugins property allows users to specify a list of plugins which should be loaded or disabled on the server. Normally you don't have to specify this. Tigase server loads a default list of plugins automatically. The default list contains all available plugins. Sometimes however you might want to load only some plugins. Typical use case is when user accounts are managed on a third-party system that Tigase server is integrated with. Then you might not want to allow users to register new accounts via the XMPP service. You can then load a list of plugins without the user registration plugin. Another case when you usually have to use this option is when you have your own plugins which replace function of the Tigase default plugins like vCard, roster management, and so on….

To prevent the plugin from loading add '-' before its ID, to load the plugin add '+' (which is optional). For example, the following setting would switch user registration off, while adding a new plugin: 'your-plugin':

```
--sm-plugins=-jabber:iq:register,+your-plugin
```

There is more. Each plugin is running in one or more separate threads. Most plugins which deal with databases have a number of threads equal to the number of CPUs or CPU cores. Sometimes this is not enough. If the database is slow or there is a specific kind of traffic at a high level you might want to adjust the number of threads for the plugin. To set a different number of threads, just add '=N' where 'N' is the number of threads you want. The previous example has been modified to assign 8 threads for your plugin and 16 threads for authentication plugin:

```
--sm-plugins=-jabber:iq:register,+your-plugin=8,jabber:iq:auth=16
```

Available since: 3.0.0

Chapter 45. --sm-threads-pool

Default value: default

Example: --sm-threads-pool = custom:100

Possible values: default|custom:NN

Description: The --sm-threads-pool property allows you to fine-tune the SM plugin's (processors) thread pool. With the default settings every plugin gets his own thread pool. This guarantees the best performance and optimal resource usage. The downside of this setting is that packets can arrive out of order if they are processed within different thread pools.

If you really need or want to preserve the order for packets processed by different plugins then the solution is to use the 'custom' thread pool. In this case packets processed by all plugins are handled within a single thread pool. This guarantees that the packets are processed and delivered in the exact order they arrive. The number after the colon ':' specifies the thread pool size.

We can even fine tune this packet processing. Let's say you want most of the plugins to be executed within a single thread pool to preserve packet ordering for them, but for some selected plugins that should execute within separate thread pools to improve performance. Let's say, authentication packets and user registration can be actually executed in a separate thread pools as we do not worry about an order for them. Users cannot send or receive anything else before they authenticates anyway. The solution is to specify a number of threads for the selected plugins in the --sm-plugins property as described above. For example, setting a common thread pool for all plugins but registration and authentication can be done with following configuration:

```
--sm-threads-pool=custom:100
--sm-plugins=jabber:iq:register=8,jabber:iq:auth=16,urn:ietf:params:xml:ns:xmpp-sa
```

Available since: 5.1.0

Chapter 46. --ssl-certs-location

Default value: certs/

Example: --ssl-certs-location = /etc/vhost-certs

Possible values: 'location of SSL certificates directory on the filesystem.'

Description: This option allows you to specify the location where SSL certificates are stored. The meaning of this property depends on the SSL container class implementation. By default it just points to the directory where the server SSL certificates are stored in files in PEM format.

Available since: 5.1.0

Chapter 47. --ssl-container-class

v2.0, June 2014: Reformatted for AsciiDoc. :toc: :numbered: :website: http://tigase.net/ :Date: 2013-02-10 01:11

Default value: tigase.io.SSLContextContainer

Example: --ssl-container-class =

Possible values: a class implementing tigase.io.SSLContectContainerIfc.

Description: The --ssl-container-class property allows you to specify a class implementing storage for SSL/TLS certificates. The class presented in the example to this description allows for loading certificates from PEM files which is a common storage used on many systems.

Available since: 5.0.0

Chapter 48. --ssl-def-cert-domain

Default value: default

Example: --ssl-def-cert-domain =

Possible values: 'DNS domain name.'

Description: This property allows you to specify a default alias/domain name for certificate. It is mostly used to load certificates for unknown domain names during the SSL negotiation. Unlike in TLS protocol where the domain name is known at the handshaking time, for SSL domain name is not known, therefore, the server does not know which certificate to use. Specifying a domain name in this property allows you to use a certificate for a specific domain in such case. This property value is also sometimes used if there is no certificate for one of virtual domains and the container does not automatically generate a self-signed certificate, then it can use a default one.

Available since: 5.1.0

Chapter 49. --stats-archiv

Default value: 'By default, --stats-archiv is not listed in the init.properties file'

Example: --stats-archiv=tigase.stats.CounterDataFileLogger:stats-file-logger:60,tigase.stats.CounterDataLogger:stats-logger:60

Possible values: 'comma separated list of statistics archivers:' <class>:<name>:<frequency>

Description: --stats-archiv configuration property allow enabling and configuring components responsible for storing statistic information. As a parameter, it takes comma separated list of all archivers that we want to enable. Each entry follows same pattern: <class>:<name>:<frequency>, where:

- <class> - is the class of any archiver implementing tigase.stats.StatisticsArchivizerIfc interface

- <name> - is the name of particular archiver under which it will be identified in Tigase

- <frequency> - is the time interval between subsequent execution of the archiver .execute() method (in seconds).

Currently following archivers classes are available:

- tigase.stats.CounterDataArchivizer - every execution put current basic server metrics (CPU usage, memory usage, number of user connections, uptime) into database (overwrites previous entry);

- tigase.stats.CounterDataLogger - every execution insert new row with new set of number of server statistics (CPU usage, memory usage, number of user connections per connector, number of processed packets of different types, uptime, etc) into the database

- tigase.stats.CounterDataFileLogger - every execution store all server statistics into separate file.

It's possible to configure each archiver by adding following entry to the etc/init.properties file:

```
<statistics_component_name>/stats-archiv/<name>/<property>=<value>
```

For example, if we want to enable CounterDataFileLogger that stores statistics every 60 seconds:

```
--stats-archiv=tigase.stats.CounterDataFileLogger:stats-file-logger:60
```

We can configure it further with following options:

```
stats/stats-archiv/stats-file-logger/stats-directory=logs/server_statistics
stats/stats-archiv/stats-file-logger/stats-filename=stat
stats/stats-archiv/stats-file-logger/stats-unixtime=false
stats/stats-archiv/stats-file-logger/stats-datetime=true
stats/stats-archiv/stats-file-logger/stats-datetime-format=HH:mm:ss
stats/stats-archiv/stats-file-logger/stats-level=FINE
```

which configures accordingly: directory to which files should be saved, filename prefix, whether to include or not unix timestamp in filename, whether to include or not datetime timestamp, control format of timestamp (using java DateFormat pattern) and also set level of the statistics we want to save (using java Logger.Level)

Available since: 5.2.0

Chapter 50. --stats-history

Default value: 'none'

Example: --stats-history=2160,10

Possible values: SIZE-NUM,INTERVAL_NUM

Description: Tigase XMPP Server can store server statistics internally for a given period of time. This allows you to connect to a running system and collect all the server metrics along with historic data which are stored on the server.

This is very useful when something happens on your production system you can connect and see when exactly this happened and what other metrics looked around this time.

The property value consists of comma separated, 2 integer numbers. The first is a size of the buffer. That is how many complete sets of historic metrics to store in memory. The second specified how often to probe metrics on the server.

Please be aware that Tigase XMPP Server produces about 1,000 different metrics of the system. Therefore caching large number of statistics sets requires lots of memory.

Available since: 5.0.0

Chapter 51. --stringprep-processor

Default value: simple

Example: --stringprep-processor = libidn

Possible values: simple|libidn|empty

Description: The --stringprep-processor property sets the stringprep processor for all JIDs handled by Tigase. The default 'simple' implementation uses regular expressions to parse and check the user JID. It does not fulfill the RFC-3920 requirements but also puts much less stress on the server CPU, hence impact on the performance is very low.

Other possible values are:

'libidn' - provides full stringprep processing exactly as described in the RFC-3920. It requires lots of CPU power and significantly impacts performance.

'empty' - doesn't do anything to JIDs. JIDs are accepted in the form they are received. No impact on the performance and doesn't use any CPU. This is suitable for use in automated systems where JIDs are generated by some algorithm, hence there is no way incorrect JIDs may enter the system.

Available since: 5.0.0

Chapter 52. --test

Default value: false

Example: --test

Possible values: true|false|empty-string

Description: This property sets the server for test mode, which means that all logging is turned off, offline message storage is off, and possibly some other changes to the system configuration are being made.

The idea behind this mode is to test Tigase XMPP Server with minimal performance impact from environment such as hard drive, database and others…

Available since: 2.0.0

Chapter 53. --tigase-config-repo-class

Default value: tigase.conf.ConfigurationCache

Example: --tigase-config-repo-class = tigase.conf.ConfigXMLRepository

Possible values: 'name of class implementing tigase.conf.ConfigRepositoryIfc'

Description: This property is a parameter which allows loading of different configuration storage engines. The default setting stores configuration in memory only. Other possible are: tigase.conf.ConfigXMLRepository which keeps configuration in the old XML file and tigase.conf.ConfigSQLRepository which stores configuration in SQL database. **Please note** in all cases the init.properties file can still be read and works the same way - provides an initial settings for Tigase at startup.

Available since: 5.0.0

Chapter 54. --tigase-config-repo-uri

Default value: 'none'

Example: --tigase-config-repo-uri = jdbc:mysql://localhost/tigase?user=root&password=mypass

Possible values: 'DB connection URI string.'

Description: The --tigase-config-repo-uri property is a parameter which allows to provide database connection string for configuration storage engines which keep server settings in a database.

Available since: 5.0.0

Chapter 55. --tls-jdk-nss-bug-workaround-active

Default value: false

Example: --tls-jdk-nss-bug-workaround-active = true

Possible values: true|false

Description: This is a workaround for TLS/SSL bug in new JDK7 using the native library for keys generation and connection encryption used with new version of nss library.

This caused a number of problems with Tigase installed on systems with JDK7 and the new library installed, such as hanging connections, or broken SSL/TLS. There is some information on our wiki page [https://projects.tigase.org/projects/tigase-server/wiki/Tigase_with_OpenJDK7_with_OpenSSL_101]. Our earlier suggestion was to avoid using either JDK7 or the problematic native library. Now we have a proper fix/workaround which allows you to run Tigase with JDK7.

- http://stackoverflow.com/q/10687200/427545

- http://bugs.sun.com/bugdatabase/view_bug.do;jsessionid=b509d9cb5d8164d90e6731f5fc44?bug_id=6928796

Available since: 5.2.0

Chapter 56. --trusted

Default value: none

Example: --trusted = user@domain.com,user-2@domain2.com

Possible values: 'comma separated list of user bare JIDs.'

Description: The --trusted property allows users to specify a list of accounts which are considered as trusted, thus whom can perform some specific actions on the server. They can execute some commands, send a broadcast message, set MOTD and so on. The configuration is similar to --adimins setting.

Available since: 3.0.0

Chapter 57. --user-db

Default value: mysql

Example: --user-db = ldap

Possible values: 'UserRepository implementation class name or predefined string.'

Description: The --user-db property allows to specify UserRepository implementation - a storage where all users' data are located. Users' data include contact list (roster), privacy lists, vCards, and similar; everything except user authentication information. The implementation can be defined by one of possible values: mysql, pgsql, xml or the class name. For a SQL database this is normally: tigase.db.jdbc.JDBCRepository.

Available since: 2.0.0

Chapter 58. --user-db-uri

Default value: jdbc:mysql://localhost/tigase?user=root&password=mypass

Example: --user-db-uri = jdbc:postgresql://localhost/tigase?user=tigase

Possible values: 'Database connection URI.'

Description: The --user-db-uri property specify database connection string - connection-uri, where connection-uri is a full resource uri for user repository data source. If you skip this parameter, the default value is used depending on database type you selected:

- jdbc:mysql://localhost/tigase?user=root&password=mypass

- jdbc:postgresql://localhost/tigase?user=tigase

- user-repository.xml

Available since: 2.0.0

Chapter 59. --user-domain-repo-pool

Default value: tigase.db.UserRepositoryMDImpl

Example: --user-domain-repo-pool = tigase.db.UserRepositoryMDImpl

Possible values: 'Name of class implementing UserRepository.'

Description: This property allows you to specify an implementation for per-domain UserRepository implementation. This is used only if the implementation provided by a default Tigase server package is not sufficient in the particular deployment. The implementation provides a DB (UserRepository to be more specific) connection pool where each connection (UserRepository) handles data for a different DNS domain (VHost). This allows for database (user data) sharing to improve the system performance and better balance the resource load.

Available since: 5.1.0

Chapter 60. --user-repo-pool

Default value: tigase.db.UserRepositoryPool

Example: --user-repo-pool = tigase.db.UserRepositoryPool

Possible values: 'Name of class implementing UserRepository.'

Description: The --user-repo-pool property provides an ability to specify an implementation for the repository connection pool. This is used only if the implementation provided by a default Tigase server package is not sufficient in a particular deployment. The implementation provides a DB (UserRepository to be more specific) connection pool to improve the data access performance. The repository pool can offer data caching for improved performance or any other features necessary.

Available since: 5.1.0

Chapter 61. --user-repo-pool-size

Default value: 10

Example: --user-repo-pool-size = 25

Possible values: 'Number of db connections as integer.'

Description: This property allows setting the database connections pool size for the UserRepository.

Please note if not specified than in some cases instead of default for this property setting for --data-repo-pool-size can be used. This depends on the repository implementation and the way it is initialized.

Available since: 4.0.0

Chapter 62. --vhost-anonymous-enabled

Default value: true

Example: --vhost-anonymous-enabled = false

Possible values: true|false

Description: The --vhost-anonymous-enabled property specifies whether anonymous user logins are allowed for the installation for all vhosts.

This is a global property which is overridden by settings for particular vhost.

Default settings for all virtual hosts are used when this property is not defined. This settings is useful mostly for installations with many virtual hosts listed in the init.property file for which there is no individual settings specified. It allows the configuration of default values for all of them, instead of having to provide individual configuration for each vhost.

It is also applied as a default value for all new vhosts added at run-time.

Available since: 5.2.0

Chapter 63. --vhost-disable-dns-check

Default value: false

Example: --vhost-disable-dns-check = true

Possible values: true|false

Description: This property disables DNS validation when adding or editing vhosts in Tigase server. This also exempts administrative accounts from validation. With this property enabled, you will not benefit from seeing if proper SRV records are set so other people can connect to specific vhosts from outside your network.

Available since: 7.1.0

Chapter 64. --vhost-max-users

Default value: 0

Example: --vhost-max-users = 1000

Possible values: 'integer number.'

Description: The --vhost-max-users property specifies how many user accounts can be registered on the installations for all vhosts.

0 - zero means unlimited and this is a default. Otherwise greater than zero value specifies accounts number limit.

This is a global property which is overridden by settings for particular vhost.

The default setting is used for all virtual hosts for which the configuration is not defined. This settings is most useful for installations with many virtual hosts listed in the init.property file for which there is no individual settings specified. It provides an ability to use default values for all of them, instead of having to provide individual configuration for each vhost.

It is also applied as a default value for all new vhosts added at run-time.

Available since: 5.2.0

Chapter 65. --vhost-message-forward-jid

Default value: <null>

Example: --vhost-message-forward-jid = archive@domain.com

Possible values: 'valid JID'

Description: This is a global property for message forwarding for the installation. This property is normally specified on the vhost configuration level, however if you want to forward all messages on your installation and you have many virtual domains this property allows to set message forwarding for all of them. A valid JID must be specified as the forwarding destination. Also a message forwarding plugin must be loaded and activated on the installation for the message forwarding to work.

This is a global property which is overridden by settings for particular vhost.

The null value is used as a default when no configuration is set. This setting is mostly useful for installations with many virtual hosts listed in the init.property file for which there is no individual settings specified. It provides the ability to configure a default values for all of them, instead of having to provide individual configuration for each vhost.

It is also applied as a default value for all new vhosts added at run-time.

Available since: 5.2.0

Chapter 66. --vhost-presence-forward-jid

Default value: <null>

Example: --vhost-presence-forward-jid = presence-collector@domain.com

Possible values: 'valid JID.'

Description: This is a global property for presence forwarding function for the installation. All user status presences will be forwarded to given XMPP address which can be a component or any other XMPP entity. If the destination entity is a bot connected via c2s connection it probably should be addressed via full JID (with resource part) or the standard XMPP presence processing would refuse to deliver presences from users who are not in the contact list.

This is a global property which is overridden by settings for particular vhost.

The null value is used as a default when no configuration is set. This settings is useful for installations with many virtual hosts listed in the init.property file for which there is no individual settings specified. It enables the ability to configure default values for all of them, instead of having to provide individual configuration for each vhost.

It is also applied as a default value for all new vhosts added at run-time.

Available since: 5.2.0

Chapter 67. --vhost-register-enabled

Default value: true

Example: --vhost-register-enabled = false

Possible values: true|false

Description: --vhost-register-enabled is a global property which allows you to switch on/off user registration on the installation. Setting this property to false does not disable the registration plugin on the server. You can enable registration for selected domains in the domain configuration settings.

This is a global property which is overridden by settings for particular vhost.

The true value is used as a default when no configuration is set. This settings is useful for installations with many virtual hosts listed in the init.property file for which there is no individual settings specified. It allows admins to configure default values for all of them, instead of having to provide individual configuration for each vhost.

It is also applied as a default value for all new vhosts added at run-time.

Available since: 5.2.0

Chapter 68. --vhost-tls-required

Default value: false

Example: --vhost-tls-required = true

Possible values: true|false

Description: This property is a global settings to switch on/off TLS required mode on the Tigase installation. Setting this property to false does not turn TLS off. The TLS is still available on the server but as an option and this is the client's decision whether to use encryption or not. If the property is set to true the server will not allow for user authentication or sending any other user data before TLS handshaking is completed.

This is a global property which is overridden by settings for particular vhost.

The false value is used as a default when no configuration is set. This settings is useful for installations with many virtual hosts listed in the init.property file for which there is no individual settings specified. It allows admins to configure default values for all of them, instead of having to provide individual configuration for each vhost.

It is also applied as a default value for all new vhosts added at run-time.

Available since: 5.2.0

Chapter 69. --virt-hosts

Default value: 'hostname'

Example: --virt-hosts = domain1,domain2,domain3

Possible values: 'comma separated list of domains.'

Description: The --virt-hosts property allows to set a list of virtual domains served by the installation. This is just a list of virtual domains loaded at the startup time. Domains can be added, removed, disabled or updated at runtime. All the actual domain metadata are stored in the database.

Some metadata for vhosts can be also provided in the init.properties configuration file using this property. Domain is separated with ':' from its metadata. For boolean values '-' in front of the parameter means the feature is **off**. + or nothing means it is **on**. If a parameter requires some additional settings it is provided in form: param=val.

Here is an example:

```
--virt-hosts=domain1:-anon:register:-tls-required:s2s-secret=s2spasswd:\
  domain-filter=LOCAL:max-users=1000,domain2,\
  domain3:c2s-ports-allowed=5222;5223;5280;5290
  domain3:domain-filter=LIST=whitedomain1;whitedomain2;whitedomain3
```

Available since: 4.3.0

Chapter 70. --watchdog_delay

Default value: 600000

Example: --watchdog_delay=30000

Possible values: 'any integer.'

Description: --watchdog_delay configuration property allows configuring delay (in miliseconds) between subsequent checks that ConnectionManager Watchdog (service responsible for detecting broken connections and closing them) will use to verify the connection. In addition to global configuration presented above a per component configuration is possible:

```
<ConnectionManager component>/watchdog_delay[L]=60000
```

for example (for ClusterConnectionManager):

```
cl-comp/watchdog_delay[L]=150000
```

All related configuration options:

- --watchdog_ping_type

- --watchdog_delay

- --watchdog_timeout

Available since: 5.2.1

Chapter 71. --watchdog_ping_type

Default value: whitespace

Example: --watchdog_ping_type=xmpp

Possible values: whitespace,xmpp

Description: watchdog_ping_type configuration property allows configuring the type of ping that ConnectionManager Watchdog (service responsible for detecting broken connections and closing them) will use to check the connection. In addition to global configuration presented above a per component configuration is possible:

```
<ConnectionManager component>/watchdog_ping_type[S]=xmpp
```

for example (for ClusterConnectionManager):

```
cl-comp/watchdog_ping_type[S]=xmpp
```

All related configuration options:

- --watchdog_ping_type

- --watchdog_delay

- --watchdog_timeout

Available since: 5.2.1

Chapter 72. --watchdog_timeout

Default value: 1740000

Example: --watchdog_timeout=60000

Possible values: 'any integer.'

Description: The watchdog_timeout property allows for fine-tuning ConnectionManager Watchdog (service responsible for detecting broken connections and closing them). Timeout property relates to the amount of time (in miliseconds) after which lack of response/activity on a given connection will considered such connection as broken an close it. In addition to global configuration presented above a per component configuration is possible:

```
<ConnectionManager component>/watchdog_timeout[L]=60000
```

for example (for ClusterConnectionManager):

```
cl-comp/watchdog_timeout[L]=150000
```

All related configuration options:

- --watchdog_ping_type

- --watchdog_delay

- --watchdog_timeout

Available since: 5.2.1

Chapter 73. --ws-allow-unmasked-frames

Default value: false

Example: --ws-allow-unmasked-frames = true

Possible values: true|false

Description: RFC 6455 specifies that all clients must mask frames that it sends to the server over Websocket connections. If unmasked frames are sent, regardless of any encryption, the server must close the connection. Some clients however, may not support masking frames, or you may wish to bypass this security measure for development purposes. This setting, when enabled true, will allow connections over websocket to be unmasked to the server, and may operate without Tigase closing that connection.

Available since: 7.1.0

Chapter 74. -config-type

Default value: -config-type:--gen-config-def

Possible values: --gen-config-def|--gen-config-all|--gen-config-sm|--gen-config-cs|--gen-config-comp

Description: Probably the only such a property not starting with double hyphen. It sets the server type and determines what components are included in the generated XML file. Possible values are listed below:

- '--gen-config-all' - creating configuration file with all available components. That is: session manager, client-to-server connection manager, server-to-server connection manager, one external component connection manager, stanza sender and stanza receiver.

- '--gen-config-def' - creating default configuration file. That is configuration which is most likely needed for a typical installation. Components included in configuration are: session manager, client-to-server connection manager and server-to-server connection manager.

- '--gen-config-sm' - creating configuration for instance with session manager and external component only. This is useful for distributed installation where you want to have session manager installed on separate machine and components managing network connections on different machines (one or more). Components included in configuration are: sm and ext2s.

- '--gen-config-cs' - creating configuration for instance with components managing network connections. This is useful for distributed installation where you want to have session manager installed on separate machine and components managing network connections on different machines (one or more). Components included in configuration are: c2s, s2s, ext2s.

- '--gen-config-comp' - generating a configuration with only one component - component managing external components connection, either XEP-0114 or XEP-0225. This is used to deploy a Tigase instance as external component connecting to the main server. You have to add more components handled by this instance, usually these are MUC, PubSub or any other custom components. You have to refer to description for --comp-name and --comp-class properties to learn how to add components to the Tigase instance. You also have to configure the external component connection, domain name, password, port, etc... Please look for a description for --external and --bind-ext-hostnames properties.

Available since: 2.0.0

Chapter 75. --client-port-delay-listening

Default value: `true`

Example: `--client-port-delay-listening=true`

Possible values: `true`, `false`

Description: The property allows to enabled or disable delaying of listening for client connections **in cluster mode** until the cluster is correctly connected.

In cluster mode, in order to ensure correct user status broadcast, we are delaying opening client ports (components: `c2s`, `ws2s`, `bosh`) and enable those only after cluster is fully and correctly connected (i.e. either there is only single node or in case of multiple nodes all nodes connected correctly).

It's possible to enable/disable this on per-component basis with the following configuration:

```
bosh/port-delay-listening[B]=true
c2s/port-delay-listening[B]=true
ws2s/port-delay-listening[B]=true
```

Maximum delay time depends on the component and it's multiplication of `ConnectionManager` default connection delay times `30s` - in case of client connection manager this delay equals 60s

Note

Only applicable in **Cluster Mode**!

Available since: 7.1.0