Grouper always available web services and client

Wiki Home | Download Grouper | Grouper Guides | Community Contributions | Developer Resources | Deployment Guide

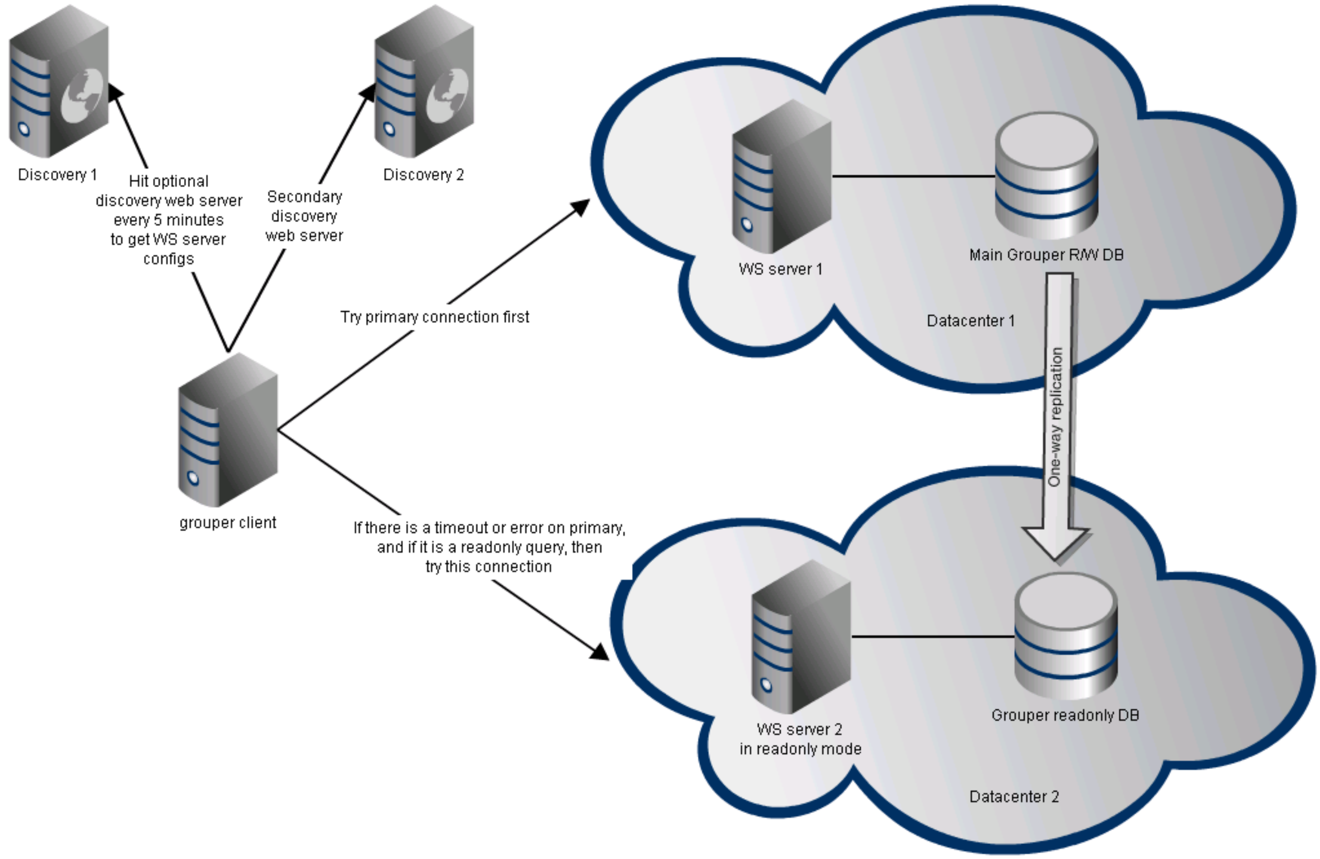
Always available web services are WS that do not have a single point of failure. This includes the loadbalancer, database, and even the data center.

Grouper is conducive to always available web services.

Grouper allows readonly database replication to an off site database, with a Grouper WS application server which runs in readonly mode against that database (set readonly in the grouper.properties of the WS application). Then the GrouperClient is a reference implementation showing how clients could interact with the multiple WS servers. Note that there might not be immediate consistency between the read/write and readonly depending on how the database replication is implemented.

If you want highly available web services (though not always available), then you could use a DNS-based loadbalancer that polls the servers to see which ones are available. Though there could be a minute or two of downtime here or there until the loadbalancer catches up. This does not require a special client.

Note that you could do local active/active load balancing but you might have a point of failure on the load balancer, firewall, or data center itself. This also does not require a special client. Note, this is just one way to do it, there are other ways to improve availability as well...


Discovery 1
Hit optional discovery web server every 5 minutes to get WS server configs
Secondary discovery web server
Discovery 2
WS server 1
Main Grouper R/W DB
Datacenter 1
Try primary connection first
One-way replication
grouper client
If there is a timeout or error on primary, and if it is a readonly query, then try this connection
WS server 2 in readonly mode
Grouper readonly DB
Datacenter 2

WS discovery

This uses the Grouper discovery client

If you are going to have multiple WS servers available for clients, and the load balancing managed on the client, then you might want to use a discovery service so the list of servers can be managed centrally. Here is an example config (grouper.client.discovery.properties) of a WS discovery service that is HTTPS based. Basically you just put this config file on multiple (and disparate) HTTPS servers, and the client would periodically download them: You can put any or all of these settings in the configuration, each can be defaulted or overridden in the client.

```
#
# Grouper client discovery configuration
#
```

```
########################################
## LDAP discovery connection settings
########################################

# urls of directory, including the base DN (distinguished name)
# add more properties and increment the integer (.1, .2, etc)
# e.g. ldap://server.school.edu/dc=school,dc=edu
# e.g. ldaps://server.school.edu/dc=school,dc=edu
grouperClient.discovery.ldap.0.url =

# active/active or active/standby
# active/active will pick a server randomly,
# and will stick with it for as long as the affinity is set
# active/standby will always use the first connection
# if no errors, then try the second one etc.
# if a connection has more errors and has a higher priority,
# then it will not be tried again until the
# takeConnectionOutOfPoolOnErrorForSeconds timeout
# passes
grouperClient.discovery.ldap.loadBalancing = active/active

# if we are active/active, then the same connection will
# be used for a certain number of seconds.  If this is -1, then
# always keep the same server (unless errors)
grouperClient.discovery.ldap.affinitySeconds = 600

# if a connection has more errors than another, it will not be
# used until this error timeout passes (unless the other is throwing errors
# too)
grouperClient.discovery.ldap.lowerConnectionPriorityOnErrorForMinutes = 3

# when a connection is attempted, this is the timeout that it will use before trying
# another connection
grouperClient.discovery.ldap.timeoutSeconds = 30

# after all connections have been attempted, it will wait for this long
# to see if any finish
grouperClient.discovery.ldap.extraTimeoutSeconds = 15

########################################
## Web service discovery Connection settings
########################################

# urls of web service, should include everything up to the first resource to access
# this is for read or write operations
# add more properties and increment the integer (.1, .2, etc)
# e.g. http://groups.school.edu:8090/grouper-ws/servicesRest
# e.g. https://groups.school.edu/grouper-ws/servicesRest
grouperClient.discovery.webService.readWrite.0.url =

# url of web service, should include everything up to the first resource to access
# this is for only read operations
# add more properties and increment the integer (.1, .2, etc)
# e.g. http://groups.school.edu:8090/grouper-ws/servicesRest
# e.g. https://groups.school.edu/grouper-ws/servicesRest
grouperClient.discovery.webService.readOnly.0.url =

# active/active or active/standby
# active/active will pick a server randomly,
# and will stick with it for as long as the affinity is set
# active/standby will always use the first connection
# if no errors, then try the second one etc.
# if a connection has more errors and has a higher priority,
# then it will not be tried again until the
# takeConnectionOutOfPoolOnErrorForSeconds timeout
# passes
grouperClient.discovery.webService.loadBalancing = active/active

# if you want to always try read/write before readOnly (i.e. if you are
# worried about if you make a write and read right after each other)
grouperClient.discovery.webService.preferReadWrite = true
```

```
# if we are active/active, then the same connection will
# be used for a certain number of seconds.  If this is -1, then
# always keep the same server (unless errors)
grouperClient.discovery.webService.affinitySeconds = 600

# if a connection has more errors than another, it will not be
# used until this error timeout passes (unless the other is throwing errors
# too)
grouperClient.discovery.webService.lowerConnectionPriorityOnErrorForMinutes = 3

# when a connection is attempted, this is the timeout that it will use before trying
# another connection
grouperClient.discovery.webService.timeoutSeconds = 60

# after all connections have been attempted, it will wait for this long
# to see if any finish
grouperClient.discovery.webService.extraTimeoutSeconds = 30
```

Here is the grouper client configuration (addition to the grouper.client.properties) to find the discovery configs, and to set defaults or overrides:

```
####
## Below here are default values and override values for the discovery
## properties at your institution.  Note: if the override keys are there
## with no value then it will blank out the discovery service value
####

# default urls of directory, including the base DN (distinguished name)
# add more properties and increment the integer (.1, .2, etc)
# e.g. ldap://server.school.edu/dc=school,dc=edu
# e.g. ldaps://server.school.edu/dc=school,dc=edu
grouperClient.discoveryDefault.ldap.0.url =
#grouperClient.discoveryOverride.ldap.0.url =

# default active/active or active/standby
# active/active will pick a server randomly,
# and will stick with it for as long as the affinity is set
# active/standby will always use the first connection
# if no errors, then try the second one etc.
# if a connection has more errors and has a higher priority,
# then it will not be tried again until the
# takeConnectionOutOfPoolOnErrorForSeconds timeout
# passes
grouperClient.discoveryDefault.ldap.loadBalancing = active/active
#grouperClient.discoveryOverride.ldap.loadBalancing = active/active

# if we are active/active, then the same connection will
# be used for a certain number of seconds.  If this is -1, then
# always keep the same server (unless errors)
grouperClient.discoveryDefault.ldap.affinitySeconds = 28800
#grouperClient.discoveryOverride.ldap.affinitySeconds = 28800

# if a connection has more errors than another, it will not be
# used until this error timeout passes (unless the other is throwing errors
# too)
grouperClient.discoveryDefault.ldap.lowerConnectionPriorityOnErrorForMinutes = 3
#grouperClient.discoveryOverride.ldap.lowerConnectionPriorityOnErrorForMinutes = 3

# when a connection is attempted, this is the timeout that it will use before trying
# another connection
grouperClient.discoveryDefault.ldap.timeoutSeconds = 30
#grouperClient.discoveryOverride.ldap.timeoutSeconds = 30

# after all connections have been attempted, it will wait for this long
# to see if any finish
grouperClient.discoveryDefault.ldap.extraTimeoutSeconds = 15
#grouperClient.discoveryOverride.ldap.extraTimeoutSeconds = 15

# urls of web service, should include everything up to the first resource to access
# this is for read or write operations
# add more properties and increment the integer (.1, .2, etc)
# e.g. http://groups.school.edu:8090/grouper-ws/servicesRest
# e.g. https://groups.school.edu/grouper-ws/servicesRest
grouperClient.discoveryDefault.webService.readWrite.0.url =
#grouperClient.discoveryOverride.webService.readWrite.0.url =

# url of web service, should include everything up to the first resource to access
# this is for only read operations
# add more properties and increment the integer (.1, .2, etc)
# e.g. http://groups.school.edu:8090/grouper-ws/servicesRest
# e.g. https://groups.school.edu/grouper-ws/servicesRest
grouperClient.discoveryDefault.webService.readOnly.0.url =
#grouperClient.discoveryOverride.webService.readOnly.0.url =

# active/active or active/standby
# active/active will pick a server randomly,
# and will stick with it for as long as the affinity is set
# active/standby will always use the first connection
# if no errors, then try the second one etc.
# if a connection has more errors and has a higher priority,
# then it will not be tried again until the
# takeConnectionOutOfPoolOnErrorForSeconds timeout
# passes
grouperClient.discoveryDefault.webService.loadBalancing = active/active
#grouperClient.discoveryOverride.webService.loadBalancing = active/active

# if you want to always try read/write before readOnly (i.e. if you are
# worried about if you make a write and read right after each other)
grouperClient.discoveryDefault.webService.preferReadWrite = true
#grouperClient.discoveryOverride.webService.preferReadWrite = true

# if we are active/active, then the same connection will
# be used for a certain number of seconds.  If this is -1, then
# always keep the same server (unless errors)
grouperClient.discoveryDefault.webService.affinitySeconds = 28800
#grouperClient.discoveryOverride.webService.affinitySeconds = 28800

# if a connection has more errors than another, it will not be
# used until this error timeout passes (unless the other is throwing errors
# too)
grouperClient.discoveryDefault.webService.lowerConnectionPriorityOnErrorForMinutes = 3
#grouperClient.discoveryOverride.webService.lowerConnectionPriorityOnErrorForMinutes = 3

# when a connection is attempted, this is the timeout that it will use before trying
# another connection
grouperClient.discoveryDefault.webService.timeoutSeconds = 60
#grouperClient.discoveryOverride.webService.timeoutSeconds = 60

# after all connections have been attempted, it will wait for this long
# to see if any finish
grouperClient.discoveryDefault.webService.extraTimeoutSeconds = 30
#grouperClient.discoveryOverride.webService.extraTimeoutSeconds = 30
```

Logging

If you set the client to debug mode (--debug=true), or in log4j (if you are using the client as a library in an application or you include the log4j jar), log for these packages to debug

```
log4j.logger.edu.internet2.middleware.grouperClient.failover = DEBUG
log4j.logger.edu.internet2.middleware.grouperClient.ws.GrouperClientWs = DEBUG
log4j.logger.edu.internet2.middleware.grouperClient.discovery = DEBUG
log4j.logger.edu.internet2.middleware.grouperClient.util.GrouperClientLdapUtils = DEBU
```

you will see logs like this (well, these are with the --debug=true, if you use log4j you will see the log4j format which should include a date stamp...):

```
DEBUG: method: FailoverClient.instanceMapFromType, read failover state from file: C:
\mchyzer\grouper\trunk\grouperClient\grouperClientFailoverState.bin, success: true, failoverClient
discoveryClient hash: 1ded0fd, failoverClient discoveryClient cache hash: 16a9d42, failoverClient
discoveryClient affinity: null, failoverClient grouperLdap hash: 7a84e4, failoverClient grouperLdap cache hash:
1aaa14a, failoverClient grouperLdap affinity: null
DEBUG: method: FailoverClient.orderedListOfConnections, connectionType: discoveryClient, failoverClient hash:
1ded0fd, failoverClient cache hash: 16a9d42, affinityConnection: null, connection.0: http://localhost:8091
/grouper-ws/, errors: 0, connection.1: http://localhost:8091/grouper-ws2/, errors: 0, Setting affinity:
http://localhost:8091/grouper-ws2/, affinityConnectionPost: http://localhost:8091/grouper-ws2/
DEBUG: method: FailoverClient.internal_failoverLogic, type: discoveryClient, connections: 2, Wait for 50 secs:
0: http://localhost:8091/grouper-ws2/, id: Q8YOQLTX, Try 0 thread conn: http://localhost:8091/grouper-ws2/, id:
Q8YOQLTX, Finish 0 thread conn in 1869ms: http://localhost:8091/grouper-ws2/, id: Q8YOQLTX, Success: 0:
http://localhost:8091/grouper-ws2/, id: Q8YOQLTX, add to affinity cache
DEBUG: method: FailoverClient.failoverLogic, saveStateEverySeconds: 0, savingStateToFile: C:
\mchyzer\grouper\trunk\grouperClient\grouperClientFailoverState.bin
DEBUG: method: DiscoveryClient.retrieveFile, existsInDiscoveryFilecache: false, existsInFilecache: true,
fileIsYoungEnough: false, fileFromServer: true, fileFound: true, fileSizeBytes: 4244, lastModified: Wed Feb 15
08:31:32 EST 2012
DEBUG: method: GrouperClientWs.configureFailoverClient, needsReconfigure: true, discoveryFile: C:
\mchyzer\grouper\trunk\grouperClient\grouper.client.discovery_20120215_083130_805_Q8YOQLTY.properties,
needsReconfigureFile: true, readWriteUrl.0: http://localhost:8091/grouper-ws/servicesRest, readWriteUrl.1:
http://localhost:8091/grouper-ws2/servicesRest, readOnlyUrl.0: http://localhost:8091/grouper-ws3/servicesRest,
affinitySeconds: 600, extraTimeoutSeconds: 30, errorsForMinutes: 3, failoverStrategy: activeActive,
preferReadWrite: true, timeoutSeconds: 60
DEBUG: method: FailoverClient.orderedListOfConnections, connectionType: grouperWsReadOnly, failoverClient hash:
d70d7a, failoverClient cache hash: b5f53a, affinityConnection: null, connection.0: http://localhost:8091
/grouper-ws/servicesRest, errors: 0, connection.1: http://localhost:8091/grouper-ws2/servicesRest, errors: 0,
connection.2: http://localhost:8091/grouper-ws3/servicesRest, errors: 0, Setting affinity: http://localhost:8091
/grouper-ws2/servicesRest, affinityConnectionPost: http://localhost:8091/grouper-ws2/servicesRest
DEBUG: method: FailoverClient.internal_failoverLogic, type: grouperWsReadOnly, connections: 3, Wait for 78
secs: 0: http://localhost:8091/grouper-ws2/servicesRest, id: Q8YOQLTZ, Try 0 thread conn: http://localhost:8091
/grouper-ws2/servicesRest, id: Q8YOQLTZ, Finish 0 thread conn in 50794ms: http://localhost:8091/grouper-ws2
/servicesRest, id: Q8YOQLTZ, Success: 0: http://localhost:8091/grouper-ws2/servicesRest, id: Q8YOQLTZ, add to
affinity cache
DEBUG: method: FailoverClient.failoverLogic, saveStateEverySeconds: 0, savingStateToFile: C:
\mchyzer\grouper\trunk\grouperClient\grouperClientFailoverState.bin
```

WS client failover protocol

Configure read/write and readonly URLs.

Identify at least a primary and a secondary URL (note that if there are non-readonly queries, then the readwrite should probably be the primary URL). Or the client could rotate servers for all readonly, or a DNS load balancer could do this for one URL.

Try a server (e.g. the primary), and look at the result code. If it is a SUCCESS, or something like INVALID_QUERY, then trust the response. If there is no response, or it is an error, then if appropriate (e.g. if readonly and the secondary is readonly), try another URL (e.g. the secondary). Note there should be a configurable timeout.

See the Grouper failover client documentation for more information about when to change which connection to try first due to errors, active/active and active/standby loadbalancing, connection affinity, timeouts, extra timeouts, startup time, etc.

LDAP for always availability

From Jim Fox:

> *"I am compelled to point out that there is another way to implement 'always availability', namely with LDAP replicas. They are fast and easy to make redundant. And they easily support the two most common queries that need to be always available: effective membership of a group and effective memberships for a user. I'd go so far as to suggest that an LDAP ought to be a standard component of any Grouper installation. Do that, with a standardized LDAP schema, and you have, almost automatically, instant updates of the LDAP and easy 'always availability' of the web service."*

Yes, if you can satisfy your availability requirements with LDAP, go for it, you will not need this component. If your LDAP does not have the availability you need, or if you need improved availability for features that LDAP does not support (e.g. if an LDAP server goes down you might have a minute of unavailability, permission information (might not be provisioned to ldap, or maybe you want to use server processed limits), etc) then maybe you need this feature...

See Also

Grouper Client

Grouper Failover Client

Info on how this feature is used at Penn