Shib2IdpCluster

Shibboleth 2.1 IdP klaszterezése

Klaszter terminológia

Node

- egy, a szolgáltatást futtató csomópont

Klaszter

- kívülről nem megkülönböztethető nodeok összessége

Szerver

- fizikai (vagy virtuális) gép, amely a nodeokat futtatja (egy gépen lehet több node is)

Failover

- amennyiben egy csomópont kiesik, egy másik csomópont automatikusan és transzparensen átveszi a munkáját

High availability

- amennyiben egy csomópont kiesik, nem veszhet el adat, a kliensek nem veszik észre a kiesést

Load balancing

- a terhelés elosztása az egyes csomópontok között

Terracotta

A Shibboleth2 IdP a Terracotta szoftvert támogatja a klaszter építéséhez. A Terracotta képes arra, hogy a különböző nodeokon futó Shibboleth IdP-k között a session és egyéb információkat (például artifact map, authnrequest replay map, stb.) szinkronban tartsa.

A Terracotta kliens-szerver architektúrában működik. A kliensek a JVM-ben futó instrumentált osztályokból, osztálybetöltőből és zármenedzserből állnak, a szerverek pedig biztosítják a klaszterezett adatok perzisztens tárolását és a csomópont-független elérést. Amennyiben egy JVM-ben szükség van egy távoli JVM-ben létrehozott klaszterezett objektumra, akkor ezt az első elérésnél a Terracotta kliens elkéri a távoli szervertől. Emiatt teljesítmény okokból érdemes az azonos felhasználóhoz tartozó kéréseket mindig ugyanahhoz a csomóponthoz küldeni. A HTTP-Artifact profil használata esetén ez nem garantálható, ezért ajánlott a HTTP-Post profil használata.

Shibboleth IdP és Terracotta

A Shibboleth IdP-ben a következő adatok klaszterezését kell megvalósítani: artifact, session, replay, transientId, loginContext. Ezeket az adatokat a Shibboleth StorageService tárolja.

A Terracotta telepítéséhez és beállításához ez a wikioldal nyújt segítséget.

1. Terracotta letöltése, kicsomagolása
2. tűzfalbeállítások (TCP/9510 kliens->szerver és TCP/9530 szerver->szerver portok engedélyezése)
3. minden csomóponton azonos JVM verziót használjunk a Terracotta szerverben és kliensben is
4. tim-vector integrációs modul telepítése
5. Shibboleth IdP-hez Terracotta konfiguráció szerkesztése [[Shib2IdpTerracottaConfiguration]] alapján
   - csomópontok definiálása (szervernév, hosztnév, logok helye)
6. terracotta szerver futtatása
7. boot jar készítése (Terracotta kliens)
8. boot jar használata a webkonténer JVM-nél
9. **FONTOS: JVM frissítés után újra kell generálni a boot jart!**

JVM beállítások:

```
JAVA_OPTS="-Dtc.install-root=$TC_INSTALL_DIR \
          -Dtc.config=$SHIB_IDP_HOME/conf/tc-config.xml \
          -Xbootclasspath/p:$TC_INSTALL_DIR/lib/dso-boot/dso-boot-hotspot_$jvm_spec_ver.jar"
```

2.1.2 -es IdP verzióhoz a következő konfigurációs rész is szükséges az instrumented-classes szekcióba:

```
<include>
  <class-expression>org.opensaml.xml.util.LazyList</class-expression>
  <honor-transient>true</honor-transient>
</include>
```

A Terracotta log- és adatfájljainak /var alatti tárolását a következőképpen végezhetjük el:

Könyvtárak létrehozása megfelelő jogosultságokkal

```
mkdir /var/{lib,log}/terracotta/{client,server}
chown tomcat55.nogroup /var/{lib,log}/terracotta/client
chown nobody.nogroup /var/{lib,log}/terracotta/server
```

Terracotta tc-config.xml -ben a data,stats,logs opciók átírása értelem szerint.

Magas rendelkezésre állás beálítása

A következő konfigurációs részt a tc-config.xml elején kell elhelyezni. Ez engedélyezi a kliens-szerver és szerver-szerver újrakapcsolódást, ezzel kivédve az apró hálózati kimaradások okozta problémákat. Sajnos mindkét beállítás megnöveli a failover idejét.

```
<tc-properties>
 <!-- server-to-server reconnect -->
 <property name="l2.nha.tcgroupcomm.reconnect.enabled" value="true" />
 <property name="l2.nha.tcgroupcomm.reconnect.timeout" value="15000" />
 <!-- client-to-server reconnect -->
 <property name="l2.l1reconnect.enabled" value="true" />
 <property name="l2.l1reconnect.timeout.millis" value="15000" />
</tc-properties>
```

Debian initszkript a Terracotta szerver indításához

- /etc/init.d/terracotta néven mentsük el root tulajdonossal a következő szkriptet, majd adjunk rá execute jogot:

```
#!/bin/sh
TC_USER=nobody
TC_GROUP=nogroup
PIDFILE=/var/run/terracotta.pid

if [`id -u` -ne 0 ](); then
        echo "You need root privileges to run this script"
        exit 1
fi

if [-f /etc/default/terracotta ](); then
    . /etc/default/terracotta
fi

if [-z "$TC_INSTALL_DIR" -o ! -d "$TC_INSTALL_DIR" ](); then
    echo "No TC_INSTALL_DIR specified or invalid TC_INSTALL_DIR"
    exit 1
fi
if [-z "$TC_CONFIG_PATH" -o ! -f "$TC_CONFIG_PATH" ](); then
    echo "No TC_CONFIG_PATH specified or invalid TC_CONFIG_PATH"
    exit 1
fi
if [-z "$NODENAME" ](); then
    echo "No NODENAME specified"
    exit 1
fi

export JAVA_HOME

JAVA_OPTS="-server -Xms512m -Xmx512m -XX:+HeapDumpOnOutOfMemoryError $JAVA_OPTS"
TC_START_OPTS="$JAVA_OPTS \
   -Dcom.sun.management.jmxremote \
```

```
   -Dtc.install-root=$TC_INSTALL_DIR \
   -cp $TC_INSTALL_DIR/lib/tc.jar \
   com.tc.server.TCServerMain -n $NODENAME -f $TC_CONFIG_PATH"
TC_STOP_OPTS="-Dtc.install-root=$TC_INSTALL_DIR \
  -cp $TC_INSTALL_DIR/lib/tc.jar \
  com.tc.admin.TCStop -n $NODENAME"

. /lib/lsb/init-functions
. /etc/default/rcS

check_stopped () {
    return `/sbin/start-stop-daemon --test --start --pidfile "$PIDFILE" \
        --user $TC_USER --startas "$JAVA_HOME/bin/java" \
        >/dev/null`
}
start () {
    log_daemon_msg "Starting Terracotta server node ($NODENAME)..."

    if check_stopped; then

        /sbin/start-stop-daemon -S -v --make-pid --pidfile "$PIDFILE" \
            --chuid $TC_USER:$TC_GROUP --background \
            --exec $JAVA_HOME/bin/java -- $TC_START_OPTS

    else
        log_progress_msg "(already running)"
    fi
    log_end_msg 0
}
stop () {
    log_daemon_msg "Stopping Terracotta server node ($NODENAME)..."
    if $JAVA_HOME/bin/java $TC_STOP_OPTS; then
        log_progress_msg "(shutdown command sent)"
    else
        log_progress_msg "(could not send shutdown command)"
    fi
    sleep 5
    if check_stopped; then
        log_progress_msg "(cleaning persistent store)"
        rm -r /var/lib/terracotta/server/*
```

```
        log_end_msg 0
    else
        log_progress_msg "(stopping in progress)"
    fi
}
force_stop () {
    log_daemon_msg "Killing Terracotta server node ($NODENAME)..."
    /sbin/start-stop-daemon -K --pidfile $PIDFILE -x $JAVA_HOME/bin/java
}

case "$1" in
  start)
      start
      ;;
  stop)
      stop
      ;;
  force-stop)
      force_stop
      ;;
  restart)
      stop
      sleep 10
      start
      ;;
  *)
    echo "Usage terracotta start|stop|force-stop|restart"
    exit 1;;
esac

exit $?
```

- A konfiguráció az /etc/default/terracotta fájlban található:

```
NODENAME=terracotta-node-name
TC_CONFIG_PATH=/path/to/shibboleth-idp/conf/tc-config.xml
JAVA_HOME=/path/to/jvm
TC_INSTALL_DIR=/path/to/terracotta
```

Monitoring (JMX, Munin, Nagios)

- JMX: Java Management Extensions
- A Terracotta támogatja a JMX-en keresztüli monitorozást és bevatkozást
  - alapértelmezésképp jmxmp protokollon keresztül
    - másoljuk be a jmxremote_optional.jar -t a terracotta lib/ könyvtárából egy üres könyvtárba
    - indítsuk a jconsole-t a következő paranccsal: *jconsole -J-Djava.endorsed.dirs=.*
    - kapcsolódjunk a `service:jmx:jmxmp://<tc-szerver-node>:9520` url-re
  - usernév/jelszavas authentikáció esetén rmi protokollon keresztül is elérhetjük a tc szervert
- Check_jmx szkriptek nagioshoz és muninhoz

Fontosabb Terracotta MBean attribútumok

- org.terracotta:type=Terracotta Server,name=DSO
  - LiveObjectCount (int)
  - ClientLiveObjectCount (string)
- org.terracotta.internal:type=DSO Client,name=Client Transactions,subsystem=Transactions,clients=Clients,node=...
  - AvgModifiedObjectsPerTransaction (int)
  - AvgNewObjectsPerTransaction (int)
  - ObjectCreationRateBySecond (int)
  - ReadTransactionCount (int)
  - WriteTransactionCount (int)
- org.terracotta.internal:type=Terracotta Server,name=Terracotta

Server

```
* Active (bool)
* PassiveStandby (bool)
* PassiveUninitialized (bool)
* HealthStatus (String)
* State (string)
* StartTime (timestamp)
* Shutdownable (bool)
```

Nagios Terracotta szerver check

```
#!/bin/sh

TERRACOTTA_SERVER_NODES="papigw.aai.niif.hu sandbox.aai.niif.hu"
TERRACOTTA_CLIENT_COUNT=2
TERRACOTTA_SERVER_COUNT=2
JAVA_HOME=/usr/lib/jvm/java-1.5.0-sun
TC_HOME=/usr/local/terracotta-2.7.2
TC_MONITORING=/home/hege/terracotta/terracotta-monitoring/terracotta_monitoring.jar

ACTIVE_NODES=""
STANDBY_NODES=""
CLUSTER_STATUS=""

function check_health() {
    TC_HEALTH=`echo "$1" | awk "/$2.health/{print "'$2}'`
    if [ "x$TC_HEALTH" = "xOK" ]; then
        return 0
    else
        echo TERRACOTTA CRITICAL - node $2 down
        exit 2
    fi
}
function check_active_count() {
    ACTIVE_NODES=`echo "$1" | awk '/ACTIVE-COORDINATOR/{print $1}' | sed 's/\.state://'
    ACTIVE_COUNT=`echo "$1" | awk 'BEGIN {cnt=0} /ACTIVE-COORDINATOR/{cnt++} END {print cnt}'`
    CLUSTER_STATUS="$CLUSTER_STATUS, active nodes: $ACTIVE_NODES"

    if [ "0$ACTIVE_COUNT" -eq 1 ]; then
        return 0
    else if [ "0$ACTIVE_COUNT" -gt 1 ]; then
            echo TERRACOTTA CRITICAL - multiple ACTIVE nodes $CLUSTER_STATUS
        else
            echo TERRACOTTA_CRITICAL - no ACTIVE nodes $CLUSTER_STATUS
        fi
        exit 2
    fi
}

function check_standby_count() {
```

```
    STANDBY_NODES=`echo "$1" | awk '/PASSIVE-STANDBY/{print $1}' | sed 's/\.state://'`
    STANDBY_COUNT=`echo "$1" | awk 'BEGIN {cnt=0} /PASSIVE-STANDBY/{cnt++} END {print cnt}'`
    CLUSTER_STATUS="$CLUSTER_STATUS, standby nodes: $STANDBY_NODES"

    if [ "0$STANDBY_COUNT" -eq $(($TERRACOTTA_SERVER_COUNT-1)) ]; then
        return 0
    else
        echo TERRACOTTA CRITICAL - not enough STANDBY node $CLUSTER_STATUS
        exit 2
    fi
}

function check_client_count() {
    CLIENTCOUNT=`echo "$1" | awk '/clientcount/{print $2}'`
    CLIENT_NODES=`echo "$1" | awk '/client.*address/{print $2}'`
    CLUSTER_STATUS="$CLUSTER_STATUS, client nodes: $CLIENT_NODES"
    if [ "0$CLIENTCOUNT" -eq $TERRACOTTA_CLIENT_COUNT ]; then
        return 0
    else
        echo TERRACOTTA WARNING - client count is $CLIENTCOUNT $CLUSTER_STATUS
        exit 1
    fi
}

OUTPUT=`$JAVA_HOME/bin/java -jar $TC_MONITORING $TERRACOTTA_SERVER_NODES 2>/dev/null`

check_active_count "$OUTPUT"
for i in $TERRACOTTA_SERVER_NODES; do
    check_health "$OUTPUT" "$i"
done
check_standby_count "$OUTPUT"
check_client_count "$OUTPUT"

echo TERRACOTTA OK - cluster is running $CLUSTER_STATUS

if [ "0$1" == "0--verbose" -o "0$1" == "0-v" ]; then
        echo -e "\n\n$OUTPUT"
fi
```

Munin plugin a terracotta szerver memóriahasználatának méréséhez

A check_jmx csomagban található jmx_ munin plugin mellé másoljuk oda a jmxremote_optional csomagot (sun.com-ról letölthető), majd végezzük el a következő módosítást:

```
JMXQUERY="java -cp $RDIR/jmxquery.jar:$RDIR/jmxremote_optional-1.0.1_04-b58.jar \
 org.munin.JMXQuery $SERVICE $RDIR/$CONFIGNAME"
```

Ezután a következő konfigurációt hozzuk létre terracotta_objectcount.conf néven:

```
graph_title Terracotta clustered object count
graph_category Terracotta
graph_order terracotta_objectcount

terracotta_objectcount.label cluster object count
terracotta_objectcount.jmxObjectName org.terracotta:type=Terracotta Server,name=DSO
terracotta_objectcount.jmxAttributeName LiveObjectCount
```

Majd symlinkeljük be a jmx_ szkriptet a munin pluginok közé jmx_terracotta_objectcount néven, és adjuk meg a munin-node.conf -ban a jmx hozzáférési paramétereket:

```
[jmx_*]
env.jmxurl service:jmx:jmxmp://localhost:9520
```

Tomcat monitorozása muninnal

A Tomcat JVM JMX elérésének engedélyezéséhez az /etc/default/tomcat5.5 fájlban a következő plusz kapcsolókat kell megadni:

```
CATALINA_OPTS="${CATALINA_OPTS} \
 -Dcom.sun.management.jmxremote \
 -Dcom.sun.management.jmxremote.port=8083 \
 -Dcom.sun.management.jmxremote.ssl=false \
 -Dcom.sun.management.jmxremote.authenticate=false"
```

Ezután a 8083 -as porton jmxrmi protokollon keresztül lehet elérni a menedzsment ágenst. A check_jmx pluginhoz a következő környezeti beállítást kell hozzárendelni az /etc/munin/plugin-conf.d/munin-node fájlban:

```
[jmx_catalina_*]
env.jmxurl service:jmx:rmi:///jndi/rmi://localhost:8083/jmxrmi
```

Terracotta 2.7.3 ismert hibák

Log file flood

Az L2 újracsatlakozás engedélyezése esetén egy elvesztett kapcsolat után hiába áll helyre a helyes működés, a szerver szemetel a logba:

```
2009-06-24 14:48:38,304 [ConnectionEstablisher] WARN
com.tc.net.protocol.transport.ClientMessageTransport -
ConnectionID(0.0e84473d1e744f4db1d539db88633a30): Timeout of 10000 milliseconds occured
2009-06-24 14:48:39,305 [ConnectionEstablisher] INFO
com.tc.net.protocol.transport.ClientMessageTransport -
ConnectionID(0.0e84473d1e744f4db1d539db88633a30): Attaching new connection: [...]
```

Ez egy ismert probléma: http://jira.terracotta.org/jira/browse/CDV-1252 a javítás elkészült, azonban a 2.7.3 -ba még nem került be.

Workaround-ként ki lehet kapcsolni az l2 újracsatlakozást a konfigurációban. Ilyenkor azonban rövid hálózati megszakadás is teljes node újraindítást fog okozni.

Terracotta 3.2.1 ismert hibák

Még nincs :)

Troubleshooting

- Mindig ellenőrizni kell a logfájlokat! Ha egy szerver folyton írja a logfájlját, az általában rossz jel és elárvult kliensre utal ("Could not find communication stack..." üzenet).
- A teljes klaszter újraindítása után kötelező újraindítani a klienseket (Tomcat), ugyanis a Terracotta nem fogja engedni újra csatlakozni. Ezt egyébként a szerver logfájlban jelzi is.
- Nem érdemes egyszerre indítani a két Terracotta szervert, annak könnyen összeakadás lehet a vége, ha nem tudnak dönteni egymás között.

    - ilyenkor az egyik szerver felszólítja a másik szervert a megállásra, ilyenkor azonban a diszken tárolt állapot megmarad, amit egy start parancs kiadása után nem hajlandó felhasználni a szerver processz.
    - a megoldás az initszkript 'restart' parancsa (vagy két egymás utáni start, ugyanis második kísérletre már hajlandó elindulni 'dirty' adatokkal is).
- Az `/etc/nagios/check_terracotta --verbose` parancs a teljes klaszter állapotát visszaadja (szerverek és kliensek is).
- Ha egy probléma nem oldható meg csak az egyik szerver és a kliensek újraindításával, akkor a teljes klasztert újra kell indítani: mindkét szerver leállítása és a perzisztens adatok gondos törlése után egyesével újraindíthatóak a szerverek majd az aktív szerver indulása után a kliensek (Tomcat).

Kliens library

- a Tomcat webkonténerben fut
- a `/var/log/terracotta/client/log*/terracotta-client.log` fájlba logol
- **JVM váltáskor, frissítéskor újra kell generálni!** Ezt a `/usr/local/terracotta/bin/make-boot-jar.sh` szkripttel lehet megtenni, előtte törölni kell a `/usr/local/terracotta/lib/dso-boot` könyvtár tartalmát.

Szerver processz

- külön processzként fut
- a `/var/log/terracotta/server/log*/terracotta-server.log` fájlba logol
- a `/var/lib/terracotta/server/` könyvtárban tárol saját adatokat, amit kézzel történő tiszta újraindítás előtt törölni kell

Nagios riasztások és megoldásuk

- Túl kevés a kliens (client count is xxx)
    - **OK**: a Tomcat kliens megállt vagy megszakadt a kommunikáció a szerverrel.
    - **Megoldás**: a kliens listában nem szereplő Tomcat-et újra kell indítani.
- Nincs passzív node (not enough passive node)
    - **OK**: az egyik Terracotta szerver épp adatot szinkronizál a másiktól és ezért `passive-uninitialized` állapotban van.
    - **Megoldás**: pár percet érdemes várni, amíg a szinkronizáció befejeződik. Ha nem oldódik meg a probléma magától, akkor újra kell indítani a Terracotta szerver processzt.
- Nem elérhető egy node (node xxx is down)
    - **OK**: az egyik Terracotta szerver nem elérhető.
    - **Megoldás**: újra kell indítani.

Adminisztrációs feladatok

JVM frissítése

A következő leírás meglehetősen az NIIF Intézet saját infrastruktúrájára specifikus, de talán más környezetekben is felhasználható.

Lépések összefoglalása

1. Terheléselosztóból a frissítendő node-ot kivenni
2. Tomcat, Terracotta leállítása a frissítendő node-on
3. JVM beállítások (`/etc/java-6-openjdk/security`, ill. esetleg `/usr/lib/jvm/java-6-openjdk/jre/lib/security`) elmentése. Ez fontos azért, mert bizonyos JVM upgrade-ek (legalábbis a múltban) felülírták a tanúsítvány tárat, és ez nehezen javítható hibához vezetett (pl. az LDAP-hoz nem tudott kapcsolódni az IdP)
4. JVM frissítés
5. Boot jar generálás
6. (Ha megváltozott a jar): Tomcat konfigban a boot jar átírása az újra
7. Ellenőrzés, hogy megváltozott-e a cacerts ($JAVA_HOME/lib/security/cacerts). Ha igen, akkor írjuk felül az elmentett változattal
8. Terracotta, **majd utána** Tomcat indítás
9. (Az LVS magától visszateszi a megjavuló klaszter node-ot, de erről nem árt meggyőződni)

Shell parancsok

```
ldir2:~# ipvsadm -d -t idp.niif.hu:8443 -r idp1.aai.niif.hu:8443
ldir2:~# ipvsadm -d -t idp.niif.hu:https -r idp1.aai.niif.hu:https
ldir2:~# watch "ipvsadm -L -t idp.niif.hu:https  && ipvsadm -L -t idp.niif.hu:8443"


idp1:~$ sudo /etc/init.d/tomcat6 stop
idp1:~$ sudo /etc/init.d/terracotta stop
idp1:~$ tar czf security.tgz /etc/java-6-openjdk/security
idp1:~$ sudo aptitude safe-upgrade

idp1:~$ sudo env JAVA_HOME=/usr/lib/jvm/java-6-openjdk
/usr/local/terracotta/platform/bin/make-boot-jar.sh \
-f /etc/shibboleth-idp/tc-config.xml
idp1:~$ sudo vim /etc/default/tomcat6

idp1:~$ sudo /etc/init.d/terracotta start
idp1:~$ sudo /etc/init.d/tomcat6 start
```

Változat #1
cziernorbert hozta létre 2026-04-14 13:22:21 CEST
cziernorbert frissítette 2026-04-14 13:24:27 CEST