Message-oriented Middleware in Python

Advanced Computer Programming

Prof. Luigi De Simone

Summary

- **Argomenti**
  - Comunicazione indiretta
  - Message-oriented middleware e sistemi event-based
  - Modelli PTP e PUB-SUB
  - Apache ActiveMQ
  - Python STOMP
  - Esempio di utilizzo di STOMP in Python

- **Riferimenti**
  - https://activemq.apache.org/
  - https://jasonrbriggs.github.io/stomp.py/

Motivazioni dietro i Message-Oriented Middleware

- Sono di solito utilizzati per **rimpiazzare** quelli basati sul paradigma RPC
  - Le chiamate RPC sono sincrone e possiamo avere problemi di scalabilità
  - Le richieste RPC arretrate rallentano l'intero sistema
  - Soluzione: Utilizzare un **meccanismo asincrono** per incrementare la scalabilità!
- Sono utilizzati per sviluppare sistemi con ***high availability*** e ***scalability***
  - C'è necessità di mantenere il sistema funzionante anche dopo un fallimento di uno o più server
  - Soluzione: Migrare da un server all'altro utilizzando i **broker** nel caso di fallimento di un server

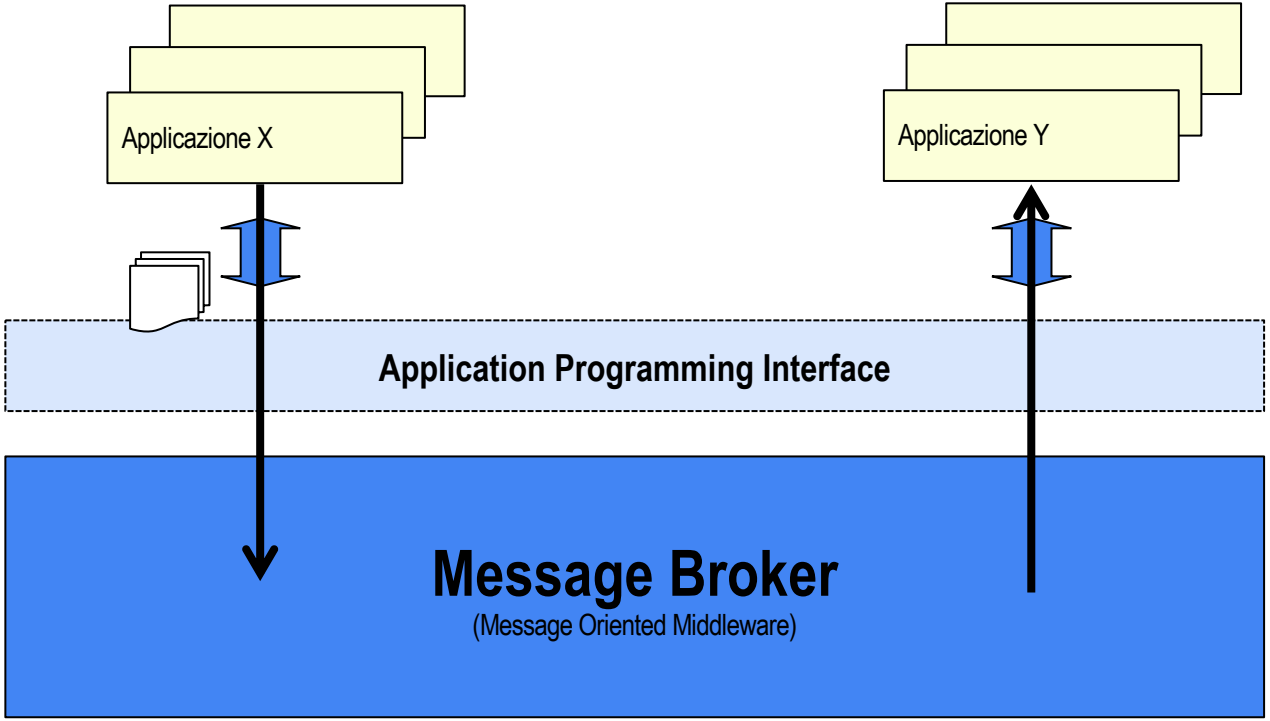
Comunicazione indiretta

- **Comunicazione indiretta**: comunicazione tra entità di un sistema distribuito tramite un intermediario:
  - assenza di **accoppiamento diretto** tra sender e receiver(s).
- La *"natura"* dell'intermediario dipende dallo specifico approccio alla comunicazione indiretta:
  - **group communication**: astrazione di gruppo, un messaggio è mandato ad un gruppo e quindi recapitato ai membri del gruppo;
  - **shared memory:** distributed shared memory, tuple space;
  - **code di messaggi**: astrazione di **coda**, meccanismo point-to-point. Il sender inserisce il messaggio in una coda, che è poi rimosso da *un solo* receiver;
  - **publish-subscribe**: il **publisher** genera messaggi, il **subscriber** esprime interesse per una certa tipologia di messaggi; meccanismo uno-a-molti.

MOM

- Soluzioni middleware basate su *code* o approcci *publish-subscribe* sono note come **message-oriented middleware** (MOM):
  - i sistemi basati su publish-subscribe sono anche noti come **distributed event-based systems.**
- Il MOM gioca il ruolo di **intermediario (*message broker*)** nella comunicazione indiretta basata su messaggi:
  - il MOM garantisce lo scambio di messaggi tra applicazioni con tecniche di di *store-and-forward*;
  - il MOM solleva il programmatore dai dettagli di *basso livello* della comunicazione (per es., RPC e protocolli di rete), ed espone una API *di alto livello* come, ad esempio, le primitive **send/receive message.**

Schema concettuale



Aspetti chiave di un MOM

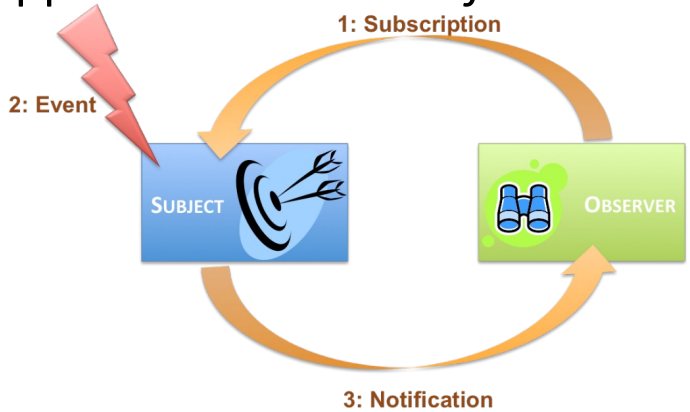
- Recapita il messaggio all'applicazione B che può **risiedere su una macchina differente** rispetto ad A;
- **Gestisce la comunicazione tramite la rete**
  - il MOM può conservare un messaggio finché la rete diventa disponibile, e poi provvede ad inoltrarlo;
- L'applicazione B potrebbe **non** essere in esecuzione quando A invia un messaggio:
  - Il MOM può **conservare** il messaggio finché B diventa disponibile;
  - **L'applicazione A non si blocca** nell'attesa che B riceva il messaggio (**comunicazione asincrona**).

Proprietà di disaccoppiamento

- I message-oriented middleware (MOM) forniscono un meccanismo per l'integrazione **flessibile** e **disaccoppiata** di applicazioni distribuite.
- **Disaccoppiamento spaziale**: il *sender* **non conosce** –o non ha bisogno di conoscere– **l'identità** del *receiver* e viceversa:
  - i partecipanti possono essere sostituiti, migrati, aggiornati, etc.
- **Disaccoppiamento temporale**: il *sender* e il *receiver* possono avere **cicli di vita differenti**:
  - il sender e il receiver non devono essere *necessariamente* in esecuzione allo stesso tempo per poter comunicare.

Evento e notifica

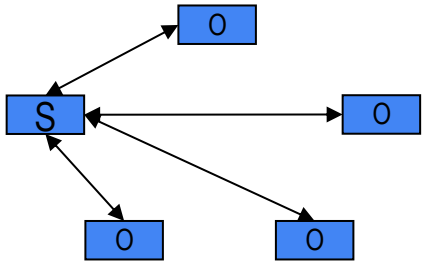
- La **comunicazione indiretta** è fortemente utilizzata per la propagazione (**dissemination**) di **eventi** nei sistemi distribuiti:
  - **Evento:** condizione rilevata da/in una applicazione e comunicata all'intermediario (per es. il MOM) sotto forma di messaggio.
  - **Notifica:** l'atto di informare un insieme di applicazioni dell'occorrenza dell'evento.
- L'approccio *evento-notifica* richiama il pattern **Observer.**



- Un **observer** dichiara interesse agli eventi generati dal/nel **subject** tramite il concetto di **sottoscrizione** (**1**);
- Il **subject** rileva l'occorrenza di un evento (**2**), e **notifica** gli osservatori invocandone un'apposita funzione (**3**).

Ruolo dell'Observer

L'Observer (i) riduce l'accoppiamento soggetto-osservatore, e (ii) supporta la **comunicazione uno-a-molti.**

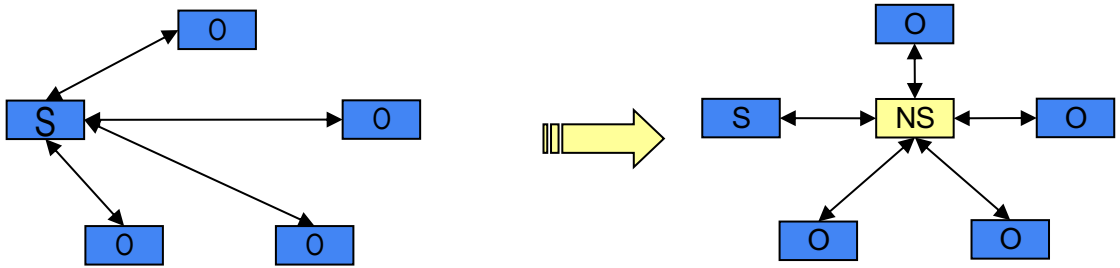


Tale approccio presenta alcuni **inconvenienti**:

- il subject deve mantenere una **lista di observer e relative sottoscrizioni**;
- il subject è **responsabile di notificare gli observer** all'occorrenza di un evento di interesse (**scarsa scalabilità**);
- **gli observer devono conoscere il subject**

Dall'observer al notification service

Per risolvere tali inconvenienti, la responsabilità di gestire gli osservatori/sottoscrizioni e di notificarli non è localizzata nei soggetti, ma delegato ad una terza entità detta **Notification Service (NS)**.



Il *publish-subscribe* non è l'unico modello per realizzare sistemi **event-based**. Il **notification service** può utilizzare code di messaggi:

i) all'occorrenza di un evento l'applicazione *sender* inserisce il messaggio in una **coda**;

ii) il messaggio viene recapitato ai *receiver* sottoscritti alla **coda**.

Sistemi event-based nell'industria

- Esiste un grande interesse in ambito industriale nei sistemi event based e nei **middleware ad eventi** per la realizzazione di sistemi innovativi caratterizzati da una grande scala e criticità

SESAR: *Il nuovo sistema di controllo del traffico aereo in Europa.*

NASPI North American SynchroPhasor Initiative

NASPI: *L'infrastruttura per il monitoraggio e controllo della rete elettrica nel nord America.*

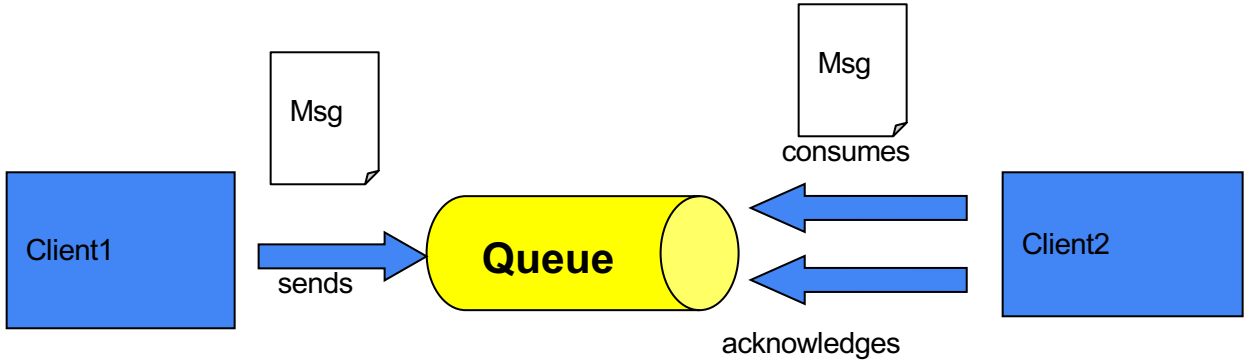
FSE*: Le recenti soluzioni di interconnessione dei sistemi di fascicolo sanitario elettronico in Italia e in Europa.*

Domini di Messaging comuni in MOM

- **Point-to-Point (PTP)**
  - Il dominio di messaging PTP ruota intorno al concetto di **coda di messaggi**;
  - *ogni messaggio ha un solo consumer*

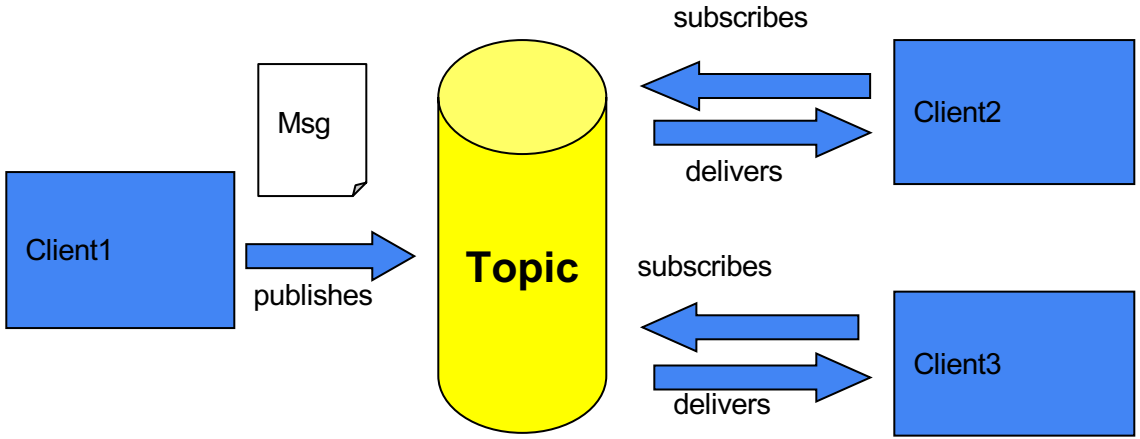
- **Publish-Subscribe**
  - ogni messaggio è associato ad un **topic**;
  - utilizza il concetto di *topic* per l'invio e la ricezione dei messaggi;
  - *ogni messaggio può avere più consumer*

Messaging Point-to-Point



- Una coda **conserva** un determinato **messaggio** finché esso ***non scade*** o ***viene consumato***;
- Il **receiver conferma (*ack*)** la corretta **ricezione** del messaggio.

Messaging Publish/Subscribe



- I ***topic*** **conservano un messaggio** finché esso **non viene rilasciato ai subscriber *correnti***.
- I **messaggi** possono essere **processati** da **0..N subscriber**.

Messaging Protocols

- Esistono diversi **protocolli** utilizzati dai middleware (broker) MOM

- **AMQP** (Advanced Message Queuing Protocol)
  - Progettato per rimpiazzare middleware di messaggistica proprietari esistenti
  - Protocollo binario wire-level (un modo per portare i dati da un punto all'altro) progettato per l'interoperabilità tra diversi fornitori
  - Ancora molto popolare

- **MQTT** (Message Queuing Telemetry Transport)
  - Protocollo di messaggistica publish-subscribe leggero, standard ISO (ISO/IEC PRF 20922), che usa TCP/IP
  - Progettato per situazioni in cui è richiesto un basso consumo energetico e in cui la larghezza di banda è limitata (scenari fog/edge computing, IoT)

- **STOMP** (Simple [or Streaming] Text Oriented Message Protocol)
  - Semplice protocollo basato su testo
  - Progettato per lavorare con middleware orientati ai messaggi (MOM)
  - Fornisce un formato *wire-level* che consente ai client STOMP di parlare con qualsiasi broker di messaggi che supporti il protocollo

Apache ActiveMQ

- Apache ActiveMQ è un popolare broker di messaggi open source, **multiprotocollo** basato su **Java**
- Supporta i protocolli standard del settore (e.g., AMQP, MQTT, STOMP)
- Supporta client scritti in diversi linguaggi come JavaScript, C, C++, **Python**, .Net, etc.

- Scaricabile da: http://activemq.apache.org/components/classic/download/
  - NOTA: **scaricare la versione 5.16.6**

Messaging in Python: STOMP

- *STOMP - Simple (or Streaming) Text Oriented Messaging Protocol*
  - Protocollo FRAME based che assume l'utilizzo di un 2-way streaming network protocol (ad esempio TCP)
  - Client e Server comunicano attraverso dei **STOMP Frame** inviati attraverso lo stream
  - Fornisce un formato interoperabile che consente di poter comunicare con qualsiasi message broker che supporti STOMP
  - Fornisce interoperabilità tra differenti linguaggi, piattaforme e broker

STOMP Frame

```
COMMAND
Header1:value1
Header2:value2

Body^@
```

- **"stomp.py"** è una client library Python per accedere a servizi di messagging attraverso il protocollo *STOMP*
  - Esempi di provider sono: **ActiveMQ**, Artemis e RabbitMQ
  - Può essere eseguito anche in modalità standalone, attraverso la command-line

- **Installazione**
  - `pip install stomp.py`

STOMP: **Connessione al provider**

```
import stomp
conn = stomp.Connection([('127.0.0.1', 62613)])
conn.connect()
```

- `Connection()` accetta svariati parametri*, il principale è ***host_and_ports***:
  - Lista di tuple, dove ogni tupla contiene la coppia **`('IP-ADDRESS', PORT)`** dove il message broker è in ascolto
  - `IP-ADDRESS` può anche essere un indirizzo IPv6
  - La possibilità di prevedere una lista di coppie IP, PORT permette al client di poter effettuare il check delle varie coppie, fin quando una socket connection non viene stabilita con successo

- `connect(username=None, passcode=None, wait=False, headers=None, with_connect_command=False, **keyword_headers)`
  - `username` (str) – specifica l'username per il login
  - `passcode` (str) – specifica la password dell'utente
  - `wait` (bool) – attende che la connessione sia completata prima di ritornare
  - `headers` (dict) – mappa con headers addizionali da inviare con la sottoscrizione
  - `with_connect_command` – se True, usa il comando CONNECT invece di STOMP
  - `keyword_headers` – ulteriori headers da inviare con la sottoscrizione
  - La disconnessione può essere effettuata con il metodo `disconnect()`

* Riferirsi alla documentazione - http://jasonrbriggs.github.io/stomp.py/api.html#establishing-a-connection

STOMP: **Invio dei messaggi**

```
conn.send('/queue/test', 'test message')
```

Dopo aver stabilito la connessione è possibile inviare messaggi con il metodo `send`: che permette l'invio di un messaggio verso una specifica destination definita in un provider:

- `send(destination, body, content_type=None, headers=None, **keyword_headers)`
  - `destination (str)`: destinazione, ad esempio una coda o un topic
  - `body`: contenuto del messaggio
  - `content_type (str)`: MIME type del messaggio
  - `headers (dict)`: headers aggiuntivi da inviare con il messaggio
  - `keyword_headers`: header aggiuntivi richiesti dal provider

STOMP: **Ricezione dei messaggi**

```
conn.set_listener('name', MyListener())
conn.subscribe(destination='/queue/test', id=1, ack='auto')
```

- La ricezione dei messaggi avviene attraverso il setup di un `Listener` e la sottoscrizione ad una destination
- I `Listener` sono sottoclassi della `ConnectionListener` (sono fornite altre classi Listener, ad esempio la `PrintingListener` che stampa tutte le interazioni tra client-server):

- **`set_listener(name, Listner)`**
  - `Name (str)`: nome da assegnare al listener (può essere utilizzato successivamente per rimuovere il listener)
  - `Listener`: istanza di listener da utilizzare

- **`subscribe(destination, id, ack='auto', headers=None, **keyword_headers)`**
  - Permette la sottoscrizione ad una destination
  - `destination (str)`: topic o queue a cui sottoscriversi
  - `id (str)`: identificativo univoco della sottoscrizione
  - `ack (str)`: modalità di acknowledgment, auto, client or client-individual (metodi `ack` e `nack`)
  - `headers (dict)`: mappa di header aggiuntivi da inviare con la sottoscrizione
  - `keyword_headers`: ulteriori header da inviare con la sottoscrizione

STOMP: **Ricezione dei messaggi (Listener)**

```
class MyListener(stomp.ConnectionListener):

    def on_message(self, frame):
        print('received a message "%s"' % frame.body)
```

- Abilitano la ricezione asincrona di messaggi
- Necessario ridefinire il metodo `on_message`, invocato alla ricezione di ogni messaggio sulla destination a cui fa riferimento
- `on_message(frame)`:
  - `frame`: messaggio (Frame STOMP) ricevuto

- `frame.body`: corpo del messaggio
- `frame.headers`: mappa con gli header del messaggio
- `frame.cmd`: comando STOMP command

STOMP: Esempio **comunicazione point-to-point** (queue) con ActiveMQ. *Receiver*

```
# receiver.py
import time, stomp

class MyListener(stomp.ConnectionListener):

    def __init__(self, conn):
        self.conn = conn

    def on_message(self, frame):
        print('received a message "%s"' % frame.body)

if __name__ == "__main__":

    conn = stomp.Connection([('127.0.0.1', 61613)])

    conn.set_listener('', MyListener(conn))

    conn.connect(wait=True)
    conn.subscribe(destination='/queue/test', id=1, ack='auto')

    time.sleep(60)

    conn.disconnect()
```

Porto di default utilizzato da ActiveMQ

STOMP: Esempio **comunicazione point-to-point** (queue) con ActiveMQ. *Sender*

```
# sender.py
import stomp

conn = stomp.Connection([('127.0.0.1', 61613)])
conn.connect(wait=True)

conn.send('/queue/test', 'test message')

conn.disconnect()
```

STOMP: Esempio **comunicazione pub-sub** (topic) con ActiveMQ. *Subscriber*

```
# receiver.py
import time, stomp

class MyListener(stomp.ConnectionListener):

    def __init__(self, conn):
        self.conn = conn

    def on_message(self, frame):
        print('received a message "%s"' % frame.body)

if __name__ == "__main__":

    conn = stomp.Connection([('127.0.0.1', 61613)])

    conn.set_listener('', MyListener(conn))

    conn.connect(wait=True)
    conn.subscribe(destination='/topic/test', id=1, ack='auto')

    time.sleep(60)

    conn.disconnect()
```

Porto di default utilizzato da ActiveMQ

STOMP: Esempio **comunicazione pub-sub** (topic) con ActiveMQ. *Publisher*

```
# sender.py
import stomp

conn = stomp.Connection([('127.0.0.1', 61613)])
conn.connect(wait=True)

conn.send('/topic/test', 'test message')

conn.disconnect()
```

STOMP: **Transazioni**

- Il protocollo STOMP mette a disposizione un metodo per trasmettere i messaggi ad un broker all'interno di una **transazione**
- I messaggi all'interno della transazione sono mantenuti dal server finché non viene effettuato:
  - ***commit*** sulla transazione: in questo caso i messaggi sono effettivamente inoltrati
  - ***abort*** sulla transazione: in questo caso i messaggi sono scartati
- L'avvio di una transazione può essere effettuata con il metodo `begin`
  - ritorna un *transaction id* che va utilizzato nell'invio dei messaggi
  - Possibilità di generare il proprio id e passarlo al metodo come paraemtro

```
# Commit
conn.subscribe('/queue/test', id=5)
txid = conn.begin()
conn.send('/queue/test', 'test1', transaction=txid)
conn.send('/queue/test', 'test2', transaction=txid)
conn.send('/queue/test', 'test3', transaction=txid)
conn.commit(txid)

# Abort
conn.subscribe('/queue/test', id=6)
txid = conn.begin()
conn.send('/queue/test', 'test4', transaction=txid)
conn.send('/queue/test', 'test5', transaction=txid)
conn.abort(txid)
```

Administered objects e ActiveMQ

- E' necessario **avviare** il provider ActiveMQ **prima** dell'esecuzione di un applicativo STOMP
- ActiveMQ si avvia digitando **`activemq start`** da prompt*

```
~/apache-activemq-5.16.6/bin ./activemq start
INFO: Loading '/Users/ldesi/apache-activemq-5.16.6//bin/env'
INFO: Using java '/usr/bin/java'
INFO: Process with pid '98352' is already running

~/apache-activemq-5.16.6/bin
```

* oppure `activemq` in versioni meno recenti di ActiveMQ

Administered objects e ActiveMQ

- Per **terminare** l'istanza di ActiveMQ si può digitare **`activemq stop`** da prompt*

```
~/apache-activemq-5.16.6/bin ./activemq stop                                    ✔ 〈 base ♣ 〈 15:44:47
INFO: Loading '/Users/ldesi/apache-activemq-5.16.6//bin/env'
INFO: Using java '/usr/bin/java'
INFO: Waiting at least 30 seconds for regular process termination of pid '98352' :
Java Runtime: Eclipse Adoptium 20.0.1 /Library/Java/JavaVirtualMachines/temurin-20.jdk/Contents/Home
  Heap sizes: current=67584k  free=65570k  max=1048576k
    JVM args: -Xms64M -Xmx1G -Djava.util.logging.config.file=logging.properties -Djava.security.auth.login.config=/Users/ldesi/apache-activemq-5.16.6
//conf/login.config --add-reads=java.xml=java.logging --add-opens=java.base/java.security=ALL-UNNAMED --add-opens=java.base/java.net=ALL-UNNAMED --add
d-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.util=ALL-UNNAMED --add-opens=java.naming/javax.naming.spi=ALL-UNNAMED --add-opens=
java.rmi/sun.rmi.transport.tcp=ALL-UNNAMED --add-opens=java.base/java.util.concurrent=ALL-UNNAMED --add-opens=java.base/java.util.concurrent.atomic=A
LL-UNNAMED --add-exports=java.base/sun.net.www.protocol.http=ALL-UNNAMED --add-exports=java.base/sun.net.www.protocol.https=ALL-UNNAMED --add-exports
=java.base/sun.net.www.protocol.jar=ALL-UNNAMED --add-exports=jdk.xml.dom/org.w3c.dom.html=ALL-UNNAMED --add-exports=jdk.naming.rmi/com.sun.jndi.url.
rmi=ALL-UNNAMED -Dactivemq.classpath=/Users/ldesi/apache-activemq-5.16.6//conf:/Users/ldesi/apache-activemq-5.16.6//../lib/: -Dactivemq.home=/Users/l
desi/apache-activemq-5.16.6/ -Dactivemq.base=/Users/ldesi/apache-activemq-5.16.6/ -Dactivemq.conf=/Users/ldesi/apache-activemq-5.16.6//conf -Dactivem
q.data=/Users/ldesi/apache-activemq-5.16.6//data
Extensions classpath:
  [/Users/ldesi/apache-activemq-5.16.6/lib,/Users/ldesi/apache-activemq-5.16.6/lib/camel,/Users/ldesi/apache-activemq-5.16.6/lib/optional,/Users/ldes
i/apache-activemq-5.16.6/lib/web,/Users/ldesi/apache-activemq-5.16.6/lib/extra]
ACTIVEMQ_HOME: /Users/ldesi/apache-activemq-5.16.6
ACTIVEMQ_BASE: /Users/ldesi/apache-activemq-5.16.6
ACTIVEMQ_CONF: /Users/ldesi/apache-activemq-5.16.6/conf
ACTIVEMQ_DATA: /Users/ldesi/apache-activemq-5.16.6/data
Connecting to pid: 98352
.Stopping broker: localhost
.. FINISHED

~/apache-activemq-5.16.6/bin                                    ✔ 〈 4s 〈 base ♣ 〈 15:45:18
```

Gestione/supervisione del provider

- ActiveMQ dispone di un'**interfaccia di amministrazione web-based** accessibile all'indirizzo: **http://localhost:8161/**
  - cliccare su "Manage ActiveMQ broker" (credenziali di default admin / admin);

Gestione di code e topic

- Accedendo all'interfaccia web di Apache ActiveMQ, è possibile gestire **code** e **topic:**
  - schede Queues/Topics.